SYBASE iAnywhere

SQL Anywhere® 服务器
SQL 参考

2009 年 2 月

11.0.1 版

版权和商标

目录

关于本手册

主题

本书提供了系统过程和目录（系统表和视图）的参考信息。也介绍了 SQL 语言（搜索条件、语法、数据类型和函数）的 SQL Anywhere 实现。

读者

本书适用于所有 SQL Anywhere 用户。它包括 MobiLink、UltraLite 和 SQL Remote 用户所特别关注的内容。您可以将它与文档集中的其它书籍结合使用。

关于 SQL Anywhere 文档

完整的 SQL Anywhere 文档以四种形式提供，但所包含信息均相同。

- **HTML 帮助** 联机帮助文档包含完整的 SQL Anywhere 文档，其中包括手册和 SQL Anywhere 工具的上下文相关帮助。

  如果使用 Microsoft Windows 操作系统，则联机帮助文档以 HTML 帮助 (CHM) 格式提供。若要访问此文档，请选择 [开始] » [程序] » [**SQL Anywhere 11**] » [文档] » [联机手册]。

  管理工具使用同一联机文档来实现帮助功能。

- **Eclipse** 在 Unix 平台上以 Eclipse 格式提供完整的联机帮助。要访问文档，请从 SQL Anywhere 11 安装的 *bin32* 或 *bin64* 目录下运行 *sadoc*。

- **DocCommentXchange** DocCommentXchange 是一个用于访问和讨论 SQL Anywhere 文档的社区。

  使用 DocCommentXchange 可以执行以下任务：

  - 查看文档
  - 检查是否有用户对文档各部分所做出的阐明
  - 提供建议和修正意见以在将来的版本中为所有用户改进文档

  访问 http://dcx.sybase.com。

- **PDF** 整套 SQL Anywhere 手册会以一组 Portable Document Format (PDF) 文件的形式提供。您必须有 PDF 阅读器才能查看信息。要下载 Adobe Reader，请访问 http://get.adobe.com/reader/。

  若要在 Microsoft Windows 操作系统上访问 PDF 文档，请选择 [开始] » [程序] » [**SQL Anywhere 11**] » 文档 » [联机手册 - PDF 格式]。

  要在 Unix 操作系统上访问 PDF 文档，请使用 Web 浏览器打开 *install-dir/documentation/zh/pdf/index.html*。

关于文档集中的手册

SQL Anywhere 文档由以下手册组成：

- **SQL Anywhere 11 - 简介** 本手册介绍 SQL Anywhere 11，一个提供数据管理和数据交换技术的综合数据包，通过它可以为服务器环境、台式机环境、移动环境以及远程办公环境快速开发由数据库驱动的应用程序。
- **SQL Anywhere 11 - 更改和升级** 本手册介绍 SQL Anywhere 11 以及该软件以前版本中的新功能。
- **SQL Anywhere 服务器 - 数据库管理** 本手册介绍如何运行、管理及配置 SQL Anywhere 数据库。它介绍了数据库连接、数据库服务器、数据库文件、备份过程、安全性、高可用性、使用复制服务器进行复制以及管理实用程序和选项。

- **SQL Anywhere 服务器 - 编程** 本手册介绍如何使用 C、C++、Java、PHP、Perl、Python 和 .NET 编程语言（例如 Visual Basic 和 Visual C#）建立和部署数据库应用程序。其中介绍了各种编程接口，如 ADO.NET 和 ODBC。
- **SQL Anywhere 服务器 - SQL 参考** 本手册提供了系统过程和目录（系统表和视图）的参考信息。也介绍了 SQL 语言（搜索条件、语法、数据类型和函数）的 SQL Anywhere 实现。
- **SQL Anywhere 服务器 - SQL 的用法** 本手册介绍如何设计和创建数据库；如何导入、导出和修改数据；如何检索数据以及如何建立存储过程和触发器。
- **MobiLink - 入门** 本手册介绍基于会话的关系数据库同步系统 MobiLink。MobiLink 技术支持双向复制并且非常适用于移动计算环境。
- **MobiLink - 客户端管理** 本手册介绍如何设置、配置和同步 MobiLink 客户端。MobiLink 客户端可以是 SQL Anywhere 或者 UltraLite 数据库。本手册同时也介绍了 Dbmlsync API，通过它可无缝地将同步集成到 C++ 或 .NET 客户端应用程序中。
- **MobiLink - 服务器管理** 本手册说明如何设置和管理 MobiLink 应用程序。
- **MobiLink - 服务器启动的同步** 本手册介绍 MobiLink 服务器启动的同步，这种功能允许 MobiLink 服务器启动同步或在远程设备上进行操作。
- **QAnywhere** 本手册介绍 QAnywhere，一个用于移动、无线、台式机和膝上型客户端的消息传递平台。
- **SQL Remote** 本手册介绍用于移动计算的 SQL Remote 数据复制系统，此系统支持使用电子邮件或文件传输等间接链接共享 SQL Anywhere 统一数据库和多个 SQL Anywhere 远程数据库之间的数据。
- **UltraLite - 数据库管理和参考** 本手册介绍适用于小型设备的 UltraLite 数据库系统。
- **UltraLite - C 及 C++ 编程** 本手册介绍 UltraLite C 和 C++ 编程接口。利用 UltraLite，可以开发数据库应用程序，并将它们部署到手持式设备、移动设备或嵌入式设备。
- **UltraLite - M-Business Anywhere 编程** 本手册介绍 UltraLite for M-Business Anywhere。利用 UltraLite for M-Business Anywhere，用户可以开发基于 Web 的数据库应用程序，并将它们部署到运行 Palm OS、Windows Mobile 或 Windows 的手持式设备、移动设备或嵌入式设备。
- **UltraLite - .NET 编程** 本手册介绍 UltraLite.NET。利用 UltraLite.NET，您可以开发数据库应用程序，并将它们部署到计算机、手持式设备、移动设备或嵌入式设备。
- **UltraLiteJ** 本手册介绍 UltraLiteJ。利用 UltraLiteJ，可以在支持 Java 的环境中开发和部署数据库应用程序。UltraLiteJ 支持 BlackBerry 智能手机和 Java SE 环境。UltraLiteJ 基于 iAnywhere UltraLite 数据库产品。
- **错误消息** 本手册提供了 SQL Anywhere 错误消息及其诊断信息的完整列表。

文档约定

本节列出了本文档中使用的约定。

操作系统

SQL Anywhere 可以在各种平台上运行。在大多数情况下，该软件在所有平台上的行为都是相同的，但也有变动或限制。这些变动或限制通常基于基础操作系统（Windows、Unix），很少基于特定变型（AIX、Windows Mobile）或版本。

为了简化对操作系统的提及，本文档按如下方式对支持的操作系统进行分组：

- **Windows** Microsoft Windows 系列包括 Windows Vista 和 Windows XP（主要用于服务器、台式计算机和膝上型计算机），以及 Windows Mobile（用于移动设备）。

  除非另外指定，否则当本文档提及 Windows 时，是指所有基于 Windows 的平台，包括 Windows Mobile。

- **Unix** 除非另外指定，否则当本文档提及 Unix 时，是指所有基于 Unix 的平台，包括 Linux 和 Mac OS X。

目录和文件名

大部分情况下，对目录和文件名的引用在所有支持的平台上都是类似的，只需在不同形式之间进行简单的转换。这时需使用 Windows 约定。在细节更为复杂的情况下，文档显示所有相关形式。

下面是文档编写中用于简化目录和文件名的约定：

- **大写和小写目录名** 在 Windows 和 Unix 上，目录和文件名可以包括大写和小写字母。创建目录和文件时，文件系统会保留字母大小写。

  在 Windows 上，对目录和文件的提及*不区分大小写*。混合使用大小写的目录和文件名很常见，但使用所有小写字母来提及目录和文件的形式也很常见。SQL Anywhere 安装包含诸如 *Bin32* 和 *Documentation* 的目录。

  在 Unix 上，对目录和文件的提及*区分大小写*。混合使用大小写的目录和文件名不常见。大多数的目录和文件名全部使用小写字母。SQL Anywhere 安装包含诸如 *bin32* 和 *documentation* 的目录。

  本文档采用 Windows 形式的目录名。大多数情况下，在 Unix 上可以将大小写混合形式的目录名转换成小写字母的等效目录名。

- **分隔目录和文件名的斜线** 文档使用反斜线作为目录分隔符。例如，PDF 格式的文档位于 *install-dir\Documentation\zh\PDF*（Windows 形式）。

  在 Unix 上，用正斜线替换反斜线。PDF 文档位于 *install-dir/documentation/zh/pdf* 下。

- **可执行文件** 文档使用 Windows 约定显示可执行文件名（带有诸如 *.exe* 或 *.bat* 后缀）。在 Unix 上，可执行文件名没有后缀。

  例如，在 Windows 上，网络数据库服务器是 *dbsrv11.exe*。在 Unix 上是 *dbsrv11*。

- ***install-dir*** 在安装过程中，选择 SQL Anywhere 的安装位置。创建环境变量 SQLANY11，用来表示此位置。文档中以 *install-dir* 表示此位置。

  例如，本文档将此文件表示为 *install-dir*\readme.txt。在 Windows 上，这等同于 *%SQLANY11%\readme.txt*。在 Unix 上，这等同于 *$SQLANY11/readme.txt* 或 *${SQLANY11}/readme.txt*。

有关 *install-dir* 缺省位置的详细信息，请参见“SQLANY11 环境变量”一节 《SQL Anywhere 服务器 - 数据库管理》。

- ***samples-dir*** 在安装过程中，选择 SQL Anywhere 随附的示例的安装位置。创建环境变量 SQLANYSAMP11，用来表示此位置。文档中以 *samples-dir* 表示此位置。

  要在 *samples-dir* 中打开 Windows 资源管理器窗口，请在 [**开始**] 菜单中，选择 [**程序**] » [**SQL Anywhere 11**] » [**示例应用程序和项目**]。

  有关 *samples-dir* 缺省位置的详细信息，请参见“SQLANYSAMP11 环境变量”一节 《SQL Anywhere 服务器 - 数据库管理》。

命令提示符和命令 shell 语法

大多数操作系统都提供一种或多种使用命令 shell 或命令提示符来输入命令和参数的方法。Windows 命令提示符包括 Command Prompt（DOS 提示符）和 4NT。Unix 命令 shell 包括 Korn shell 和 bash。每个 shell 都具有一些功能，其能力不仅仅局限于简单命令。这些功能通过特殊字符来驱动。特殊字符和功能随 shell 的不同而不同。如果没有正确使用这些特殊字符，通常会导致语法错误或意外行为。

本文档以普通形式提供命令行示例。如果这些示例中包含 shell 的特殊字符，则命令需要根据特定 shell 进行修改。修改方法不在本文档所述范围之内，但通常是在包含这些特殊字符的参数两旁加上引号，或是在特殊字符前面使用转义字符。

下面是命令行语法的一些示例，不同的平台可能会有不同的形式：

- **括号和大括号** 有些命令行选项需要一个参数，该参数将以列表形式接受详细的值指定。该列表通常用括号或大括号括起来。本文档使用括号。例如：

  ```
  -x tcpip(host=127.0.0.1)
  ```

  如果括号导致出现语法问题，用大括号替代：

  ```
  -x tcpip{host=127.0.0.1}
  ```

  如果两种形式都将产生语法问题，应按照 shell 的要求，用引号将整个参数括起来：

  ```
  -x "tcpip(host=127.0.0.1)"
  ```

- **引号** 如果必须在参数值中指定引号，该引号可能会与用于括参数的引号的传统用法发生冲突。例如，要指定值中包含双引号的加密密钥，则可能必须用引号括起密钥，然后转义嵌入的引号：

  ```
  -ek "my \"secret\" key"
  ```

  在许多 shell 中，密钥的值为 `my "secret" key`。

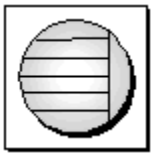
- **环境变量** 本文档介绍设置环境变量。在 Windows shell 中，环境变量使用语法 *%ENVVAR%* 来指定。在 Unix shell 中，环境变量使用语法 *$ENVVAR* 或 *${ENVVAR}* 来指定。

图标

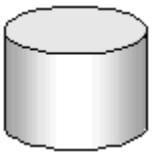
本文档中使用了下列图标。

- 客户端应用程序。

- 数据库服务器，如 Sybase SQL Anywhere。



- 数据库。在某些高水平的图中，可以使用此图标表示数据库和管理该数据库的数据库服务器。



- 复制或同步中间件。用于帮助在数据库之间共享数据。例如 MobiLink 服务器和 SQL Remote 消息代理。

- 编程接口。

联系文档小组

我们欢迎您就本帮助文档提出意见、建议和反馈信息。

要提交意见和建议，请发送电子邮件到 SQL Anywhere 文档小组，地址为 iasdoc@sybase.com。虽然我们不对这些电子邮件进行回复，但您的反馈会帮助我们改进文档，因此我们真诚地欢迎您提出宝贵的意见和建议。

DocCommentXchange

也可以使用 DocCommentXchange 将意见或建议直接置于帮助主题中。DocCommentXchange (DCX) 是一个用于访问和讨论 SQL Anywhere 文档的社区。使用 DocCommentXchange 可以执行以下任务：

- 查看文档
- 检查是否有用户对文档各部分所做出的阐明
- 提供建议和修正意见以在将来的版本中为所有用户改进文档

访问 http://dcx.sybase.com。

查找详细信息并请求技术支持

附加信息和资源可从 Sybase iAnywhere 开发人员社区获得，网址是 http://www.sybase.com/developer/library/sql-anywhere-techcorner。

如果您有问题或是需要帮助，可将邮件发布到下面所列的 Sybase iAnywhere 新闻组。

当您向这些新闻组发布邮件时，请务必提供问题的详细信息，包括 SQL Anywhere 版本的内部版本号。可以通过运行以下命令找到此信息：**dbeng11 -v**。 **dbeng11 -v**.

新闻组位于 *forums.sybase.com* 新闻服务器上。

这些新闻组包括：

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

有关 Web 开发问题，请访问 http://groups.google.com/group/sql-anywhere-web-development。

新闻组免责声明

iAnywhere Solutions 没有义务为其新闻组提供解决方案、信息或建议，除提供系统操作员监控服务和确保新闻组的运行和可用性外，iAnywhere Solutions 也没有义务提供任何其它服务。

如果时间允许，iAnywhere 技术顾问以及其他员工也会对新闻组服务提供帮助。他们是在自愿的基础上提供帮助的，所以可能无法定期提供解决方案和信息。他们可以提供多少帮助取决于他们的工作量。

使用 SQL

本部分介绍 SQL Anywhere SQL 语言，包括数据类型、函数和语句。

SQL 语言元素

目录

关键字

每个 SQL 语句包含一个或多个关键字。SQL 的关键字不区分大小写，但在这些手册中，关键字都以大写字母表示。

例如，在下面的语句中，SELECT 和 FROM 是关键字：

```
SELECT *
   FROM Employees;
```

下列语句等同于上一语句：

```
Select *
   From Employees;
select * from Employees;
sELECT * FRoM Employees;
```

有些关键字只有用双引号引起来才能用作标识符。这些关键字被称为保留字。而其它关键字（如 DBA）则不需要加双引号，不属于保留字。

保留字

SQL 中的一些关键字也是**保留字**。要在 SQL 语句中使用保留字作为标识符，必须用双引号将它引起来。SQL 语句中出现的很多关键字都是保留字，但并非全部都是。例如，必须使用以下语法来检索名为 SELECT 的表的内容。

```
SELECT *
   FROM "SELECT"
```

SQL 关键字不区分大小写，因此以下关键字可能会以大写形式、小写形式或两种形式的任意组合显示。与以下某个字仅在大小写形式上有所不同的所有字符串都是保留字。

如果使用嵌入式 SQL，可以利用数据库库函数 SQL_needs_quotes 确定一个字符串是否需要加上引号。如果一个字符串是保留字，或者包含标识符中通常不允许的字符，则该字符串需要加上引号。

SQL Anywhere 中有以下保留的 SQL 关键字：

add	**all**	**alter**	**and**
any	**as**	**asc**	**attach**
backup	**begin**	**between**	**bigint**
binary	**bit**	**bottom**	**break**
by	**call**	**capability**	**cascade**
case	**cast**	**char**	**char_convert**
character	**check**	**checkpoint**	**close**

comment	commit	compressed	conflict
connect	constraint	contains	continue
convert	create	cross	cube
current	current_timestamp	current_user	cursor
date	dbspace	deallocate	dec
decimal	declare	default	delete
deleting	desc	detach	distinct
do	double	drop	dynamic
else	elseif	encrypted	end
endif	escape	except	exception
exec	execute	existing	exists
externlogin	fetch	first	float
for	force	foreign	forward
from	full	goto	grant
group	having	holdlock	identified
if	in	index	index_lparen
inner	inout	insensitive	insert
inserting	install	instead	int
integer	integrated	intersect	into
is	isolation	join	kerberos
key	lateral	left	like
lock	login	long	match
membership	merge	message	mode
modify	natural	nchar	new
no	noholdlock	not	notify

null	**numeric**	**nvarchar**	**of**
off	**on**	**open**	**openstring**
option	**options**	**or**	**order**
others	**out**	**outer**	**over**
passthrough	**precision**	**prepare**	**primary**
print	**privileges**	**proc**	**procedure**
publication	**raiserror**	**readtext**	**real**
reference	**references**	**refresh**	**release**
remote	**remove**	**rename**	**reorganize**
resource	**restore**	**restrict**	**return**
revoke	**right**	**rollback**	**rollup**
save	**savepoint**	**scroll**	**select**
sensitive	**session**	**set**	**setuser**
share	**smallint**	**some**	**sqlcode**
sqlstate	**start**	**stop**	**subtrans**
subtransaction	**synchronize**	**syntax_error**	**table**
temporary	**then**	**time**	**timestamp**
tinyint	**to**	**top**	**tran**
trigger	**truncate**	**tsequal**	**unbounded**
union	**unique**	**uniqueidentifier**	**unknown**
unsigned	**update**	**updating**	**user**
using	**validate**	**values**	**varbinary**
varbit	**varchar**	**variable**	**varying**
view	**wait**	**waitfor**	**when**
where	**while**	**window**	**with**

with_cube	**with_lparen**	**with_rollup**	**within**
work	**writetext**	**xml**	

另请参见

- “sql_needs_quotes 函数”一节 《SQL Anywhere 服务器 - 编程》

标识符

标识符是数据库中的对象（如用户 ID、表和列）的名称。

注释

标识符的最大长度为 128 个字节。当以下任一条件成立时，标识符需要用双引号引起来或用方括号括起来：

- 标识符包含空格。
- 标识符的首字符不是字母字符（定义将在后面提供）。
- 标识符包含保留字。
- 标识符包含字母和数字以外的其它字符。

字母字符包括字母表中的字母、下划线字符 (_)、at 符号 (@)、井号 (#) 和美元符号 ($)。数据库归类序列指出了哪些字符被视为字母字符或数字字符。

下面字符不允许在标识符中使用：

- 双引号
- 控制字符（任何小于 0x20 的字符）
- 反斜线

如果 quoted_identifier 数据库选项设置为 Off，则双引号可用于分隔 SQL 字符串，而不能用于标识符中。但不管 quoted_identifier 如何设置，您都可以用方括号来界定标识符。对于 Open Client 和 jConnect 连接，quoted_identifier 选项的缺省设置为 Off；其它情况下缺省为 On。

另请参见

- 有关保留字的完整列表，请参见“保留字”一节第 4 页。
- 有关 quoted_identifier 选项的信息，请参见“quoted_identifier 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

示例

以下各种情况均为有效标识符。

- Surname
- "Surname"
- [Surname]
- SomeBigName
- "Client Number"

字符串

字符串是大小可达 2 GB 的字符序列。字符串可以如下方式出现在 SQL 中：

- 作为**字符串文字**。字符串文字是以单引号（撇号）围起的字符序列。字符串文字表示一个特定的常量值，可能含有无法轻易作为字符键入的特殊字符的转义序列。请参见“字符串文字”一节第 11 页。
- 作为数据类型为 CHAR 或 NCHAR 的列或变量的值。
- 作为表达式的计算结果。

字符串长度可以用以下两种方式计量：

- **字节长度**　字节长度是字符串中的字节数。
- **字符长度**　字符长度是字符串中的字符数，它基于所使用的字符集。

对于单字节字符集（例如，cp1252），其字节长度和字符长度相同。对于多字节字符集，字符串的字节长度大于或等于其字符长度。

常量

本节介绍二进制文字和字符串文字。

二进制文字

二进制文字是由数字 0-9 和大小写字母 A-F 组成的十六进制字符序列。将二进制数据作为文字输入时，必须在数据前面加上 0x（零，后面接个 x），该前缀右侧的位数应为偶数。例如，39 的十六进制等效值是 0027，以 0x0027 表示。

形式为 0x12345678 的十六进制常量被视为二进制字符串。0x 后可以添加的位数没有限制。

二进制文字有时也称为二进制常量。在 SQL Anywhere 中，更多地使用二进制文字这一术语。

转换至/自十六进制值

可以使用 CAST、CONVERT、HEXTOINT 和 INTTOHEX 函数将二进制字符串转换为整数。CAST 和 CONVERT 函数可将十六进制常量转换为 TINYINT、有符号和无符号的 32 位整数、有符号和无符号的 64 位整数、NUMERIC 等等。HEXTOINT 函数只将十六进制常量转换为有符号的 32 位整数。

CAST 函数的返回值不能超过 8 位。如果值超过 8 位，将返回一个错误。少于 8 位的值将在其左侧添零。例如，以下参数会返回值 -2,147,483,647：

```
SELECT CAST ( 0x0080000001 AS INT );
```

以下参数会返回错误，因为无法将 10 位数的值表示为有符号的 32 位整数：

```
SELECT CAST ( 0xff80000001 AS INT );
```

如果 HEXTOINT 函数的返回值可以表示为有符号的 32 位整数，则该值可以超过 8 位数。HEXTOINT 函数只接受由数字以及大写或小写字母 A-F 组成的字符串文字或字符串变量（带有或不带有 0x 前缀）。当从右侧数第 8 位数是数字 8-9 以及大写或小写字母 A-F 中的某一个时，或前面的前导数位都是大写或小写字母 F 时，则十六进制值表示负整数。

以下参数会返回值 -2、147、483、647：

```
SELECT HEXTOINT( '0xFF80000001' );

SELECT HEXTOINT( '0x80000001' );

SELECT HEXTOINT ( '0xFFFFFFFFFFFFFFFF80000001' );
```

以下参数会返回错误，因为该参数表示一个正整数值，而该值无法表示为有符号的 32 位整数：

```
SELECT HEXTOINT( '0x0080000001' );
```

另请参见

- “CAST 函数 [Data type conversion]”一节第 141 页
- “CONVERT 函数 [Data type conversion]”一节第 152 页
- “HEXTOINT 函数 [Data type conversion]”一节第 206 页
- “INTTOHEX 函数 [Data type conversion]”一节第 220 页

字符串文字

字符串文字是以单引号围起的字符序列。例如，'Hello world' 是 CHAR 型字符串文字。其字节长度和字符长度均为 11。

字符串文字有时也称为字符串常量、文字字符串或简称为字符串。在 SQL Anywhere 中，更常用字符串文字这一术语。

通过将 N 作为所引值前缀的这种方式可以指定 NCHAR 字符串文字。例如，N'Hello world' 是 NCHAR 型字符串文字。其字节长度和字符长度均为 11。NCHAR 字符串文字中的字节通过 CHAR 数据库的字符集解释，然后转换为 NCHAR。语法 N'string' 是 CAST('string' AS NCHAR) 的缩写形式。

转义序列

有时，您需要在字符串文字中放入一些无法正常键入或输入的字符，例如控制字符（如换行符）、单引号（其它情况下它用于标记字符串文字结束）和十六进制字节值。为此，应使用转义序列。

以下示例说明如何在字符串文字中使用转义序列。

- 单引号用于标记字符串的开始和结束，因此字符串中的单引号必须再使用一个单引号进行转义，如： 'John''s database'
- 十六进制转义序列可用于任何字符或二进制值。十六进制转义序列由反斜线后跟一个 x 再跟两位十六进制数构成。对于 CHAR 和 NCHAR 字符串文字，在 CHAR 字符集中将十六进制值解释为字符。以下示例（在代码页 1252 中）表示数字 1、2 和 3，后跟欧元符号： '123\x80'
- 要表示换行符，请使用一个反斜线后跟一个 n，如： 'First line:\nSecond line:'
- 反斜线用于标记转义序列的开始，因此字符串中的反斜线字符必须再使用一个反斜线进行转义，如： 'c:\\temp'

对于 NCHAR 字符串文字，可以使用与 CHAR 字符串文字相同的字符和转义序列。

如果需要使用无法直接键入字符串文字中的 Unicode 字符，请使用 UNISTR 函数。请参见“UNISTR 函数 [String]”一节第 319 页。

运算符

本节将介绍算术运算符、字符串运算符以及位运算符。有关比较运算符的信息，请参见“搜索条件”一节第 33 页。

正常的运算优先级顺序仍适用。圆括号中的表达式先进行运算，然后是乘除，继而加减。字符串连接在加减之后进行。

有关详细信息，请参见“运算符优先级”一节第 15 页。

比较运算符

比较所使用的语法如下：

expression comparison-operator expression

其中 *comparison-operator* 是以下运算符之一：

运算符	说明
=	等于
>	大于
<	小于
>=	大于或等于
<=	小于或等于
!=	不等于
<>	不等于
!>	不大于
!<	不小于

区分大小写　缺省情况下所创建的 SQL Anywhere 数据库不区分大小写。执行比较时不仅注意所在的数据库，还注意大小写。创建数据库时可以用 -c 选项来控制 SQL Anywhere 数据库是否区分大小写。

有关进行字符串比较时是否区分大小写的详细信息，请参见“初始化实用程序 (dbinit)”一节《SQL Anywhere 服务器 - 数据库管理》。

注意
所有字符串比较都*不区分大小写*，除非创建的数据库区分大小写。

尾随空白　比较字符串时，SQL Anywhere 的行为受创建数据库时所设置的 -b 选项控制。

有关空格填充的详细信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

逻辑运算符

可以使用 AND 或 OR 运算符将搜索条件组合起来。还可以使用 NOT 运算符对它们取非，或使用 IS 运算符测试表达式的计算结果是 true、false 还是 unknown。

- **AND 运算符** AND 运算符按如下所示放在搜索条件之间：

  .. **WHERE** *condition1* **AND** *condition2*

  使用 AND 时，如果两个条件都为 TRUE，则组合条件为 TRUE；如果其中某一条件为 FALSE，则组合条件为 FALSE；其它情况下为 UNKNOWN。

- **OR 运算符** OR 运算符按如下所示放在搜索条件之间：

  .. **WHERE** *condition1* **OR** *condition2*

  使用 OR 时，如果其中某一条件为 TRUE，则组合条件为 TRUE；如果两个条件都为 FALSE，则组合条件为 FALSE；其它情况下为 UNKNOWN。

- **NOT 运算符** NOT 运算符放在某个条件之前可对该条件取非，如下所示：

  .. **WHERE NOT** *condition*

  如果 *condition* 为 FALSE，则 NOT 条件为 TRUE；如果 *condition* 为 TRUE，则 NOT 条件为 FALSE；如果 *condition* 为 UNKNOWN，则 NOT 条件为 UNKNOWN。

- **IS 运算符** IS 运算符放在表达式和要测试的真值之间。IS 运算符的语法如下：

  expression **IS** [**NOT**] *truth-value*

  如果 *expression* 的值为提供的 *truth-value*（必须是 TRUE、FALSE、UNKNOWN 或 NULL 其中之一），则 IS 条件为 TRUE。否则，条件值为 FALSE。

  例如，`5*3=15 IS TRUE` 用于测试表达式 `5*3=15` 的计算结果是否为 TRUE。

另请参见： “三值逻辑”一节第 53 页.

算术运算符

expression + expression 加法。如果其中某一表达式值为 NULL，则结果为 NULL。

expression - expression 减法。如果其中某一表达式值为 NULL，则结果为 NULL。

- expression 取非。如果表达式值为 NULL，则结果为 NULL。

expression * expression 乘法。如果其中某一表达式为 NULL，则结果为 NULL。

expression/expression 除法。如果其中某一表达式为 NULL，或第二个表达式为 0，则结果为 NULL。

expression % expression 模运算结果为两个整数相除后的整数余数。例如，21 % 11 = 10，因为 21 除以 11 等于 1，余数为 10。

标准和兼容性

- **模** SQL Anywhere 支持 % 运算符。

字符串运算符

expression || expression 字符串连接（两个竖线）。如果其中某一字符串为 NULL，则连接时将其视为空字符串。

expression + expression 另一种字符串连接运算方式。使用 + 连接运算符时，必须确保操作数已显式设置为字符数据类型，而不是依赖隐式的数据转换。

例如，以下查询返回整数值 579：

```
SELECT 123 + 456;
```

而以下查询则返回字符串 123456：

```
SELECT '123' + '456';
```

可以使用 CAST 或 CONVERT 函数来显式转换数据类型。

标准和兼容性

- **SQL/2003** || 运算符是 SQL/2003 的字符串连接运算符。

位运算符

在 SQL Anywhere 中，以下运算符可用于整数数据类型和位数组数据类型。

运算符	说明
&	逐位与
|	逐位或
^	逐位异或
~	逐位非

位运算符 &、| 和 ~ 与逻辑运算符 AND、OR 和 NOT 不能互换。

示例

例如，以下语句选择设置了正确位的行。

```
SELECT *
FROM tableA
WHERE ( options & 0x0101 ) <> 0;
```


连接运算符

支持在 FROM 子句中使用表表达式的 SQL/2003 连接语法。请参见“FROM 子句”一节第 582 页。

不建议使用 Transact-SQL 外连接运算符 *= 和 =*。要使用 Transact SQL 外连接，tsql_outer_joins 数据库选项必须设置为 On。请参见“tsql_outer_joins 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

运算符优先级

表达式的运算符优先级如下。列表顶部的运算符先于底部的运算符进行运算。

1. 一元运算符（需要单操作数的运算符）
2. &, |, ^, ~
3. *, /, %
4. +, -
5. ||
6. **not**
7. **and**
8. **or**

在一个表达式中使用一个以上的运算符时，建议使用圆括号显式设置运算顺序。

表达式

表达式是可以通过计算返回值的语句。

语法

expression:
case-expression
| *constant*
| [*correlation-name*.]*column-name*
| **-** *expression*
| *expression operator expression*
| **(** *expression* **)**
| *function-name* **(** *expression*, ... **)**
| *if-expression*
| *special value*
| **(** *subquery* **)**
| *variable-name*

参数

case-expression:
CASE *expression*
WHEN *expression*
THEN *expression*,...
[**ELSE** *expression*]
END

alternative form of case-expression:
CASE
WHEN *search-condition*
THEN *expression*, ...
[**ELSE** *expression*]
END

constant:
integer | *number* | *string* | *host-variable*

special-value:
CURRENT { **DATE** | **TIME** | **TIMESTAMP** }
| **NULL**
| **SQLCODE**
| **SQLSTATE**
| **USER**

if-expression:
IF *condition*
THEN *expression*
[**ELSE** *expression*]
ENDIF

operator:
{ **+** | **-** | ***** | **/** | **||** | **%** }

注释

表达式用于许多不同的位置。

表达式由若干不同种类的元素构成。有关函数和变量的几节将就此进行论述。请参见“SQL 函数”第 117 页和“变量”一节第 64 页。

要计算表达式必须连接到数据库。

副作用

无。

另请参见

- “表达式中的常量”一节第 17 页
- “特殊值”一节第 56 页
- “表达式中的列名称”一节第 17 页
- “SQL 函数”第 117 页
- “表达式中的子查询”一节第 18 页
- “搜索条件”一节第 33 页
- “SQL 数据类型”第 75 页
- “变量”一节第 64 页
- “CASE 表达式”一节第 18 页

标准和兼容性

- 请参见后续章节中对各类表达式的单独说明。

表达式中的常量

常量是数字或字符串文字。字符串常量用撇号围起来（'单引号'）。字符串中的撇号以两个连续的撇号来表示。

表达式中的列名称

列名称是前面带有可选相关名的标识符。相关名通常是表名。有关相关名的详细信息，请参见“FROM 子句”一节第 582 页。

如果列名称含有字母、数字和下划线以外的其它字符，必须用引号（""）将它引起来。例如，以下是有效的列名称：

- Employees.Name
- address
- "date hired"
- "salary"."date paid"

另请参见： “标识符”一节第 8 页.

表达式中的子查询

子查询是一个嵌入在另一个 SELECT、INSERT、UPDATE 或 DELETE 语句中或者其它子查询中的 SELECT 语句。

如果子查询没有任何匹配的行，则计算结果为 NULL。

SELECT 语句必须用圆括号括起来，而且必须包含一个且只能包含一个选择列表项。用作表达式时，子查询通常只能返回一个值。

子查询可以在任何能够使用列名称的地方使用。例如，子查询可以在另一个 SELECT 语句的选择列表中使用。

有关子查询的其它用法，请参见“搜索条件中的子查询”一节第 34 页。

IF 表达式

IF 表达式的语法如下：

IF *condition*
THEN *expression1*
[**ELSE** *expression2*]
{ **ENDIF** | **END IF** }

该表达式的返回方式如下：

- 如果 *condition* 为 TRUE，则 IF 表达式返回 *expression1*。
- 如果 *condition* 为 FALSE，则 IF 表达式返回 *expression2*。
- 如果 *condition* 为 FALSE，且无 *expression2*，则 IF 表达式返回 NULL。
- 如果 *condition* 为 UNKNOWN，则 IF 表达式返回 NULL。

condition 为任何有效的搜索条件。请参见“搜索条件”一节第 33 页。

有关 TRUE、FALSE 和 UNKNOWN 条件的详细信息，请参见“NULL 值”一节第 71 页和“搜索条件”一节第 33 页。

IF 语句不同于 IF 表达式
IF 表达式与 IF 语句不同。有关 IF 语句的信息，请参见“IF 语句”一节第 607 页。

CASE 表达式

CASE 表达式提供条件 SQL 表达式。Case 表达式可以在任何能够使用表达式的地方使用。

CASE 表达式的语法如下：

CASE *expression*
WHEN *expression*

THEN *expression*, ...
[**ELSE** *expression*]
{ **END** | **END CASE** }

如果 CASE 语句后的表达式等于 WHEN 语句后的表达式，则返回 THEN 语句后的表达式。否则，返回 ELSE 语句后的表达式（如果它存在）。

例如，以下代码用 CASE 表达式作为 SELECT 语句的第二个子句。

```
SELECT ID,
   ( CASE Name
      WHEN 'Tee Shirt' then 'Shirt'
      WHEN 'Sweatshirt' then 'Shirt'
      WHEN 'Baseball Cap' then 'Hat'
      ELSE 'Unknown'
   END ) as Type
FROM Products;
```

也可以使用以下语法：

CASE
WHEN *search-condition*
THEN *expression*, ...
[**ELSE** *expression*]
END [**CASE**]

如果满足 WHEN 语句后的搜索条件，则返回 THEN 语句后的表达式。否则，返回 ELSE 语句后的表达式（如果它存在）。

例如，以下语句使用 CASE 表达式作为 SELECT 语句的第三子句，以将字符串与搜索条件相关联。

```
SELECT ID, Name,
   ( CASE
      WHEN Name='Tee Shirt' then 'Sale'
      WHEN Quantity >= 50  then 'Big Sale'
      ELSE 'Regular price'
   END ) as Type
FROM Products;
```

用于简写 CASE 表达式的 NULLIF 函数

NULLIF 函数提供了一种方法来以短格式编写某些 CASE 语句。NULLIF 的语法如下：

NULLIF (*expression-1*, *expression-2*)

NULLIF 比较两个表达式的值。如果第一个表达式的值与第二个表达式的值相等，NULLIF 返回 NULL。如果它们不相等，NULLIF 则返回第一个表达式。

CASE 语句不同于 CASE 表达式
不要将 CASE 表达式和 CASE 语句的语法相混淆。有关 CASE 语句的信息，请参见“CASE 语句”一节第 402 页。

正则表达式概述

正则表达式是一个字符、通配符或运算符的序列，用于定义在字符串内进行搜索所使用的模式。SQL Anywhere 支持将正则表达式作为 SELECT 语句的 WHERE 子句中 REGEXP 或 SIMILAR TO 搜索条件的一部分，或将其作为 REGEXP_SUBSTR 函数的参数。LIKE 搜索条件不支持正则表达式，尽管使用 LIKE 时所指定的某些通配符和运算符与正则表达式的通配符和运算符类似。

以下 SELECT 语句使用正则表达式 (`(K|C[^h])%`) 搜索 Contacts 表并返回姓氏以 K 或 C（但非 Ch）开头的联系人：

```
SELECT Surname, GivenName
   FROM Contacts
   WHERE Surname SIMILAR TO '(K|C[^h])%';
```

正则表达式可包括附加的语法来指定组合、量化、断言和替代，如下所述。

- **组合** 组合允许您将各个正则表达式部分组合在一起，以应用某种附加的匹配条件。例如，`'(abc){2}'` 匹配 abcabc。

  您还可以使用组合来控制表达式各部分的计算顺序。例如，`'ab(cdcd)'` 首先查找 cdcd 的出现，然后判断 cdcd 实例是否存在前缀 ab。

- **量化** 量化允许您控制表达式的前面部分可以出现的次数。例如，问号 (?) 是一个匹配零或前面字符的一个实例的量词。这样，`'honou?r'` 会与 honor 和 honour 都匹配。

- **断言** 通常，搜索一个模式就会返回该模式。断言允许您测试某个模式是否存在，而无需将该模式作为返回内容的一部分。例如，对于 `'SQL(?= Anywhere)'`，仅当 SQL 后跟空格和 Anywhere 时才会匹配 SQL。

- **替代** 替代允许您在无法找到前面的模式时指定替代模式来进行搜索。替代模式从左到右进行计算，找到第一个匹配时即停止搜索。例如，`'col(o|ou)r'` 会查找 color 的实例。如果未找到任何实例，则改为搜索 colour。

另请参见

- “搜索条件”一节第 33 页
- “LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页
- “REGEXP 搜索条件”一节第 43 页
- “SIMILAR TO 搜索条件”一节第 44 页
- “REGEXP_SUBSTR 函数 [String]”一节第 263 页

正则表达式语法

通过 SIMILAR TO 和 REGEXP 搜索条件以及 REGEXP_SUBSTR 函数支持正则表达式。对于 SIMILAR TO，正则表达式语法符合 ANSI/ISO SQL 标准。对于 REGEXP 和 REGEXP_SUBSTR，正则表达式的语法和支持符合 Perl 5。

REGEXP 和 SIMILAR TO 使用正则表达式是与*字符串*相匹配，而 REGEXP_SUBSTR 使用正则表达式则是与*子串*相匹配。要实现 REGEXP 和 SIMILAR TO 的子串匹配行为，可在要尝试匹配的模式的任何一侧指定通配符。例如，`REGEXP '.*car.*'` 会与 car、carwash 和 vicar 匹配。或者，

可重写查询以使用 REGEXP_SUBSTR 函数。请参见“REGEXP_SUBSTR 函数 [String]”一节第 263 页。

通过 SIMILAR TO 匹配的正则表达式不区分大小写，也不区分重音。REGEXP 和 REGEXP_SUBSTR 不受数据库区分重音和大小写的影响。请参见“LIKE、REGEXP 和 SIMILAR TO：字符比较上的差异”一节第 37 页。

正则表达式：元字符

元字符是在正则表达式中具有特殊含义的符号或字符。

元字符的处理视以下情况而异：

- 正则表达式是与 SIMILAR TO 或 REGEXP 搜索条件一起使用，还是与 REGEXP_SUBSTR 函数一起使用
- 元字符是否在正则表达式的字符类的内部

在继续之前，应了解**字符类**的定义。字符类是一组括在方括号内的字符，将根据这组字符对字符串中的字符进行匹配。例如，在 `SIMILAR TO 'ab[1-9]'` 语法中，[1-9] 就是一个字符类，它与 1 到 9 范围中（包括 1 和 9）的某一数字匹配。正则表达式中元字符的处理方式各不相同，这取决于元字符是否被放在字符类的内部。具体来说，当元字符放在字符类的内部时，多数元字符作为常规字符来处理。

对于 SIMILAR TO（仅限于 SIMILAR TO），元字符 *、?、+、_、|、(、)、{ 必须在字符类内进行转义。

要在字符类中包括减号 (-)、脱字符 (^) 或直角方括号 (]) 字符，必须将字符转义。

下面给出了所支持的正则表达式元字符的列表。当 SIMILAR TO、REGEXP 和 REGEXP_SUBSTR 使用元字符时，几乎所有元字符的处理方式都相同：

字符	其它信息
[和]	左右方括号用于指定**字符类**。字符类是进行匹配时所要依据的一组字符。 除连字符 (-) 和脱字符 (^) 外，在字符类中指定的元字符和量词（如 * 和 {m}，分别为元字符和量词）没有特殊意义，可当作实际字符进行运算。 SQL Anywhere 还支持子字符类，例如 POSIX 字符类。请参见“正则表达式：特殊子字符类”一节第 23 页。
*	星号可用于与字符匹配 0 次或多次。例如，`REGEXP '.*abc'` 匹配的字符串以 abc 结尾并以任何前缀开头。因此，aabc、xyzabc 和 abc 匹配，但 bc 和 abcc 则不匹配。
?	问号可用于与字符匹配 0 次或 1 次。例如，`'colou?r'` 匹配 color 和 colour。
+	加号可用于与字符匹配 1 次或多次。例如，`'bre+'` 匹配 bre 和 bree，但不匹配 br。

字符	其它信息
-	可以在字符类中使用一个连字符来表示一个范围。例如，`REGEXP '[a-e]'` 匹配 a、b、c、d 和 e。 有关 REGEXP 和 SIMILAR TO 如何对范围求值的详细信息，请参见“LIKE、REGEXP 和 SIMILAR TO：字符比较上的差异”一节第 37 页。
%	百分号可与 SIMILAR TO 配合使用来匹配任意数目的字符。 不将百分号视为 REGEXP 和 REGEXP_SUBSTR 所使用的元字符。当指定时，它匹配百分号 (%)。
_（下划线字符）	可将下划线与 SIMILAR TO 配合使用来匹配单个字符。 不将下划线视为 REGEXP 和 REGEXP_SUBSTR 所使用的元字符。当指定时，它匹配下划线 (_)。
|	管道符号用于指定匹配字符串时要使用的替代模式。在由竖线分隔的一行模式中，竖线被解释为 OR，匹配过程从最左侧的模式开始，在找到第一个匹配项时停止。因此，您应按优先级的降序顺序列出模式。您可以指定任意数量的替代模式。
(和 **)**	当左括号和右括号用于正则表达式的各个组合部分时，它们为元字符。例如，`(ab)*` 匹配零个或多个 ab 的重复项。与使用数学表达式一样，您使用组合来控制正则表达式各部分的计算顺序。
{ 和 }	当左大括号和右大括号用于指定**量词**时，它们为元字符。量词指定一个模式要构成某个匹配所必须重复的次数。例如： ● ***{m}*** 匹配某个字符正好 *m* 次。例如，`'519-[0-9]{3}-[0-9]{4}'` 匹配 519 地区号中的一个电话号码（假定数据按语法中定义的方式进行格式设置）。 ● ***{m,}*** 匹配某个字符至少 *m* 次。例如，`'[0-9]{5,}'` 匹配任何含有五个或更多数字的字符串。 ● ***{m,n}*** 匹配某个字符至少 *m* 次，但不超过 *n* 次。例如，`SIMILAR TO '_{5,10}'` 匹配任何含有 5 到 10（含 5 和 10）个字符的字符串。
\	反斜线被用作元字符的转义字符。它也可被用于转义非元字符。

字符	其它信息
^	对于 REGEXP 和 REGEXP_SUBSTR，当脱字符在字符类的外部时，脱字符匹配字符串的开头字符。例如，'^[hc]at' 匹配 hat 和 cat，但只在字符串的开头。 当用在字符类内部时，以下行为适用： ● **REGEXP 和 REGEXP_SUBSTR** 当脱字符为字符类中的第一个字符时，它与字符集中字符以外的任何字符匹配。例如，REGEXP '[^abc]' 匹配 a、b 或 c 以外的任何字符。 如果脱字符不是方括号内的第一个字符，那么它匹配脱字符。例如，REGEXP_SUBSTR '[a-e^c]' 匹配 a、b、c、d、e 和 ^。 ● **SIMILAR TO** 对于 SIMILAR TO，脱字符被视作减号运算符。例如，SIMILAR TO '[a-e^c]' 匹配 a、b、d 和 e。
$	当与 REGEXP 和 REGEXP_SUBSTR 一起使用时，匹配字符串的结尾字符。例如，SIMILAR TO 'cat$' 匹配 cat，但不匹配 catfish。 当与 SIMILAR TO 一起使用时，它匹配问号。
.	当与 REGEXP 和 REGEXP_SUBSTR 一起使用时，匹配任何单个字符。例如，REGEXP 'a.cd' 匹配以 a 开头并以 cd 结尾的含有四个字符的任何字符串。 当与 SIMILAR TO 一起使用时，它匹配句点 (.)。
:	在字符集中使用冒号来指定子字符类。例如，'[[:alnum:]]'。

正则表达式：特殊子字符类

子字符类是嵌入到较大字符类中的特殊字符类。除了自定义字符类（在其中定义要匹配的字符集，例如，[abxq4] 将匹配字符集限制为 a、b、x、q 和 4）以外，SQL Anywhere 还支持子字符类，例如，大部分 POSIX 字符类。例如，[[:alpha:]] 表示所有大写和小写字母的集合。

REGEXP 搜索条件和 REGEXP_SUBSTR 函数支持下表中的所有语法约定，但 SIMILAR TO 搜索表达式不支持。SIMILAR TO 支持的约定在 SIMILAR TO 列中有一个 Y。

在 REGEXP 中，当使用 REGEXP_SUBSTR 函数时，可以使用脱字符对子字符类取非。例如，[[:^alpha:]] 匹配除字母字符以外的所有字符的集合。

子字符类	其它信息	SIMILAR TO
[:alpha:]	匹配当前归类中的大写和小写字母字符。例如，'[0-9]{3}[[:alpha:]]{2}' 匹配三个数字，后跟两个字母。	Y
[:alnum:]	匹配当前归类中的数字、大写和小写字母字符。例如，'[[:alnum:]]+' 匹配含有一个或多个字母和数字的字符串。	Y

子字符类	其它信息	SIMILAR TO
[:digit:]	匹配当前归类中的数字。例如，'[[:digit:]-]+' 匹配含有一个或多个数字或横线的字符串。同样，'[^[:digit:]-]+' 匹配含有一个或多个不是数字或横线的字符的字符串。	Y
[:lower:]	匹配当前归类中的小写字母字符。例如，'[[:lower:]]' 不匹配 A，因为 A 为大写。	Y

[:space:]	匹配单个空格 (' ')。例如，以下语句搜索 Contacts.City 以查找任何名称为两个词的城市：
	`SELECT City`
	`FROM Contacts`
	`WHERE City REGEXP '.*[[:space:]].*';`
		Y

[:upper:]	匹配当前归类中的大写字母字符。例如，'[[:upper:]ab]' 与以下其中一项匹配：任何大写字母、a 或 b。	Y
[:whitespace:]	匹配一个空白字符，例如，空格、制表符、换页符和回车符。	Y
[:ascii:]	匹配任何七位的 ASCII 字符（0 到 127 之间的顺序值）。	
[:blank:]	匹配一个空白区或水平制表符。 [[:blank:]] 等效于 [\t]。	
[:cntrl:]	匹配顺序值小于 32 或字符值为 127 的 ASCII 字符（控制字符）。控制字符包括换行符、换页符、退格符，等等。	
[:graph:]	匹配打印字符。 [[:graph:]] 等效于 [[:alnum:][:punct:]]。	
[:print:]	匹配打印字符和空格。 [[:print:]] 等效于 [[:graph:][:whitespace:]]。	
[:punct:]	匹配其中一个字符： !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~. [:punct:] 子字符类不能包括当前归类中可用的非 ASCII 标点字符。	
[:word:]	匹配当前归类中的字母、数字或下划线字符。 [[:word:]] 等效于 [[:alnum:]_]。	
[:xdigit:]	匹配字符类 [0-9A-Fa-f] 中的字符。	

正则表达式：所支持的其它语法约定

REGEXP 搜索条件和 REGEXP_SUBSTR 函数支持以下语法约定，同时它们假定反斜线为转义字符。*而 SIMILAR TO 搜索表达式不支持这些约定。*

正则表达式语法	名称和含义
\0*xxx*	匹配值为 \0*xxx* 的字符，其中 *xxx* 是任何八进制数字序列，0 是零。例如，`\0134` 匹配反斜线。
\a	匹配报警字符。
\A	用在字符集外部以便匹配字符串的开头。 等效于在字符集外部使用的 ^。
\b	匹配退格字符。
\B	匹配反斜线字符 (\)。
\c*X*	匹配已命名的控制字符。例如，\cZ 代表 ctrl-Z。
\d	匹配当前归类中的一个数字。例如，以下语句搜索 Contacts.Phone 以查找以 00 结尾的所有电话号码： `SELECT Surname, Surname, City, Phone FROM Contacts WHERE Phone REGEXP '\\d{8}00';` \d 既可用在字符类的内部也可用在字符类的外部，等效于 `[[:digit:]]`。
\D	匹配数字以外的任何字符。它的作用与 \d 正好相反。 \D 既可用在字符类的内部也可用在字符类的外部，等效于 `[^[:digit:]]`。 在方括号内使用取非速记时请务必谨慎。`[\D\S]` 与 `[^\d\s]` 并不相同。后者匹配数字或空格以外的任何字符。所以它匹配 x，但不匹配 8。而前者匹配不是数字或不是空格（满足两个条件之一）的任何字符。因为数字不是空格，空格也不是数字，所以 `[\D\S]` 可以匹配任何字符、数字、空格或其它字符。
\e	匹配转义字符。

正则表达式语法	名称和含义
\E	将由 \Q 启动的将元字符视为非元字符这一功能停止。 有关正则表达式元字符的列表，请参见“正则表达式：元字符”一节第 21 页。
\f	匹配换页符。
\n	匹配换行符。
\Q	将所有元字符视为非元字符，直到遇到 \E。例如，`\Q[$\E` 等效于 `\[\$`。 有关正则表达式元字符的列表，请参见“正则表达式：元字符”一节第 21 页。
\r	匹配回车符。

\s	匹配一个被视为白空格的空格或字符。例如，以下语句从 Products.ProductName 中返回名称中至少有一个空格的所有产品名：
	`SELECT Name`
	`FROM Products`
	`WHERE Name REGEXP '.*\\s.*'`
	\s 既可用在字符类的内部也可用在字符类的外部，等效于 `[[:whitespace:]]`。请参见“正则表达式：特殊子字符类”一节第 23 页。

\S	匹配非白空格字符。它的作用与 \d 正好相反，而等效于 `[^[:whitespace:]]`。 \S 既可用在字符类的内部也可用在字符类的外部。请参见“正则表达式：特殊子字符类”一节第 23 页。 在方括号内使用取非速记时请务必谨慎。`[\D\S]` 与 `[^\d\s]` 并不相同。后者匹配数字或空格以外的任何字符。所以它匹配 x，但不匹配 8。而前者匹配不是数字或不是空格（满足两个条件之一）的任何字符。因为数字不是空格，空格也不是数字，所以 `[\D\S]` 可以匹配任何字符、数字、空格或其它字符。
\t	匹配水平制表符。

正则表达式语法	名称和含义
\v	匹配垂直制表符。
\w	匹配当前归类中的字母字符、数字或下划线。例如，以下语句从 Contacts.Surname 返回长度正好为七个字母数字字符的所有姓： `SELECT Surname FROM Contacts WHERE Surname REGEXP '\\w{7}';` \w 既可用在字符类的内部也可用在字符类的外部。请参见“正则表达式：特殊子字符类”一节第 23 页。 等效于 `[[:alnum:]_].`。
\W	匹配当前归类中字母字符、数字或下划线以外的任何字符。它的作用与 \w 正好相反，而等效于 `[^[:alnum:]_]`。 在字符类的内部和外部都可使用此正则表达式。请参见“正则表达式：特殊子字符类”一节第 23 页。
\x*hh*	匹配值为 0x*hh* 的字符，其中 *hh* 最多为两个十六进制数字。例如，`\x2D` 等效于一个连字符。 等效于 \x{*hh*}。
\x{*hhh***}**	匹配值为 0x*hhh* 的字符，其中 *hhh* 最多为三个十六进制数字。
\z 和 **\Z**	匹配字符串结尾处的位置（而非字符）。 等效于 `$`。

正则表达式：断言

断言测试条件是否为真，并影响字符串中开始匹配的位置。断言不返回字符；最终匹配中不包括断言模式。REGEXP 搜索条件和 REGEXP_SUBSTR 函数支持这些断言模式。而 SIMILAR TO 搜索表达式不支持这些约定。

在尝试拆分字符串时，lookahead 和 lookbehind 断言对于 REGEXP_SUBSTR 将非常有用。例如，您可以通过执行以下语句返回 Customers 表的 Address 列中街道名称（不带街道编号）的列表：

```
SELECT REGEXP_SUBSTR( Street, '(?<=^\\S+\\s+).*$' )
FROM Customers;
```

另一个示例：假定您想要使用正则表达式来验证口令是否符合某些规则。您可以使用类似于下面内容的零宽度断言：

```
IF password REGEXP '(?=.*[[:digit:]])(?=.*[[:alpha:]].*[[:alpha:]])[[:word:]]
{4,12}'
   MESSAGE 'Password conforms' TO CLIENT;
ELSE
   MESSAGE 'Password does not conform' TO CLIENT;
END IF
```

当满足以下条件时，口令有效：

- *password* 至少有一位数（零宽度肯定断言 [[:digit:]]）
- *password* 至少有两个字母字符（零宽度肯定断言 [[:alpha:]].*[[:alpha:]]）
- *password* 只含有字母数字字符或下划线字符 ([[:word:]])
- *password* 最少含有 4 个字符，最多含有 12 个字符 ({4,12})

下表包含 SQL Anywhere 支持的断言：

语法	含义
(?=*pattern*)	**肯定的 lookahead 零宽度断言** 查看字符串中的当前位置是否紧跟着出现了 *pattern*，而 *pattern* 不会成为匹配字符串的一部分。`'A(?=B)'` 匹配后面跟有 B 的 A，但不使 B 成为匹配的一部分。 例如，`SELECT REGEXP_SUBSTR( 'in new york city', 'new(?=\\syork)');` 会返回子串 new，因为它后面紧跟着 'york'（请注意 york 前面的空格）。
(?!*pattern*)	**否定的 lookahead 零宽度断言** 查看字符串中的当前位置是否*没有*紧跟着出现 *pattern*，而 *pattern* 不会成为匹配字符串的一部分。所以，`'A(?!B)'` 匹配后面未跟着 B 的 A。 例如，`SELECT REGEXP_SUBSTR('new jersey', 'new(?!\\syork)');` 会返回子串 new。
(?<=*pattern*)	**肯定的 lookbehind 零宽度断言** 查看字符串中的当前位置是否前面紧挨着出现了 *pattern*，而 *pattern* 不会成为匹配字符串的一部分。所以，`'(?<=A)B'` 匹配前面紧挨着 A 的 B，但不使 A 成为匹配的一部分。 例如，`SELECT REGEXP_SUBSTR('new york', '(?<=new\\s)york');` 会返回子串 york。
(?<!*pattern*)	**否定的 lookbehind 零宽度断言** 查看字符串中的当前位置的前面是否*没有*紧挨着出现 *pattern*，而 *pattern* 不会成为匹配字符串的一部分。 例如，`SELECT REGEXP_SUBSTR('about york', '(?<!new\\s)york');` 会返回子串 york。
(?>*pattern*)	**所属关系局部子表达式** 仅匹配与 *pattern* 匹配的剩余字符串的最大前缀。 例如，在 `'aa' REGEXP '(?>a*)a'` 中，`(?>a*)` 匹配（并消耗）aa，而决不仅仅是前导 a。因此，`'aa' REGEXP '(?>a*)a'` 的计算结果为 false。

语法	含义
(?:*pattern*)	**非捕获块**　该语法在功能上就等效于 *pattern*，是为实现兼容性而提供。 例如，在 `'bb' REGEXP '(?:b*)b'` 中，`(?:b*)` 匹配（并消耗）bb。但是，与所属关系局部子表达式不同，bb 中的最后一个 b 会被放弃，以允许整个匹配成功（即，允许与在非捕获块的外部找到的 b 匹配）。 同样，`'a(?:bc|b)c'` 匹配 abcc 和 abc。在匹配 abc 时，bc 中最后面的 c 会发生回溯，以便可以使用组外的 c 来使匹配成功。
(?#*text*)	用于注释。*text* 的内容会被忽略。

另请参见

- “正则表达式示例”一节第 29 页

正则表达式示例

下表显示正则表达式的使用示例。所有示例都适用于 REGEXP，部分示例也适用于 SIMILAR TO（如[示例]列中注释）。结果视您用于搜索的搜索条件而异。对于使用 SIMILAR TO 的示例，结果还要另外根据是否区分大小写和重音而异。

有关 REGEXP 和 SIMILAR TO 如何处理匹配和计算范围的比较，请参见“LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页。

请注意，如果在文字字符串中使用这些示例（例如，`'.+@.+\\..+'`），则应使用双反斜线

示例	匹配示例

信用卡号（仅限 REGEXP）：
Visa：
`4[0-9]{3}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}`
MasterCard：
`5[0-9]{3}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}`
American Express：
`37[0-9]{2}\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}`
Discover：
`6011\s[0-9]{4}\s[0-9]{4}\s[0-9]{4}`
	匹配 (Visa)： 4123 6453 2222 1746
	非匹配 (Visa)：
	3124 5675 4400 4567, 4123-6453-2222-1746
	同样，MasterCard 匹配一组 16 位的号码，以 5 开头，每四位号码组成的子集之间各有一个空格。American Express 和 Discover 是相同的，但是必须分别以 37 和 6011 开头。

示例	匹配示例
日期（REGEXP 和 SIMILAR TO 均适用）： `([0-2][0-9]|30|31)/(0[1-9]|1[0-2])/[0-9]{4}`	匹配： 31/04/1999, 15/12/4567 非匹配： 31/4/1999, 31/4/99, 1999/04/19, 42/67/25456
Windows 绝对路径（仅限 REGEXP）： `([A-Za-z]:|\\)\\[[:alnum:][:whitespace:]!"#$%&'()+,-.\\;=@\[\]^_`{}~.]*`	匹配：\\server\share\file 非匹配：\directory\directory2, /directory2
电子邮件地址（仅限 REGEXP）： `[[:word:]\-.]+@[[:word:]\-.]+\.[[:alpha:]]{2,3}`	匹配：abc.123@def456.com, _123@abc.ca 非匹配：abc@dummy, ab*cd@efg.hijkl
电子邮件地址（仅限 REGEXP）： `.+@.+\..+`	匹配：*@qrstuv@wxyz.12345.com, __1234^%@@abc.def.ghijkl 非匹配：abc.123.*&ca, ^%abcdefg123
HTML 十六进制颜色代码（REGEXP 和 SIMILAR TO 均适用）： `[A-F0-9]{6}`	匹配：AB1234, CCCCCC, 12AF3B 非匹配：123G45, 12-44-CC
HTML 十六进制颜色代码（仅限 REGEXP）： `[A-F0-9]{2}\s[A-F0-9]{2}\s[A-F0-9]{2}`	匹配：AB 11 00, CC 12 D3 非匹配：SS AB CD, AA BB CC DD, 1223AB
IP 地址（仅限 REGEXP）： `((2(5[0-5]|[0-4][0-9])|1([0-9][0-9])|([1-9][0-9])|[0-9])\.){3}(2(5[0-5]|[0-4][0-9])|1([0-9][0-9])|([1-9][0-9])|[0-9])`	匹配： 10.25.101.216 非匹配： 0.0.0, 256.89.457.02
Java 注释（仅限 REGEXP）： `/\*.*\*/|//[^\n]*`	匹配位于 /* 和 */ 之间的 Java 注释，或者前缀为 // 的一行注释。 非匹配：a=1
货币（仅限 REGEXP）： `(\+|-)?\$[0-9]*\.[0-9]{2}`	匹配： $1.00, -$97.65 非匹配： $1, 1.00$, $-75.17
正数、负数和小数值（仅限 REGEXP）： `(\+|-)?[0-9]+(\.[0-9]+)?`	匹配： +41, -412, 2, 7968412, 41, +41.1, -3.141592653 非匹配： ++41, 41.1.19, -+97.14

示例	匹配示例
口令（REGEXP 和 SIMILAR TO 均适用）： `[[:alnum:]]{4,10}`	匹配：abcd, 1234, A1b2C3d4, 1a2B3 非匹配：abc, *ab12, abcdefghijkl
口令（仅限 REGEXP）： `[a-zA-Z]\w{3,7}`	匹配：AB_cd, A1_b2c3, a123_ 非匹配：*&^g, abc, 1bcd
电话号码（REGEXP 和 SIMILAR TO 均适用）： `([2-9][0-9]{2}-[2-9][0-9]{2}-[0-9]{4})|([2-9][0-9]{2}\s[2-9][0-9]{2}\s[0-9]{4})`	匹配： 519-883-6898, 519 888 6898 非匹配： 888 6898, 5198886898, 519 883-6898
句子（仅限 REGEXP）： `[A-Z0-9].*(\.|\?|!)`	匹配：Hello, how are you? 非匹配：i am fine
句子（仅限 REGEXP）： `[[:upper:]0-9].*[.?!]`	匹配：Hello, how are you? 非匹配：i am fine
社保号码（REGEXP 和 SIMILAR TO 均适用）： `[0-9]{3}-[0-9]{2}-[0-9]{4}`	匹配： 123-45-6789 非匹配：123 45 6789, 123456789, 1234-56-7891
URL（仅限 REGEXP）： `(http://)?www\.[a-zA-Z0-9]+\.[a-zA-Z]{2,3}`	匹配：http://www.sample.com、www.sample.com 非匹配：http://sample.com, http://www.sample.comm

另请参见

- “正则表达式语法”一节第 20 页

表达式的兼容性

分隔字符串的缺省解释

SQL Anywhere 采用 SQL/2003 约定，用撇号围起来的字符串是常量表达式，用引号（双引号）引起来的字符串是界定标识符（数据库对象的名称）。

quoted_identifier 选项

SQL Anywhere 提供了一个允许改变分隔字符串解释的 quoted_identifier 选项。缺省情况下，quoted_identifier 选项在 SQL Anywhere 中设置为 On。请参见“quoted_identifier 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

如果 quoted_identifier 选项被设置为 Off，则 SQL 保留字不能用作标识符。

有关保留字的完整列表，请参见“保留字”一节第 4 页。

设置该选项

以下语句将 quoted_identifier 选项的设置更改为 On：

```
SET quoted_identifier On;
```

以下语句将 quoted_identifier 选项的设置更改为 Off：

```
SET quoted_identifier Off;
```

分隔字符串的兼容解释

可以选择在 SQL Anywhere 中使用 SQL/2003 或缺省 Transact-SQL 约定，只要每个 DBMS 中的 quoted_identifier 选项设为相同的值。

示例

如果选择在 quoted_identifier 选项为 On（缺省设置）时进行操作，则以下涉及 SQL 关键字 **user** 的语句对两个 DBMS 均有效。

```
CREATE TABLE "user" ( col1 char(5) ) ;
INSERT "user" ( col1 )
   VALUES ( 'abcde' );
```

如果选择在 quoted_identifier 选项为 Off 时操作，则以下语句对两个 DBMS 均有效。

```
SELECT *
FROM Employees
WHERE Surname = "Chin":
```


搜索条件

搜索条件是为 WHERE 子句、HAVING 子句、CHECK 子句、连接中 ON 短语或 IF 表达式指定的标准。搜索条件也称作**谓语**。

语法

search-condition :
expression comparison-operator expression
| *expression comparison-operator* { [**ANY** | **SOME**] | **ALL** } (*subquery*)
| *expression* **IS** [**NOT**] **NULL**
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **LIKE** *pattern* [**ESCAPE** *expression*]
| *expression* [**NOT**] **SIMILAR TO** *pattern* [**ESCAPE** *escape-expression*]
| *expression* [**NOT**] **REGEXP** *pattern* [**ESCAPE** *escape-expression*]
| *expression* [**NOT**] **IN (** { *expression*
| *subquery*
| *value-expression1* **,** ... } **)**
| **CONTAINS (***column-name* [,...] **,** *query-string* **)**
| **EXISTS (** *subquery* **)**
| **NOT** *condition*
| *search-condition* [{ **AND** | **OR** } *search-condition*] [...]
| **(** *search-condition* **)**
| **(** *search-condition* **,** *estimate* **)**
| *search-condition* **IS** [**NOT**] { **TRUE** | **FALSE** | **UNKNOWN** }
| *trigger-operation*

comparison-operator :
=
| **>**
| **<**
| **>=**
| **<=**
| **<>**
| **!=**
| **!<**
| **!>**

trigger-operation :
INSERTING
| **DELETING**
| **UPDATING** [**(** *column-name-string* **)**]
| **UPDATE(** *column-name* **)**

参数

- **ALL 搜索条件** 请参见“ALL 搜索条件”一节第 35 页。
- **ANY 和 SOME 搜索条件** 请参见“ANY 和 SOME 搜索条件”一节第 35 页。
- **BETWEEN 搜索条件** 请参见“BETWEEN 搜索条件”一节第 36 页。
- **CONTAINS 搜索条件** 请参见“CONTAINS 搜索条件”一节第 46 页。
- **EXISTS 搜索条件** 请参见“EXISTS 搜索条件”一节第 51 页。
- **LIKE 搜索条件** 请参见“LIKE 搜索条件”一节第 38 页。

- **SIMILAR TO 搜索条件** 请参见“SIMILAR TO 搜索条件”一节第 44 页。
- **REGEXP 搜索条件** 请参见“REGEXP 搜索条件”一节第 43 页。

注释

搜索条件用于从表中选择行的子集或者在控制语句（如 IF 语句）中确定控制流。

在 SQL 中，每个条件的值均为 TRUE、FALSE 或 UNKNOWN 中的一个。这叫做三值逻辑。如果所比较的值中某一个为 NULL，则比较结果为 UNKNOWN。有关显示三值逻辑中逻辑运算符组合方式的表，请参见“三值逻辑”一节第 53 页。

当且仅当条件的结果为 TRUE 时，行才满足搜索条件。条件为 UNKNOWN 或 FALSE 时，行不满足搜索条件。有关 NULL 的详细信息，请参见“NULL 值”一节第 71 页。

子查询构成了一类很重要的表达式，该表达式可用在很多搜索条件中。有关在搜索条件中使用子查询的信息，请参见“搜索条件中的子查询”一节第 34 页。

以下几节论述了不同类型的搜索条件。

LIKE、SIMILAR TO 和 REGEXP 搜索条件非常相似。要了解它们之间的相似之处和差异，请参见“LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页。

权限

必须连接到数据库。

副作用

无。

另请参见

- “表达式”一节第 16 页

搜索条件中的子查询

在 SQL 语句中可以使用列名称的任何地方（包括在表达式中），都可以使用只返回一列并且零行或一行的子查询。

例如，只要子查询返回的行数不超过一行，就可以在比较条件中将表达式与子查询进行比较。如果子查询（必须只含一列）返回一行，则该行的值将与表达式进行比较。如果子查询未返回行，则其值为 NULL。

只返回一列及任意个行的子查询可用在 IN、ANY、ALL 和 SOME 搜索条件中。返回任意个列和行的子查询可用在 EXISTS 搜索条件中。以下几节将就这些搜索条件进行论述。

另请参见

- “比较运算符”一节第 12 页

ALL 搜索条件

语法

expression comparison-operator **ALL (** *subquery* **)**

comparison-operator:
=
| >
| <
| >=
| <=
| <>
| !=
| !<
| !>

注释

对于 ALL 搜索条件，如果子查询结果集的值为空集，则搜索条件的值为 TRUE。否则，搜索条件的值为 TRUE、FALSE 或 UNKNOWN，这要视 *expression* 的值以及子查询所返回的结果集而定，具体对应关系如下：

如果表达式值为..	并且子查询所返回的结果集至少包含一个 NULL，那么..	或者子查询所返回的结果集不含 NULL，那么..
NULL	UNKNOWN	UNKNOWN
非 NULL	如果子查询结果集中至少存在一个值与表达式进行比较后所得的值为 FALSE，则搜索条件的值为 FALSE。否则，搜索条件的值为 UNKNOWN。	如果子查询结果集中至少存在一个值与表达式进行比较后所得的值为 FALSE，则搜索条件的值为 FALSE。否则，搜索条件的值为 TRUE。

ANY 和 SOME 搜索条件

语法

expression comparison-operator { **ANY** | **SOME** }**(** *subquery* **)**

comparison-operator:
=
| >
| <
| >=
| <=
| <>
| !=
| !<
| !>

注释

关键字 ANY 与 SOME 是同义字。

对于 ANY 和 SOME 搜索条件，如果子查询结果集为空集，则搜索条件的结果值为 FALSE。否则，搜索条件的值为 TRUE、FALSE 或 UNKNOWN，这要视 *expression* 的值以及子查询所返回的结果集而定，具体对应关系如下：

如果表达式值为..	并且子查询所返回的结果集至少包含一个 NULL，那么..	或者子查询所返回的结果集不含 NULL，那么..
NULL	UNKNOWN	UNKNOWN
非 NULL	如果子查询结果集中至少存在一个值与表达式进行比较后所得的值为 TRUE，则搜索条件的值为 TRUE。否则，搜索条件的值为 UNKNOWN。	如果子查询结果集中至少存在一个值与表达式进行比较后所得的值为 TRUE，则搜索条件的值为 TRUE。否则，搜索条件的值为 FALSE。

对于带相等运算符的 ANY 或 SOME 搜索条件，如果 *expression* 等于子查询结果中的任何一个值，则该搜索条件的结果值为 TRUE；如果表达式的值不为 NULL 且不等于子查询结果中的任何值，而且结果集中也不包含 NULL，则该搜索条件的结果值为 FALSE。

注意
使用 = **ANY** 或 = **SOME** 就相当于使用 IN 关键字。

BETWEEN 搜索条件

语法

expression [**NOT**] **BETWEEN** *start-expression* **AND** *end-expression*

注释

BETWEEN 搜索条件的值可以是 TRUE、FALSE 或 UNKNOWN。无 NOT 关键字时，如果 *expression* 介于 *start-expression* 和 *end-expression* 之间，则搜索条件的值为 TRUE。NOT 关键字会使搜索条件的含义相反，但对于 UNKNOWN，其含义保持不变。

BETWEEN 搜索条件相当于两个不等式的组合：

[**NOT**] (*expression* >= *start-expression* **AND** *expression* <= *end-expression*)

LIKE、REGEXP 和 SIMILAR TO 搜索条件

REGEXP、LIKE 和 SIMILAR TO 搜索条件都是试图将某个模式与字符串进行匹配，在这一方面它们是类似的。而且，这三者都试图匹配整个字符串，而不是字符串内的子串。

所有这三个搜索条件的基本语法类似：

expression search-condition pattern

LIKE、REGEXP 和 SIMILAR TO：模式定义上的差异

REGEXP、LIKE 和 SIMILAR TO 搜索条件的不同之处在于如何定义 *pattern*：

- REGEXP 支持基于 SIMILAR TO 所支持正则表达式语法的超集。此外，为与其它产品兼容，REGEXP 搜索条件还支持几个语法扩展。另外，REGEXP 和 SIMILAR TO 还具有不同的缺省转义字符。REGEXP 的行为与 Perl 5 比较接近（除了不支持 Perl 语法和运算符的方面）。
- *pattern* 的 LIKE 语法很简单，它支持一小组通配符，但不支持完整的正则表达式语法。
- *pattern* 的 SIMILAR TO 语法可使用 ANSI/ISO SQL 标准中定义的正则表达式语法进行可靠的模式匹配。

LIKE、REGEXP 和 SIMILAR TO：字符比较上的差异

在执行比较时，REGEXP 的行为不同于 LIKE 和 SIMILAR TO。对于 REGEXP 的比较，数据库服务器使用**数据库字符集**中的代码点值进行比较。这与 Perl 等其它正则表达式的实现方法一致。

对于 LIKE 和 SIMILAR TO，数据库服务器使用**数据库归类**中的等同性和排序顺序进行比较。这与数据库计算 > 和 = 等比较运算符的方法一致。

字符比较方法不同意味着 REGEXP 和 LIKE/SIMILAR 的匹配及范围计算的结果也不同。

- **匹配上的差异** 因为 REGEXP 使用代码点值，所以只有它是完全相同的字符时才匹配模式中的文字。因此，REGEXP 匹配不受数据库归类区分大小写或区分重音等因素的影响。例如，"A" 决不会作为 "a" 的匹配项返回。

  因为 LIKE 和 SIMILAR TO 使用数据库归类，在确定字符的等同性时结果会受区分大小写和区分重音的影响。例如，如果数据库归类不区分大小写和重音，则匹配项也不区分大小写和重音。因此，"A" 会作为 "a" 匹配项返回。

- **范围计算上的差异** 因为 REGEXP 使用代码点值进行范围计算，所以，如果字符的代码点值等于范围起点和终点的代码点值，或介于这两个值之间，则认为字符在此范围之内。例如，对单个字符 *x* 的比较关系 `x REGEXP '[A-C]'` 等同于 `CAST(x AS BINARY) >= CAST(A AS BINARY) AND CAST(x AS BINARY) <= CAST(C AS BINARY)` 。

  因为 LIKE 和 SIMILAR TO 使用归类排序顺序进行范围计算，所以，如果字符在归类中的位置与范围的起点字符和终点字符的位置相同，或在这两个位置之间，则认为字符在此范围内。例如，比较关系 `x SIMILAR TO '[A-C]'`（这里 *x* 为单字符）等同于 `x >= A AND x <= C`，并使用归类排序顺序计算比较运算符。

  下表显示了在 LIKE、SIMILAR TO 和 REGEXP 所计算的 `'[A-C]'` 范围中包括的字符集。两个数据库都使用 1252LATIN1 归类，但第一个数据库不区分大小写，而第二个数据库区分大小写。

	LIKE/SIMILAR TO '[A-C]'	REGEXP '[A-C]'
demo.db（不区分大小写）	A、B、C、a、b、c、ª、À、Á、Â、Ã、Ä、Å、Æ、Ç、à、á、â、ã、ä、å、æ、ç	A、B、C

	LIKE/SIMILAR TO '[A-C]'	REGEXP '[A-C]'
charsensitive.db（区分大小写）	A、B、C、b、c、À、Á、Â、Ã、Ä、Å、Æ、Ç、ç	A、B、C

在结果中可以观察到以下几点：

- LIKE 和 SIMILAR TO 在范围中包含重音字符。
- 根据数据库是否区分大小写，LIKE 和 SIMILAR TO 包含不同的字符。具体地说，它们包含在范围内找到的任何小写字母，您在区分大小写的数据库中搜索时可能还没有预见到这些字母。

  同样，在不区分大小写的数据库中，您假定会包含在范围中的一些字符却没有被包含进去。例如，对不区分大小写数据库执行的 `SIMILAR TO '[a-c]'` 包含 a、A、b、B、c，但没有包含 C，因为按照排序顺序 C 出现在小写 c 之后。
- REGEXP 只返回 A、B、C，不管数据库是否区分大小写。如果想要范围内包含小写字符，必须将它们添加到范围定义中。例如，`REGEXP '[a-cA-C]'`。
- REGEXP 的字符集不会更改，不管数据库是否区分大小写。

尽管与上述示例相比，您的数据库使用不同的归类，或者具有不同的大小写区分或重音区分设置，但您可以通过连接到数据库并执行 LIKE、SIMILAR TO 或 REGEXP 这些语句中的任一句来执行类似的测试，以查看这些语句会返回什么结果：

```
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) LIKE '[A-
C]';
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) REGEXP '[A-
C]';
SELECT CHAR( row_num ) FROM RowGenerator WHERE CHAR( row_num ) SIMILAR TO
'[A-C]';
```

另请参见

- “正则表达式概述”一节第 20 页
- “正则表达式语法”一节第 20 页
- “正则表达式示例”一节第 29 页

LIKE 搜索条件

语法

LIKE 搜索条件的语法如下：

expression [**NOT**] **LIKE** *pattern* [**ESCAPE** *escape-character*]

参数

- **expression**　被搜索的字符串。
- **pattern**　在 *expression* 内搜索的模式。

- **escape-character** 用于转义下划线和百分号等特殊字符的字符。缺省的转义字符为零字符，可被指定为 "\x00" 形式的字符串文字。

注释

LIKE 搜索条件尝试将 *expression* 与 *pattern* 匹配，从而得出 TRUE、FALSE 或 UNKNOWN。

如果 *expression* 与 *pattern* 匹配（假设未指定 NOT），搜索条件的结果值为 TRUE。如果 *expression* 或 *pattern* 的值为 NULL，则搜索条件的结果值为 UNKNOWN。NOT 关键字会使搜索条件的含义相反，但对于 UNKNOWN，其含义保持不变。

expression 被解释为 CHAR 或 NCHAR 字符串。匹配时会使用 *expression* 的整个内容。同样，*pattern* 也被解释为 CHAR 或 NCHAR 字符串，可以包含下表中任意数目的支持的通配符：

通配符	匹配项
_（下划线）	任意一个字符。例如，`a_` 与 ab 和 ac 匹配，但与 a 不匹配。
%（百分号）	包含零个或多个字符的任意字符串。例如，`bl%` 与 bl 和 bla 匹配。
[]	指定范围或集合中的任何单个字符。例如，`T[oi]m` 与 Tom 或 Tim 匹配。
[^]	不在指定范围或集合中的任何单个字符。例如，`M[^c]` 与 Mb 和 Md 匹配，但与 Mc 不匹配。

所有其它字符都必须完全匹配。

例如，对于任何名称以字母 a 开头且倒数第二个字符为字母 b 的行，以下搜索条件都会返回 TRUE。

```
... name LIKE 'a%b_'
```

如果指定了 *escape-character*，其结果值必须为单字节 CHAR 或 NCHAR 字符。转义字符可以位于百分号、下划线、左方括号或 *pattern* 中另一转义字符之前，以防止特殊字符具有特殊的含义。以这种方式转义之后，百分号匹配百分号，下划线匹配下划线。

支持所有小于或等于 126 个字节的模式。不支持不含通配符的且大于 126 个字节的模式。支持超过 126 个字节且含通配符字符的模式，具体情况取决于模式的内容。用于表示模式的字节数取决于模式的数据类型是 CHAR 还是 NCHAR。

LIKE 搜索条件的不同使用方式

搜索对象	示例	其它信息
某字符集合中的一个字符	`LIKE 'sm[iy]th'`	可以通过将字符列在方括号中来指定要查找的字符集。在此示例中，搜索条件与 *smith* 和 *smyth* 匹配。
某字符范围中的一个字符	`LIKE '[a-r]ough'`	可以通过将字符范围的两端列在方括号中（用连字符分隔）来指定要查找的字符范围。在此示例中，搜索条件与 bough 和 rough 匹配，但与 tough 不匹配。 字符范围 [a-z] 被解释为 "大于等于 a 且小于等于 z"，其中大于和小于运算在数据库归类中执行。有关匹配范围的信息，请参见“LIKE、REGEXP 和 SIMILAR TO：字符比较上的差异”一节第 37 页。 范围的下限必须位于上限之前。例如，`[z-a]` 与任何字符都不匹配，因为没有字符可与范围 [z-a] 匹配。
范围与集合的组合	`... LIKE '[a-rt]ough'`	可以在方括号中将范围和集合组合起来。在此示例中，`... LIKE '[a-rt]ough'` 与 bough、rough 和 tough 匹配。 模式 [a-rt] 被解释为一个介于 a 和 r 之间（包括 a 和 r）或等于 t 的字符。

搜索对象	示例	其它信息
不在某范围内的一个字符	`... LIKE '[^a-r]ough'`	脱字符 (^) 用于指定排除在搜索之外的字符范围。在此示例中，`LIKE '[^a-r]ough'` 与字符串 tough 匹配，但与字符串 rough 或 bough 不匹配。 脱字符用于对方括号内的其余内容取反。例如，方括号 [^a-rt] 被解释为不在 a 和 r 之间（包括 a 和 r）且不为 t 的字符。
搜索带尾随空白的模式	`'90 '`、`'90[ ]'` 和 `'90_'`	当搜索模式含尾随空白时，数据库服务器只将此模式与包含空白的值匹配，而并不会为字符串填充空白。例如。模式 '90 '、'90[ ]' 和 '90_' 与表达式 '90 ' 匹配，而与表达式 '90' 不匹配，即使所测试的值位于宽为三个或三个字符以上的 CHAR 或 VARCHAR 列中也是如此。

范围和组的特殊情况

方括号中的任何单个字符都只代表该字符本身。例如，`[a]` 仅与字符 a 匹配。`[^]` 仅与脱字符匹配，`[%]` 仅与百分号字符匹配（在此上下文中百分号字符不作为通配符使用），`[_]` 仅与下划线字符匹配。同样，`[[]` 仅与字符 [匹配。

其它特殊情况如下：

- 模式 `[a-]` 与字符 a 或 - 匹配。
- 模式 `[]` 永远没有匹配项，始终不会返回任何行。
- 模式 `[` 或 `[abp-q` 会返回语法错误，因为它们缺少闭括号。
- 方括号内不能使用通配符。模式 `[a%b]` 会找到 a、% 或 b 之一。
- 除非作为方括号内的第一个字符，否则脱字符不能用于对一个范围取非。模式 `[a^b]` 会找到 a、^ 或 b 之一。

区分大小写以及如何执行比较

如果数据库归类区分大小写，则搜索条件也区分大小写。要通过一个区分大小写的归类执行不区分大小写的搜索，则必须包括大写和小写字符。例如，对于字符串 Bough、rough 和 TOUGH，以下搜索条件的结果值为 true：

```
LIKE '[a-zA-Z][oO][uU][gG][hH]'
```

比较是逐个字符执行的，这与等号 (=) 运算符或其它运算符的逐字符串比较不同。例如，在 UCA 归类（归类设置为 UCA 的 CHAR 或 NCHAR）中进行比较时，`'?'='AE'` 为 true，但 `'?' LIKE 'AE'` 为 false。

对于逐个字符进行的匹配比较，被搜索的表达式中的每个字符都必须与 LIKE 表达式中的一个字符（使用归类的字符等同性）或一个通配符匹配。

有关对 LIKE、SIMILAR TO 和 REGEXP 之间如何处理匹配和范围计算进行的比较，请参见“LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页。

国家字符 (NCHAR) 支持

LIKE 搜索条件可以用于比较 CHAR 和 NCHAR 字符串。这种情况下会执行字符集转换，以便使用共同的数据类型进行比较。然后逐个字符地执行比较。请参见“CHAR 和 NCHAR 之间的比较”一节第 107 页。

通过在引号引起的值之前加前缀 N（例如，`expression LIKE N'pattern'`），可以将 *expression* 或 *pattern* 指定为 NCHAR 字符串文字。还可以使用 CAST 函数将模式转换为 CHAR 或 NCHAR（例如，`expression LIKE CAST(pattern AS datatype)`）。

请参见“字符串文字”一节第 11 页和“CAST 函数 [Data type conversion]”一节第 141 页。

空格填充的数据库

如果数据库以空格填充，LIKE 模式的语义不会更改，因为 *expression* 与 *pattern* 的匹配是按左到右的方式逐个字符进行比较。求值过程中不会再对 *expression* 或 *pattern* 执行其它空格填充。因此，表达式 `a1` 与模式 a1 匹配，而与模式 'a1'（a1 后加一个空格）或 a1_ 不匹配。

另请参见

- “LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页
- “WHERE 子句：指定行”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “优化 LIKE 谓语”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “REGEXP 搜索条件”一节第 43 页
- “SIMILAR TO 搜索条件”一节第 44 页

标准和兼容性

- LIKE 搜索条件是 ANSI SQL/2003 标准的核心特性。
- SQL Anywhere 支持 ANSI SQL/2003 特性 F281，即模式和转义字符串可以是在执行时求值的任意表达式。特性 F281 还允许 *expression* 是比简单列引用更为复杂的表达式。
- 在方括号 [] 内使用字符范围和集合是一项服务商扩展功能。

- SQL Anywhere 支持 ANSI SQL/2003 特性 T042，即允许 LIKE 搜索条件引用值类型为 LONG VARCHAR 的字符串表达式。
- 指定 NCHAR 字符串表达式或模式的 LIKE 搜索条件是 ANSI SQL/2003 标准的特性 F421。

REGEXP 搜索条件

将模式与字符串进行匹配。

语法

expression [**NOT**] **REGEXP** *pattern* [**ESCAPE** *escape-expression*]

参数

expression 被搜索的字符串。

pattern 在 *expression* 内搜索的正则表达式。

有关正则表达式的语法的详细信息，请参见“正则表达式概述”一节第 20 页。

escape-expression 在匹配中要使用的转义字符。缺省为反斜线字符 (\)。

注释

REGEXP 搜索条件匹配整个字符串，而不是子串。要让某子串与该字符串匹配，将字符串用匹配字符串其余部分的通配符包围起来 (`.*pattern.*`)。例如，`SELECT ... WHERE Description REGEXP 'car'` 仅与 car 匹配，与 sportscar 不匹配。但是，`SELECT ... WHERE Description REGEXP '.*car'` 则与 car、sportscar 和所有以 car 结尾的字符串都匹配。或者，也可以重写查询以使用 REGEXP_SUBSTR 函数，该函数用于搜索字符串中的子串。

当只依据子字符类进行匹配时，必须包含外部方括号以及用于子字符类的方括号。例如，`expression REGEXP '[[:digit:]]'`)。有关子字符类匹配的详细信息，请参见“正则表达式：特殊子字符类”一节第 23 页。

数据库归类和匹配

如果模式中的某文字是完全相同的字符（即，它们有相同的代码点值），则 REGEXP 只匹配该文字。字符类中的范围（例如，`'[A-F]'`）只匹配代码点值大于或等于范围 (A) 中第一个字符的代码点值且小于或等于范围 (F) 中第二个字符的代码点值的字符。

有关对 LIKE、SIMILAR TO 和 REGEXP 之间如何处理匹配和范围计算进行的比较，请参见“LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页。

比较是逐个字符执行的，这与等号 (=) 运算符或其它运算符的逐字符串比较不同。例如，在 UCA 归类（归类设置为 UCA 的 CHAR 或 NCHAR）中进行比较时，`'?'='AE'` 为 true，但 `'?' REGEXP 'AE'` 为 false。

国家字符 (NCHAR) 支持

REGEXP 搜索条件可以用于比较 CHAR 和 NCHAR 字符串。这种情况下会执行字符集转换，以便使用共同的数据类型进行比较。然后逐个代码点地执行比较。请参见“CHAR 和 NCHAR 之间的比较”一节第 107 页。

通过在引号引起的值之前加前缀 N（例如，`expression REGEXP N'pattern'`），可以将 *expression* 或 *pattern* 指定为 NCHAR 字符串文字。还可以使用 CAST 函数将模式转换为 CHAR 或 NCHAR（例如，`expression REGEXP CAST(pattern AS datatype)`）。

请参见“字符串文字”一节第 11 页和“CAST 函数 [Data type conversion]”一节第 141 页。

另请参见

- “正则表达式概述”一节第 20 页
- “SIMILAR TO 搜索条件”一节第 44 页
- “LIKE 搜索条件”一节第 38 页
- “REGEXP_SUBSTR 函数 [String]”一节第 263 页
- “LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页

SIMILAR TO 搜索条件

将模式与字符串进行匹配。

语法

expression [**NOT**] **SIMILAR TO** *pattern* [**ESCAPE** *escape-expression*]

参数

expression 要在其中进行搜索的表达式。

pattern 在 *expression* 内搜索的正则表达式。

有关支持的正则表达式语法的详细信息，请参见“正则表达式概述”一节第 20 页。

escape-expression 在匹配中要使用的转义字符。缺省的转义字符为零字符，可被指定为 "\x00" 形式的字符串文字。

正则表达式语法	含义
\x	匹配任何等于 *x* 的字符，其中转义字符采用反斜线字符 (\)。例如，`\[` 与 '[' 匹配。
x	任何字符（除了元字符）都与其自身匹配。例如，`A` 与 'A' 匹配。

注释

要将子串与字符串匹配，应使用百分号通配符 (%*expression*)。例如，`SELECT ... WHERE Description SIMILAR TO 'car'` 仅与 car 匹配，与 sportscar 不匹配。但是，`SELECT ...`

`WHERE Description SIMILAR TO '%car'` 则与 car、sportscar 和所有以 car 结尾的字符串都匹配。

当只依据子字符类进行匹配时，必须包含外部方括号以及用于子字符类的方括号。例如，*expression* SIMILAR TO '[[:digit:]]')。有关子字符类匹配的详细信息，请参见“正则表达式：特殊子字符类”一节第 23 页。

比较是逐个字符执行的，这与等号 (=) 运算符或其它运算符的逐字符串比较不同。例如，在 UCA 归类（归类设置为 UCA 的 CHAR 或 NCHAR）中进行比较时，`'?'='AE'` 为 true，但 `'?' SIMILAR TO 'AE'` 为 false。

对于逐个字符进行的匹配比较，被搜索的表达式中的每个字符都必须与 SIMILAR TO 模式中的一个字符或一个通配符匹配。

数据库归类和匹配

SIMILAR TO 使用归类来确定字符等同性并计算字符类范围。例如，如果数据库不区分大小写和重音，则匹配项也不区分大小写和重音。范围也同样使用归类排序顺序进行计算。

有关对 LIKE、SIMILAR TO 和 REGEXP 之间如何处理匹配和范围计算进行的比较，请参见“LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页。

国家字符 (NCHAR) 支持

SIMILAR TO 搜索条件可以用于比较 CHAR 和 NCHAR 字符串。这种情况下会执行字符集转换，以便使用共同的数据类型进行比较。然后逐个字符地执行比较。请参见“CHAR 和 NCHAR 之间的比较”一节第 107 页。

通过在引号引起的值之前加前缀 N（例如，`expression SIMILAR TO N'pattern'`），可以将 *expression* 或 *pattern* 指定为 NCHAR 字符串文字。还可以使用 CAST 函数将模式转换为 CHAR 或 NCHAR（例如，`expression SIMILAR TO CAST(pattern AS datatype)`）。

请参见“字符串文字”一节第 11 页和“CAST 函数 [Data type conversion]”一节第 141 页。

另请参见

- “正则表达式概述”一节第 20 页
- “REGEXP 搜索条件”一节第 43 页
- “LIKE 搜索条件”一节第 38 页
- “REGEXP_SUBSTR 函数 [String]”一节第 263 页
- “LIKE、REGEXP 和 SIMILAR TO 搜索条件”一节第 36 页

IN 搜索条件

语法

expression [**NOT**] **IN** { **(** *subquery* **)** | **(** *expression2* **)** | **(** *value-expression1*, ... **)** }

注释

没有 NOT 关键字时，IN 搜索条件按以下规则求值：

- 如果 *expression* 不为 NULL 并且至少等于其中一个值，则值为 TRUE。
- 如果 *expression* 为 NULL 而值列表非空，或者如果至少其中一个值为 NULL 且 *expression* 不等于任何其它值，则值为 UNKNOWN。
- 如果 *expression* 为 NULL 且 *subquery* 不返回任何值，或者如果 *expression* 不为 NULL，所有值均不为 NULL，并且 *expression* 不等于其中任何一个值，则值为 FALSE。

NOT 关键字将 TRUE 转换为 FALSE，或将 FALSE 转换为 TRUE。

搜索条件 *expression* **IN (** *values* **)** 相当于 *expression* = **ANY (** *values* **)**。

搜索条件 *expression* **NOT IN (** *values* **)** 相当于 *expression* <> **ALL (** *values* **)**。

value-expression 参数是采用单值的表达式，它的值可以是字符串、数字、日期或任何其它 SQL 数据类型。

CONTAINS 搜索条件

语法

CONTAINS (*column-name* [,...], *contains-query-string* **)**

contains-query-string :
simple-expression
| *or-expression*

simple-expression :
primary-expression
| *and-expression*

or-expression :
simple-expression { **OR** | **|** } *contains-query-string*

primary-expression :
basic-expression
| **FUZZY "** *fuzzy-expression* **"**
| *and-not-expression*

and-expression :
primary-expression [**AND** | **&**] *simple-expression*

and-not-expression :
primary-expression [**AND** | **&**] { **NOT** | **-** } *basic-expression*

basic-expression :
term
| *phrase*
| **(** *contains-query-string* **)**
| *near-expression*

fuzzy-expression :
term
| *fuzzy-expression term*

term :
simple-term
| *prefix-term*

prefix-term :
*simple-term**

phrase :
" *phrase-string* **"**

near-expression :
term **NEAR[***distance***]** *term*
| *term* { **NEAR** | **~** } *term*

phrase-string :
term
| *phrase-string term*

simple-term : A string separated by whitespace and special characters that represents a single indexed term (word) to search for.

distance : a positive integer

参数

- **and-expression** 使用 *and-expression* 来指定 *primary-expression* 和 *simple-expression* 都必须出现在文本索引中。

  缺省情况下，如果在术语或表达式之间未指定任何运算符，则假定为 *and-expression*。例如，`'a b'` 被解释为 `'a AND b'`。

  和号 (&) 可用来代替 AND，并且可以紧靠表达式或术语的任何一侧（例如，`'a &b'`）。

  请参见“对于特殊字符允许的语法”一节第 50 页。

- **and-not-expression** 使用 *and-not-expression* 来指定 *primary-expression* 必须存在于文本索引中，但 *basic-expression* *一定不能* 出现在文本索引中。这也称为**取非**。

  如果使用连字符表示取非，则在连字符左侧必须留有一个空格，且连字符右侧必须与术语邻接；否则不将连字符解释为取非。例如，`'a -b'` 等同于 `'a AND NOT b'`；而对于 `'a - b'`，连字符被忽略，该字符串等同于 `'a AND b'`。`'a-b'` 等同于短语 `'"a b"'`。请参见“对于连字符 (-) 允许的语法”一节第 49 页。

- **or-expression** 使用 *or-expression* 来指定 *simple-expression* 或 *contains-query-string* 至少得有一个存在于文本索引中。例如，`'a|b'` 被解释为 `'a OR b'`。请参见“对于特殊字符允许的语法”一节第 50 页。

- **fuzzy-expression** 使用 *fuzzy-expression* 来查找与您指定的术语相类似的术语。只有 NGRAM 文本索引上支持模糊匹配。请参见“模糊搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **near-expression** 使用 *near-expression* 来搜索彼此邻近的术语。这也称为**邻近搜索**。例如，`'b NEAR[5] c'` 搜索相互间隔五个或五个以下术语的 b 和 c 实例。术语的顺序并不重要；`'b NEAR c'` 等同于 `'c NEAR b'`。

  如果指定 NEAR 时未指定 *distance*，则应用缺省值 10 个术语。

可指定代字号 (~) 来代替 NEAR。这相当于指定 NEAR 时未指定距离，所以应用了缺省值 10 个术语。

NEAR 表达式不可以链接在一起（例如，`'a NEAR[1] b NEAR[1] c'`）。

请参见“对于特殊字符允许的语法”一节第 50 页和“邻近搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **prefix-term** 使用 *prefix-term* 来搜索以特定前缀开始的术语。例如，`'datab*'` 用于搜索以 datab 开始的任何术语。这也称为**前缀搜索**。在前缀搜索中，执行匹配的是星号左侧的术语部分。请参见“对于星号 (*) 允许的语法”一节第 49 页和“前缀搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》。

注释

CONTAINS 搜索条件将一个列列表和 *contains-query-string* 作为参数。它可以用在任何能够指定搜索条件（也称作谓语）的地方，并返回 TRUE 或 FALSE。*contains-query-string* 必须是含有在查询时就已知的值的常量字符串或变量。

如果指定了多个列，则它们全都必须引用单个基表；一个文本索引不能跨多个基表。基表可以在 FROM 子句中直接引用，也可以用在视图或派生表中，前提是该视图或派生表不使用 DISTINCT、GROUP BY、ORDER BY、UNION、INTERSECT、EXCEPT 或行限制。

以下警告适用于在查询字符串中使用非字母数字字符的情况：

- 在术语中间使用星号会返回错误。
- 不要在 *fuzzy-expression* 中使用非字母数字（包括特殊字符），因为它们会被看作是空白并用作术语中断符。
- 如果可能，不要在查询字符串中包含不是特殊字符的非字母数字字符。任何不是特殊字符的非字母数字字符都会使包含该字符的术语被视为一个短语，从而在字符所在的位置断开术语。例如，`'things we've done'` 被解释为 `'things "we ve" done'`。

在短语内，星号是唯一被继续解释为特殊字符的特殊字符。短语中所有其它特殊字符都被视作空白并用作术语中断符。

对 *contains-query-string* 进行解释主要包括两步：

- **第 1 步：解释运算符和优先级** 在执行此步骤期间，将关键字解释为运算符，并应用优先级规则。请参见“CONTAINS 搜索条件中的运算符优先级”一节第 48 页。
- **第 2 步：应用文本配置对象设置** 在执行此步骤期间，将文本配置对象设置应用于术语。例如，在 NGRAM 文本索引中，术语被分解为 n 元语法词表示。在执行此步骤期间，将删除超过术语长度设置或在非索引字表中的查询术语。有关删除术语时如何解释查询字符串的详细信息，请参见“文本配置对象示例”一节 《SQL Anywhere 服务器 - SQL 的用法》。

CONTAINS 搜索条件中的运算符优先级

在进行查询计算期间，使用以下优先级顺序计算表达式：

1. FUZZY、NEAR

2. AND NOT
3. AND
4. OR

对于星号 (*) 允许的语法

星号被用于**前缀搜索**。星号可以出现在查询字符串的结尾处，或在其后紧跟空格、和号、竖线、右括号或右引号。其它所有的星号用法都会返回错误。

下表显示了允许的星号用法：

查询字符串	等同于：	解释为：
`'th*'`		查找所有以 th 开头的术语。
`'th*&best'`	`'th* AND best'` 和 `'th* best'`	查找所有以 th 开头的术语，并查找术语 best。
`'th*|best'`	`'th* OR best'`	查找所有以 th 开头的术语，或查找术语 best
`'very&(best|th*)'`	`'very AND (best OR th*)'`	查找术语 very，并查找术语 best 或任何以 th 开头的术语。
`'"fast auto*"'`		查找术语 fast，随后紧跟以 auto 开头的术语。
`'"auto* price"'`		查找以 auto 开头的术语，随后紧跟术语 price。

注意
对包含星号的查询字符串的解释视文本配置对象设置而异。请参见“前缀搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》。

对于连字符 (-) 允许的语法

连字符可用于对术语或表达式**取非**，等同于 NOT。是否将连字符解释为取非要取决于连字符在查询字符串中的位置。例如，如果连字符紧靠在术语或表达式之前，则将连字符解释为取非。如果将连字符嵌入术语内，则将连字符解释为连字符。

用于取非的连字符的前面必须有一个空白，而后面紧跟表达式。

当用在模糊表达式的短语中时，连字符被视为空白并用作术语分隔符。

下表显示了对于连字符允许的语法：

查询字符串	等同于：	解释为：
'the -best'	'the AND NOT best', 'the AND -best', 'the & -best', 'the NOT best'	查找术语 the，但不查找术语 best。
'the -(very best)'	'the AND NOT (very AND best)'	查找术语 the，但不查找术语 very 和 best。
'the -"very best"'	'the AND NOT "very best"'	查找术语 the，但不查找短语 very best。
'alpha-numerics'	'"alpha numerics"'	查找术语 alpha，随后紧跟术语 numerics。
'wild - west'	'wild west'和 'wild AND west'	查找术语 wild，并查找术语 west。

对于特殊字符允许的语法

下表显示了对于除星号和连字号之外（这两个特殊字符在前面两节中已进行介绍）的其它所有特殊字符所允许的语法：“对于星号 (*) 允许的语法”一节第 49 页和“对于连字符 (-) 允许的语法”一节第 49 页。

如果在短语中找到了这些字符，则不认为它们是特殊字符，并将它们删除。

注意
指定字符串文字时的限制同样适用于查询字符串。例如，必须对撇号进行转义等。有关格式化字符串文字的详细信息，请参见“字符串文字”一节第 11 页。

字符或语法	用法示例和注释
和号 (&)	和号等同于 AND，可指定为以下形式： ● 'a & b' ● 'a &b' ● 'a&b' ● 'a& b'
竖线 (|)	竖线等同于 OR，可指定为以下形式： ● 'a|b' ● 'a |b' ● 'a | b' ● 'a| b'

字符或语法	用法示例和注释
双引号 (")	双引号用于包含那些顺序和相对距离都很重要的一连串术语。例如，在查询字符串 `'learn "full text search"'` 中，"full text search" 是一个短语。在此示例中，learn 可以放置在短语的前面或后面，或者位于另一个列中（如果文本索引基于不止一个列构建的），但必须在单个列中找到完全一样的短语。
括号 ()	如果表达式的计算顺序与缺省顺序不同，则使用括号指定表达式的计算顺序。例如，`'a AND (b|c)'` 被解释为 a 和 b 或 c。 有关缺省计算顺序的详细信息，请参见“CONTAINS 搜索条件中的运算符优先级”一节第 48 页。
代字号 (~)	代字号等同于 NEAR，没有特殊的语法规则。查询字符串 `'full~text'` 等同于 `'full NEAR text'`，可解释为：术语 full 和术语 text 的距离在十个术语范围之内。
方括号 [ ]	方括号与关键字 NEAR 结合使用以包含 *distance*。方括号的其它用法会返回错误。

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本配置对象设置”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本配置对象示例”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “FROM 子句”一节第 582 页
- “sa_char_terms 系统过程”一节第 798 页
- “sa_nchar_terms 系统过程”一节第 866 页

EXISTS 搜索条件

语法

EXISTS (*subquery* **)**

注释

如果子查询结果至少包含一行，则 EXISTS 搜索条件为 TRUE；如果子查询结果一行也不包含，则条件为 FALSE。EXISTS 搜索条件不能为 UNKNOWN。

IS NULL 和 IS NOT NULL 搜索条件

语法

expression **IS** [**NOT**] **NULL**

注释

无 NOT 关键字时，如果表达式值为 NULL，则 IS NULL 搜索条件为 TRUE，其它情况下为 FALSE。NOT 关键字使搜索条件的含义相反。

真值搜索条件

语法

IS [**NOT**] *truth-value*

注释

无 NOT 关键字时，如果 *condition* 的值为提供的 *truth-value*（必须是 TRUE、FALSE 或 UNKNOWN 中的一个），则搜索条件为 TRUE。否则，条件值为 FALSE。NOT 关键字会使搜索条件的含义相反，但对于 UNKNOWN，其含义保持不变。

标准和兼容性

- 服务商扩充。

触发器操作条件

语法

trigger-operation:
INSERTING
| **DELETING**
| **UPDATING** [**(** *column-name-string* **)**]
| **UPDATE (** *column-name* **)**

注释

触发器操作条件只能用在触发器中，根据激发触发器的操作种类来完成操作。

UPDATING 的参数是用引号引起的字符串（例如，`UPDATING( 'mycolumn' )`）。UPDATE 的参数是一个标识符（例如，`UPDATE( mycolumn )`）。两种版本可以互操作，包括这两种版本是为了与其他供应商的 DBMS 的 SQL 方言兼容。

如果您提供一个 UPDATING 或 UPDATE 函数，则还必须在 CREATE TRIGGER 语句中提供一个 REFERENCING 子句以避免出现语法错误。

示例

下面的触发器显示一条消息，说明哪个操作导致激发触发器。

```
CREATE TRIGGER tr BEFORE INSERT, UPDATE, DELETE
ON sample_table
REFERENCING OLD AS t1old
FOR EACH ROW
BEGIN
    DECLARE msg varchar(255);
    SET msg = 'This trigger was fired by an ';
    IF INSERTING THEN
        SET msg = msg || 'insert'
    ELSEIF DELETING THEN
        set msg = msg || 'delete'
    ELSEIF UPDATING THEN
        set msg = msg || 'update'
    END IF;
    MESSAGE msg TO CLIENT
END;
```

另请参见

- “BEGIN 语句”一节第 395 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》

三值逻辑

下表显示了 SQL 中的 AND、OR、NOT 和 IS 逻辑运算符如何以三值逻辑的方式操作。

AND 运算符

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR 运算符

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT 运算符

TRUE	FALSE	UNKNOWN
FALSE	TRUE	UNKNOWN

IS 运算符

IS	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE
UNKNOWN	FALSE	FALSE	TRUE

显式选择性估计

SQL Anywhere 利用统计信息来确定每个语句最有效的执行策略。SQL Anywhere 会自动收集和更新这些统计信息。这些统计信息永久存储在数据库的系统表 ISYSCOLSTAT 中。处理一个语句时所收集的统计信息对于搜索后续语句的有效执行方式很有用。

有时候，统计信息可能变得不准确，或者相关的统计信息不可用。最有可能发生这种情况的时候是：添加、更新或删除了大量数据后，只执行了很少的查询。在这种情况下，最好执行 CREATE STATISTICS 语句。请参见“CREATE STATISTICS 语句”一节第 491 页。

如果某个特定的执行计划存在问题，可以使用优化程序提示来要求使用特定索引。有关详细信息，请参见“FROM 子句”一节第 582 页。

然而，在极少数情况下，这些措施可能无效。这种情况下，有时可以通过提供显式选择性估计来提高性能。

对于某个潜在执行计划中的每张表，优化程序必须估计将进入结果集的行数。如果您知道某条件的成功率与优化程序估计的不一样，可以在搜索条件中显式地提供用户的估计。

估计值是一个百分比。它可以是正整数，也可以包含小数部分。

小心
应尽量避免在持续使用的语句中提供显式估计。如果数据改变，显式估计可能会变得不准确并强制优化程序选择低效率的计划。如果您确实要使用显式选择性估计，那么请确保数字的精确性。例如，请不要提供值 0% 或 100% 来强制使用索引。

可以通过将数据库选项 user_estimates 设置为 Off 来禁用用户估计。user_estimates 的缺省值为 Override-Magic，这表示只有在优化程序对条件使用 MAGIC（缺省）选择性值时才可以使用用户提供的选择性估计。当优化程序无法准确预测一个谓语的选择性时，它的最后一个手段就是利用 MAGIC 值。

有关禁用用户定义的选择性估计的详细信息，请参见“user_estimates 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关统计信息的详细信息，请参见“优化程序估计值和列统计信息”一节 《SQL Anywhere 服务器 - SQL 的用法》。

示例

以下查询估计 1% 的 ShipDate 值晚于 2001/06/30：

```
SELECT  ShipDate
   FROM  SalesOrderItems
WHERE ( ShipDate > '2001/06/30', 1 )
ORDER BY ShipDate DESC;
```

以下查询估计 0.5% 的行会满足条件：

```
SELECT *
   FROM Customers c, SalesOrders o
WHERE (c.ID = o.CustomerID, 0.5);
```

小数部分值为连接和大表提供了更准确的用户估计值。

特殊值

特殊值可用在表达式中，也可在创建表时用作列缺省值。

尽管有些特殊值可以查询，但有些特殊值只能作为列的缺省值使用。例如，**USER**、**LAST USER**、**TIMESTAMP** 和 **UTC TIMESTAMP** 只能作为缺省值。

CURRENT DATABASE 特殊值

CURRENT DATABASE 返回当前数据库的名称。

数据类型

STRING

另请参见

- “表达式”一节第 16 页

CURRENT DATE 特殊值

CURRENT DATE 返回当前年月日。

数据类型

DATE

另请参见

- “表达式”一节第 16 页
- “TIME 数据类型”一节第 100 页

CURRENT PUBLISHER 特殊值

CURRENT PUBLISHER 返回其中包含 SQL Remote 复制中数据库发布者用户 ID 的字符串。

数据类型

STRING

注释

CURRENT PUBLISHER 可在具有字符数据类型的列中用作缺省值。

另请参见

- “表达式”一节第 16 页
- “SQL Remote 复制设计和设置” 《SQL Remote》

CURRENT TIME 特殊值

返回当前小时、分钟、秒和秒的小数部分。

数据类型

TIME

注释

秒的小数部分存储到 6 个小数位。当前时间的准确性受系统时钟准确性的限制。

另请参见

- “表达式”一节第 16 页
- “TIME 数据类型”一节第 100 页

CURRENT TIMESTAMP 特殊值

CURRENT TIMESTAMP 组合 CURRENT DATE 和 CURRENT TIME，以构成包含年月日、小时、分钟、秒和秒的小数部分的 TIMESTAMP 值。秒的小数部分存储到 3 个小数位。其准确性受系统时钟准确性的限制。

与 DEFAULT TIMESTAMP 不同，以 DEFAULT CURRENT TIMESTAMP 声明的列不必包含唯一值。如果需要唯一性，则请考虑使用 DEFAULT TIMESTAMP。

CURRENT TIMESTAMP 返回的信息与 GETDATE 和 NOW 函数返回的信息相同。

CURRENT_TIMESTAMP 与 CURRENT TIMESTAMP 等价。

注意
DEFAULT CURRENT TIMESTAMP 和 DEFAULT TIMESTAMP 之间的主要差异在于，DEFAULT CURRENT TIMESTAMP 仅在 INSERT 时设置，而 DEFAULT TIMESTAMP 则在 INSERT 和 UPDATE 时设置。

数据类型

TIMESTAMP

另请参见

- “CURRENT TIME 特殊值”一节第 57 页
- “TIMESTAMP 特殊值”一节第 61 页
- “表达式”一节第 16 页
- “TIMESTAMP 数据类型”一节第 100 页
- “GETDATE 函数 [Date and time]”一节第 202 页
- “NOW 函数 [Date and time]”一节第 248 页

CURRENT USER 特殊值

CURRENT USER 返回一个包含当前连接的用户 ID 的字符串。

数据类型

STRING

注释

CURRENT USER 可在具有字符数据类型的列中用作缺省值。

在 UPDATE 时，不更改具有 CURRENT USER 缺省值的列。CURRENT_USER 与 CURRENT USER 等同。

另请参见

- “表达式”一节第 16 页

CURRENT UTC TIMESTAMP 特殊值

CURRENT UTC TIMESTAMP 组合根据服务器时区调整值调整后的 CURRENT DATE 和 CURRENT TIME，以构成包含年月日、小时、分钟、秒和秒的小数部分的协调通用时间（Coordinated Universal Time，简称 UTC）TIMESTAMP 值。此功能允许在任何时区下都以一致的时间参照输入数据。

数据类型

TIMESTAMP

另请参见

- “TIMESTAMP 数据类型”一节第 100 页
- “UTC TIMESTAMP 特殊值”一节第 62 页
- “CURRENT TIMESTAMP 特殊值”一节第 57 页
- “truncate_timestamp_values 选项 [数据库] [MobiLink 客户端]”一节 《SQL Anywhere 服务器 - 数据库管理》

LAST USER 特殊值

LAST USER 是上次修改行的用户的名称。

数据类型

字符串

注释

LAST USER 可在具有字符数据类型的列中用作缺省值。

在 INSERT 时，此常量与 CURRENT USER 的效果相同。在 UPDATE 时，如果未显式修改具有 LAST USER 缺省值的列，该列会变为当前用户的名称。

与 DEFAULT TIMESTAMP 组合时，LAST USER 缺省值可用于记录（在单独的列中）用户和上次更改行的日期和时间。

另请参见

- “CURRENT USER 特殊值”一节第 58 页
- “CURRENT TIMESTAMP 特殊值”一节第 57 页
- “CREATE TABLE 语句”一节第 497 页

SQLCODE 特殊值

SQLCODE 指出最近执行的 SQL 语句的配置。

数据类型

有符号的 INTEGER

注释

数据库服务器会为其执行的每个 SQL 语句设置 SQLSTATE 和 SQLCODE。SQLCODE 特定于产品（例如，MobiLink 拥有其自己的 SQLCODE），可用于了解有关 SQLSTATE 的附加信息。例如，除 100 之外的正值指示特定于产品的 *warning* 状态。负值指示特定于产品的 *exception* 状态。值 100 表示 "无数据"（例如，通过游标在结果集的末端读取）。

SQLSTATE 和 SQLCODE 相关联：每个 SQLCODE 对应一个 SQLSTATE，而每个 SQLSTATE 对应一个或多个 SQLCODE。

要返回与 SQLCODE 相关联的错误状态，可以使用 ERRORMSG 函数。请参见“ERRORMSG 函数 [Miscellaneous]”一节第 186 页。

注意
SQLSTATE 是 SQL 语句结果的首选状态指示符。请参见“SQLSTATE 特殊值”一节第 60 页。

另请参见

- “SQLSTATE 特殊值”一节第 60 页
- “按 SQLCODE 排序的 SQL Anywhere 错误消息”一节 《错误消息》
- “表达式”一节第 16 页

标准和兼容性

对于 SQLCODE，在 ANSI SQL/1992 标准中就已不再被支持，并已从 SQL/1999 中完全删除。为了实现应用程序的向后兼容性，SQLCODE 值继续保留在 SQL Anywhere 中。SQLSTATE 是首选的状态指示符。

SQLSTATE 特殊值

SQLSTATE 指示最近执行的 SQL 语句是否产生了成功、错误或警告状态。

数据类型

字符串

注释

数据库服务器会为其执行的每个 SQL 语句设置 SQLSTATE 和 SQLCODE。SQLSTATE 是一个字符串，用以指示最近执行的 SQL 语句是否产生了成功、警告或错误状态。

每个 SQLSTATE 都代表所有平台通用的错误，这些错误通常包含非特定于产品的说明文字。SQLSTATE 值的格式是一个双字符类值，后面跟一个三字符子类值。关于类和子类值的 SQLSTATE 合规性的原则在 ISO/ANSI SQL 标准中有介绍。

除了以下补充和例外，SQL Anywhere 符合 ISO/ANSI SQLSTATE 约定：

类和子类	条件
01WC*x*	与字符集转换相关的警告
38*xxx*	外部函数异常
42X*xx*	语法错误：表达式
42R*xx*	语法错误：参照完整性（例如，尝试创建第 2 个主键）
42W*xx*	语法错误：通用
42U*xx*	语法错误：重复的、未定义的或不明确的对象引用
42Z*xx*	访问违规
54W*xx*	超出产品限制
55W*xx*	对象未处于操作成功所需的状态
57*xxx*	资源不可用或运算符干预
5R*xxx*	SQL Remote 错误
WB*xxx*	联机备份错误
WI*xxx*	内部数据库错误
WP*xxx*	过程、变量等的错误
WL*xxx*	装载和/或卸载错误

类和子类	条件
WW*xxx*	其它特定于 SQL Anywhere 的错误/警告（包括系统故障）
WO*xxx*	与远程数据访问功能相关的错误
WJ*xxx*	JCS 和 JDBC 相关的错误
WC*xxx*	字符转换错误
WX*xxx*	与 XML 相关的错误
WT*xxx*	与文本相关的错误

成功完成的类为 '00*xxx*'（例如，'00000'）。

SQLSTATE 和 SQLCODE 相关联：每个 SQLCODE 对应一个 SQLSTATE，而每个 SQLSTATE 对应一个或多个 SQLCODE。

要返回与 SQLSTATE 相关的错误状态，可以使用 ERRORMSG 函数。请参见“ERRORMSG 函数 [Miscellaneous]”一节第 186 页。

要查看 SQL Anywhere 所使用的 SQLSTATE 值，请参见“按 SQLSTATE 排序的 SQL Anywhere 错误消息”一节 《错误消息》。

另请参见

- “SQLCODE 特殊值”一节第 59 页
- “表达式”一节第 16 页

标准和兼容性

- **SQL/2003：核心特性** 以值 '0' 至 '4' 和 'A' 至 'H' 开头的 SQLSTATE 类（前两个字符）由 ANSI 标准定义。其它类则是由具体实现自行定义。同样，以值 '0' 至 '4' 和 'A' 至 'H' 开头的子类值由 ANSI 标准定义。这些范围之外的子类值由具体实现自行定义。

TIMESTAMP 特殊值

TIMESTAMP 指示表中各行上次的修改时间。当用 DEFAULT TIMESTAMP 声明列时，会提供一个缺省的插入值，每当更新行时，该值都用当前日期和时间更新。

数据类型

TIMESTAMP

注释

以 DEFAULT TIMESTAMP 声明的列包含唯一值，以便应用程序能检测到同一行几乎同时发生的更新。如果当前时间戳的值与上一个值相同，它将按 default_timestamp_increment 选项的值相应递增。

在 SQL Anywhere 中，您可以根据 default_timestamp_increment 选项自动截断时间戳的值。这对于同其它记录精度较低的时间戳值的数据库软件保持兼容很有用。

全局变量 @@dbts 返回一个 TIMESTAMP 值，该值表示上次使用 DEFAULT TIMESTAMP 为列生成的值。

注意
DEFAULT TIMESTAMP 和 DEFAULT CURRENT TIMESTAMP 之间的主要差异在于，DEFAULT CURRENT TIMESTAMP 仅在 INSERT 时设置，而 DEFAULT TIMESTAMP 则在 INSERT 和 UPDATE 时设置。

另请参见

- “TIMESTAMP 数据类型”一节第 100 页
- “CURRENT TIMESTAMP 特殊值”一节第 57 页
- “CURRENT UTC TIMESTAMP 特殊值”一节第 58 页
- “default_timestamp_increment 选项 [数据库] [MobiLink 客户端]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “truncate_timestamp_values 选项 [数据库] [MobiLink 客户端]”一节 《SQL Anywhere 服务器 - 数据库管理》

USER 特殊值

USER 返回包含当前连接的用户 ID 的字符串。

数据类型

STRING

注释

USER 可在具有字符数据类型的列中用作缺省值。

在 UPDATE 时，不更改具有 USER 缺省值的列。

另请参见

- “表达式”一节第 16 页
- “CURRENT USER 特殊值”一节第 58 页
- “LAST USER 特殊值”一节第 58 页

UTC TIMESTAMP 特殊值

UTC TIMESTAMP 表示上次修改表内各行时的协调通用时间 (UTC)。

以 DEFAULT UTC TIMESTAMP 声明某列时，会将一个缺省值提供给插入操作，每当更新行时，该值都会更新为当前 UTC 日期和时间。

数据类型

TIMESTAMP

注释

以 DEFAULT UTC TIMESTAMP 声明的列包含唯一值，以便应用程序能够检测到同一行几乎同时发生的更新。如果当前 UTC 时间戳的值与上一个值相同，它将按 default_timestamp_increment 选项的值相应递增。

在 SQL Anywhere 中，您可以根据 default_timestamp_increment 选项自动截断 UTC 时间戳的值。这对于同其它记录精度较低的时间戳值的数据库软件保持兼容很有用。

注意
DEFAULT UTC TIMESTAMP 和 DEFAULT CURRENT UTC TIMESTAMP 之间的主要差异在于，DEFAULT CURRENT UTC TIMESTAMP 仅在 INSERT 时设置，而 DEFAULT UTC TIMESTAMP 在 INSERT 和 UPDATE 时设置。

另请参见

- “TIMESTAMP 数据类型”一节第 100 页
- “CURRENT UTC TIMESTAMP 特殊值”一节第 58 页
- “TIMESTAMP 特殊值”一节第 61 页
- “default_timestamp_increment 选项 [数据库] [MobiLink 客户端]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “truncate_timestamp_values 选项 [数据库] [MobiLink 客户端]”一节 《SQL Anywhere 服务器 - 数据库管理》

变量

SQL Anywhere 支持三个级别的变量：

- **局部变量** 这些是使用 DECLARE 语句在过程或批处理中的复合语句内定义的变量。它们只存在于复合语句内。
- **连接级变量** 这些是使用 CREATE VARIABLE 语句定义的变量。它们属于当前连接，当断开现有数据库连接或使用 DROP VARIABLE 语句时，它们就会消失。
- **全局变量** 这些是系统提供的具有系统提供值的变量。所有全局变量的名称都以两个 @ 符号开头。例如，全局变量 **@@version** 的值是数据库服务器的当前版本号。用户不能定义全局变量。

本地和连接级变量由用户声明，可用在 SQL 语句的过程或批处理中来保存信息。而全局变量是系统提供的变量，这些变量可以提供系统提供的值。

另请参见

- “TIMESTAMP 数据类型”一节第 100 页
- “CREATE VARIABLE 语句”一节第 517 页

局部变量

SQL Anywhere 支持局部变量。局部变量是使用 DECLARE 语句声明的，它们只能用在复合语句（即用关键字 BEGIN 和 END 括起来的语句）内。在 SQL Anywhere 中，每个 DECLARE 语句只能声明一个变量。

如果在复合语句内执行 DECLARE 语句，作用域仅限于该复合语句。

这类变量最初被设置为 NULL。变量值可以用 SET 语句进行设置，或用带有 INTO 子句的 SELECT 语句指派。

DECLARE 语句的语法如下：

```
DECLARE variable-name data-type
```

局部变量可作为参数传递给过程，只要该过程是从复合语句内调用的。

示例

下面的批处理语句说明了如何使用局部变量。

```
BEGIN
   DECLARE local_var INT;
   SET local_var = 10;
   MESSAGE 'local_var = ', local_var TO CLIENT;
END
```

从 Interactive SQL 运行此批处理语句时，Interactive SQL 的 [消息] 选项卡中会出现 `local_var = 10` 消息。

变量 local_var 在声明此变量的复合语句外部不存在。下面的批处理语句无效，会出现 `column not found` 错误。


```
-- This batch is invalid.
BEGIN
   DECLARE local_var INT;
   SET local_var = 10;
END;
MESSAGE 'local_var = ', local_var TO CLIENT;
```

下例说明了如何用带有 INTO 子句的 SELECT 语句设置局部变量的值：

```
BEGIN
   DECLARE local_var INT;
   SELECT 10 INTO local_var;
   MESSAGE 'local_var = ', local_var TO CLIENT;
END
```

从 Interactive SQL 运行此批处理语句时，数据库服务器的消息窗口中会出现 `local_var = 10` 消息。

有关批处理和局部变量作用域的详细信息，请参见“Transact-SQL 过程中的变量”一节 《SQL Anywhere 服务器 - SQL 的用法》。

连接级变量

连接级变量用 CREATE VARIABLE 语句声明。连接级变量可作为参数传递给过程。

CREATE VARIABLE 语句的语法如下：

CREATE VARIABLE *variable-name data-type*

创建的变量最初设置为 NULL。连接级变量的值可用与局部变量相同的方法设置：使用 SET 语句或带有 INTO 子句的 SELECT 语句。

在连接终止前或用 DROP VARIABLE 语句显式删除变量前，连接级变量会一直存在。以下语句会删除变量 con_var：

```
DROP VARIABLE con_var;
```

示例

下面的 SQL 批处理语句说明如何使用连接级变量。

```
CREATE VARIABLE con_var INT;
SET con_var = 10;
MESSAGE 'con_var = ', con_var TO CLIENT;
```

从 Interactive SQL 运行此批处理语句时，数据库服务器的消息窗口中会出现 `con_var = 10` 消息。

全局变量

全局变量的值由数据库服务器设置。例如，全局变量 @@version 的值是数据库服务器的当前版本号。

全局变量的名称前有两个 @ 符号，以便与局部和连接级变量加以区分。例如，@@error 和 @@rowcount 都是全局变量。用户不能创建全局变量，也不能直接更新全局变量的值。

有些全局变量（如 @@identity）保存特定于连接的信息，因此具有特定于连接的值。其它变量（如 @@connections）则具有所有连接共有的值。

全局变量和特殊常量

特殊常量（例如 CURRENT DATE、CURRENT TIME、USER 和 SQLSTATE）类似于全局变量。

以下语句检索版本全局变量的值。

```
SELECT @@version;
```

在过程和触发器中，全局变量可以选入变量列表。以下过程用参数 *ver* 返回服务器版本号。

```
CREATE PROCEDURE VersionProc ( OUT ver VARCHAR(100) )
 BEGIN
   SELECT @@version
   INTO ver;
 END;
```

在嵌入式 SQL 中，全局变量可以选入主机变量列表。

全局变量列表

下表列出了 SQL Anywhere 中可用的全局变量。有些全局变量是为与 Transact-SQL 相兼容而提供的，它们会返回固定值 0、1 或 NULL（见下文）。

变量名称	含义
@@char_convert	0（用于与 Transact-SQL 兼容。）
@@client_csid	-1（用于与 Transact-SQL 兼容。）
@@client_csname	NULL（用于与 Transact-SQL 兼容。）
@@connections	自服务器上次启动后的登录数。
@@cpu_busy	0（用于与 Transact-SQL 兼容。）
@@dbts	一个 TIMESTAMP 类型的值，表示上次生成的值，用于以 DEFAULT TIMESTAMP 定义的所有列。
@@error	检查最近执行的语句是成功还是失败的 Transact-SQL 错误代码。如果前一个事务成功，则返回 0。如果前一个事务未成功，则返回由系统生成的上一个错误编号。要查看 @@error 返回的值的说明，请参见“Transact-SQL 过程中的错误处理”一节 《SQL Anywhere 服务器 - SQL 的用法》。 如果发生错误，`if @@error != 0 return` 这样的语句会导致退出。每个语句（包括 PRINT 语句或 IF 测试）都重置 @@error，因此必须在想要验证其是否成功的语句之后立即执行状态检查。

变量名称	含义
@@fetch_status	包含上次读取语句所产生的状态信息。此功能与 @@sqlstatus 相同，只不过它返回的值不同。这是为了与 Microsoft SQL Server 兼容。 @@fetch_status 可包含以下值： ● 0 - 读取语句成功完成。 ● -1 - 读取语句导致错误。 ● -2 - 结果集中没有其它数据。
@@identity	上次用 INSERT 或 SELECT INTO 语句插入到任何 IDENTITY 或 DEFAULT AUTOINCREMENT 列中的值。请参见“@@identity 全局变量”一节第 68 页。
@@idle	0（用于与 Transact-SQL 兼容。）
@@io_busy	0（用于与 Transact-SQL 兼容。）
@@isolation	当前连接的隔离级别。@@isolation 采用活动级别的值。
@@langid	当前连接所用语言的唯一语言 ID。
@@language	连接所用语言的名称。
@@max_connections	对于个人服务器，为可与服务器建立的同步连接的最大数目，即为 10。对于网络服务器，为活动客户端（而非数据库连接，因为每个客户端可支持多个连接）的最大数目。
@@maxcharlen	CHAR 字符集中的最大字符长度（以字节为单位）。
@@ncharsize	NCHAR 字符集中的最大字符长度（以字节为单位）。
@@nestlevel	-1（用于与 Transact-SQL 兼容。）
@@pack_received	0（用于与 Transact-SQL 兼容。）
@@pack_sent	0（用于与 Transact-SQL 兼容。）
@@packet_errors	0（用于与 Transact-SQL 兼容。）
@@procid	当前执行的过程的存储过程 ID。

变量名称	含义
@@rowcount	受上一语句影响的行数。应在该语句后立即检查 @@rowcount 的值。 插入、更新和删除会将 @@rowcount 设置为受影响的行数。 在带有游标的情况下，@@rowcount 表示从游标结果集返回给客户端的累计行数（截止到上次读取请求）。 任何不影响行的语句（如 IF 语句）都不将 @@rowcount 重置为零。
@@servername	当前数据库服务器的名称。
@@spid	当前连接的连接句柄。该值与 sa_conn_info 过程显示的值相同。
@@sqlstatus	包含上次读取语句所产生的状态信息。@@sqlstatus 可以包含下列值： ● 0 - 读取语句成功完成。 ● 1 - 读取语句导致错误。 ● 2 - 结果集中没有其它数据。
@@textsize	返回 SET TEXTSIZE 选项的当前值，它指定了用选择语句返回的文本和图形数据的最大长度（以字节为单位）。缺省设置为 32765，这是用 READTEXT 可返回的最大字节字符串。也可用 SET 语句设置该值。
@@thresh_hysteresis	0（用于与 Transact-SQL 兼容。）
@@timeticks	0（用于与 Transact-SQL 兼容。）
@@total_errors	0（用于与 Transact-SQL 兼容。）
@@total_read	0（用于与 Transact-SQL 兼容。）
@@total_write	0（用于与 Transact-SQL 兼容。）
@@tranchained	当前事务模式；0（非链接模式）或 1（链接模式）。
@@trancount	事务的嵌套级别。批处理中的各 BEGIN TRANSACTION 都增加事务计数。
@@transtate	-1（用于与 Transact-SQL 兼容。）
@@version	SQL Anywhere 当前版本的版本号。

@@identity 全局变量

@@identity 变量保存插入到 IDENTITY 列或 DEFAULT AUTOINCREMENT 列中的最新值，如果最新插入是在不含这些列的表中进行的，则为零。

值 @@identity 特定于连接，并且随着每次向表中插入行而递增。如果一个语句插入多行，则 @@identity 反映上次插入的行的 IDENTITY 值。如果受影响的表不包含 IDENTITY 列，则 @@identity 设置为 0。

INSERT 或 SELECT INTO 语句的失败，或者包含 INSERT 或 SELECT INTO 语句的事务的回退，都不会影响 @@identity 的值。@@identity 会保留上次插入到 IDENTITY 列中的值，即使插入该值的语句未能提交。

@@identity 和触发器

当插入操作导致了参照完整性动作或者触发了触发器时，@@identity 的行为就如同堆栈一样。例如，如果表 T1（具有标识列或自动增量列）的插入操作触发了向表 T2（也具有标识列或自动增量列）中插入一行记录的触发器，那么返回给执行此插入操作的应用程序或过程的值就是插入到 T1 的值。在触发器内，@@identity 在插入到 T2 前具有 T1 值，插入后具有 T2 值。如果触发器需要访问这两个值，它可以将其复制到局部变量。

注释

注释用于在 SQL 语句或语句块中附加说明性文字。数据库服务器不执行注释。

SQL Anywhere 支持以下注释指示符：

- **--（双连字符）** 数据库服务器忽略行上的任何其余字符。这是 SQL/2003 注释指示符。通过在 Interactive SQL 中和 Sybase Central 的 [存储的过程] 窗口中按 Ctrl+ 减号，可添加和移除此注释指示符。请参见“Interactive SQL 键盘快捷方式”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **//（双斜线）** 双斜线与双连字符的含义相同。通过在 Interactive SQL 中和 Sybase Central 的 [存储的过程] 窗口中按 Ctrl+ 正斜线，可添加和移除此注释指示符。请参见“Interactive SQL 键盘快捷方式”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **/* ... */（斜线加星号）** 忽略两个注释标记间的任何字符。这两个注释标记可以在同一行中，也可以在不同行中。可以嵌套以此风格指示的注释。这种注释风格也称为 **C 样式注释**。

示例

下例说明了如何使用双连字符注释：

```
CREATE FUNCTION fullname ( firstname CHAR(30),
         lastname CHAR(30))
RETURNS CHAR(61)
-- fullname concatenates the firstname and lastname
-- arguments with a single space between.
BEGIN
   DECLARE name CHAR(61);
   SET name = firstname || ' ' || lastname;
   RETURN ( name );
END;
```

下例说明如何使用 C 样式注释：

```
/* Lists the names and employee IDs of employees
   who work in the sales department. */
CREATE VIEW SalesEmployees AS
   SELECT EmployeeID, Surname, GivenName
   FROM Employees
   WHERE DepartmentID = 200;
```

NULL 值

NULL 值指定了未知或不适用的值。

语法

NULL

注释

NULL 是一个特殊值，它不同于任何数据类型的任何有效值。然而，NULL 值在任何数据类型中都是合法值。NULL 用于表示缺少或不适用的信息。NULL 有两种截然不同的使用情况：

情况	说明
缺少	字段具有值，但该值未知。
不适用	字段不适用于特定的行。

SQL 允许在创建列时带有 NOT NULL 限制。这意味着那些特殊的列不能包含 NULL。

NULL 值将三值逻辑的概念引入 SQL。NULL 值通过任何比较运算符与任何值（包括 NULL 值）进行比较时都是"UNKNOWN"。返回 TRUE 的唯一搜索条件是 IS NULL 谓语。在 SQL 中，只有 WHERE 子句中搜索条件的值为 TRUE 时才选择行；不选择值为 UNKNOWN 或 FALSE 的行。

NULL 值的列空间占用率是每列 1 位，而分配的空间是 8 位的整数倍。在允许 NULL 值的表中，NULL 位的占用在列数的基础上是固定不变的。

IS [NOT] *truth-value* 子句（其中 *truth-value* 是 TRUE、FALSE 或 UNKNOWN 中的一个）可用于选择包含 NULL 值的行。有关此子句的说明，请参见“搜索条件”一节第 33 页。

以下示例中，列 Salary 包含 NULL。

条件	真值	是否选择？
Salary = NULL	UNKNOWN	NO
Salary <> NULL	UNKNOWN	NO
NOT (Salary = NULL)	UNKNOWN	NO
NOT (Salary <> NULL)	UNKNOWN	NO
Salary = 1000	UNKNOWN	NO
Salary IS NULL	TRUE	YES
Salary IS NOT NULL	FALSE	NO
Salary = *expression* IS UNKNOWN	TRUE	YES

同样的规则也适用于比较两个不同表中的列。因此，将两个表连接到一起不会选择其所比较的任何一列含有 NULL 值的行。

用于数字表达式时，NULL 还有一个有趣的属性。任何涉及 NULL 值的数字表达式的结果都为 NULL。这意味着，如果将 NULL 加上一个数字，结果为 NULL，而不是数字。如果想将 NULL 视为 0，则必须使用 **ISNULL(** *expression*, **0)** 函数。

公式化 SQL 查询时的很多常见错误是由 NULL 的行为引起的。必须小心避免这些问题。有关组合搜索条件时三值逻辑的作用的说明，请参见“搜索条件”一节第 33 页。

设置运算符和 DISTINCT 子句

在集合操作（UNION、INTERSECT、EXCEPT）和 DISTINCT 操作中，会针对不同的搜索条件对 NULL 进行不同的处理。包含 NULL 而其余列均相同的行在此类操作中被视为相同。

例如，在表 T1 的所有行中名为 redundant 的列的值均为 NULL，则下面的语句将返回一个单独的行：

```
SELECT DISTINCT redundant FROM T1;
```

权限

必须连接到数据库。

副作用

无。

标准和兼容性

- **SQL/2003**　核心特性。
- **Sybase**　在有些环境中，Adaptive Server Enterprise 将 NULL 按值处理，而 SQL Anywhere 不是这样。例如，在 SQL Anywhere 中，使用以下 WHERE 子句所查询的结果中不包括 c1 列中为 NULL 的行，因为条件的值为 UNKNOWN：

  ```
  WHERE NOT( C1 = NULL )
  ```

  在 Adaptive Server Enterprise 中，条件的值为 TRUE，因此返回这些行。为了兼容，应该使用 IS NULL 而不是比较运算符。

  SQL Anywhere 中的唯一索引可以包含带有 NULL 且其它值相同的行。而 Adaptive Server Enterprise 中的唯一索引不允许这样的条目。

  如果使用 jConnect，则 tds_empty_string_is_null 选项将控制返回的空字符串是作为 NULL 字符串，还是作为包含一个空白字符的字符串。

  有关详细信息，请参见“tds_empty_string_is_null 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

另请参见

- “表达式”一节第 16 页
- “搜索条件”一节第 33 页

示例

以下 INSERT 语句将一个 NULL 值插入到表 Borrowed_book 的 date_returned 列中。

```
INSERT INTO Borrowed_book ( date_borrowed, date_returned, book )
VALUES ( CURRENT DATE, NULL, '1234' );
```

SQL 数据类型

目录

字符数据类型

字符数据类型用于存储字母、数字和其它符号的字符串。

SQL Anywhere 提供了两个字符数据类型类和一些使用这些类型定义的域。

- **CHAR、VARCHAR、LONG VARCHAR** 存储在单字节或多字节字符集中的字符数据，经常被选择用于与数据库中存储的主要语言或其它语言进行最接近的对应。
- **NCHAR、NVARCHAR、LONG NVARCHAR** 以 Unicode 的 UTF-8 编码格式存储的字符数据。可以使用这些类型存储全部 Unicode 代码点，而不必考虑数据库中存储的主要语言或其它语言。
- **TEXT、UNIQUEIDENTIFIERSTR、XML** 基于其它字符数据类型的域。

存储

所有字符数据值都以相同方式存储。缺省情况下，在一段中存储不超过 128 个字节的值。长度超过 128 个字节的值的存储方式是：其 4 个字节的前缀保存在本地数据库页面上，而完整值则存储在一个或多个其它数据库页面上。这些缺省大小由 CREATE TABLE 语句的 INLINE 和 PREFIX 子句控制。

另请参见

- “CREATE TABLE 语句”一节第 497 页
- “string_rtruncation 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》

CHAR 数据类型

CHAR 数据类型存储字符数据，最多可存储 32767 个字节。

语法

CHAR [**(** *max-length* [**CHAR** | **CHARACTER**] **)**]

参数

- **max-length** 字符串的最大长度。如果使用字节长度语义（不将 CHAR 或 CHARACTER 指定为长度的一部分），则长度将以字节为单位，并且长度必须在 1 到 32767 范围内。如果未指定长度，则值为 1。

  如果使用字符长度语义（将 CHAR 或 CHARACTER 指定为长度的一部分），则长度将以字符为单位，并且必须指定 *max-length*。使用字符长度语义时，长度与数据库编码中字符最大长度的乘积不得超过 32767 个字节。下表显示所支持字符集类型的最大长度：

字符集	CHAR 的最大长度
单字节字符集	32767 个字符
双字节字符集	16383 个字符

字符集	CHAR 的最大长度
UTF-8	8191 个字符

注释

可以将多字节字符存储为 CHAR 类型，但除非使用字符长度语义，否则所声明的长度指的是字节长度，而不是字符长度。

也可以将 CHAR 指定为 CHARACTER。无论使用哪种语法，都会将数据类型描述为 CHAR。

虽然 CHAR 和 VARCHAR 属于不同的类型，但它们在语义上是等同的。在 SQL Anywhere 中，CHAR 是一种可变长度类型。在其它关系数据库管理系统中，CHAR 是一种固定长度类型，将使用空白填充数据来使其达到 *max-length* 个存储字节。SQL Anywhere 不会使用空白填充所存储的字符数据。

视所使用的接口而定，当客户端应用程序对某列执行 DESCRIBE 时，使用字符长度语义可能会影响返回的值。例如，当某个嵌入式 SQL 客户端对某列执行通过字节长度语义声明的 DESCRIBE 时，所返回的长度为指定的字节长度。因此，CHAR(10) 列将被描述为长度为 10 个字节的类型 DT_FIXCHAR。不过，当嵌入式 SQL 客户端对某列执行通过字符长度语义声明的 DESCRIBE 时，所返回的长度将是在客户端的 CHAR 字符集中指定的最大字节长度。例如，对于将 UTF-8 用作 CHAR 字符集的嵌入式 SQL 客户端，CHAR(10 CHAR) 列将被描述为长度是 40 个字节（10 个字符乘以每个字符的最大字节数四）的类型 DT_FIXCHAR。

另请参见

- “VARCHAR 数据类型”一节第 81 页
- “LONG VARCHAR 数据类型”一节第 78 页
- “NCHAR 数据类型”一节第 78 页

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。字符长度语义是服务商扩充。

LONG NVARCHAR 数据类型

LONG NVARCHAR 数据类型存储任意长度的 Unicode 字符数据。

语法

LONG NVARCHAR

注释

最大大小为 2 GB。

字符以 UTF-8 格式存储。每个字符都需要一到四个字节。可以 LONG NVARCHAR 形式存储的最大字符数可能会超过 5 亿个，也可能会超过 20 亿个，具体要视存储的字符长度而定。

当嵌入式 SQL 客户端对 LONG NVARCHAR 列执行 DESCRIBE 时，返回的数据类型将为 DT_LONGVARCHAR 或 DT_LONGNVARCHAR，具体视是否调用了 db_change_nchar_charset 函数而定。请参见“db_change_nchar_charset 函数”一节 《SQL Anywhere 服务器 - 编程》。

如果是 ODBC，LONG NVARCHAR 表达式将被描述为 SQL_WLONGVARCHAR。

另请参见

- “NCHAR 数据类型”一节第 78 页
- “NVARCHAR 数据类型”一节第 80 页
- “LONG VARCHAR 数据类型”一节第 78 页

标准和兼容性

- **SQL/2003** 服务商扩充。

LONG VARCHAR 数据类型

LONG VARCHAR 数据类型存储任意长度的字符数据。

语法

LONG VARCHAR

注释

最大大小为 2 GB。

可以将多字节字符存储为 LONG VARCHAR 类型，但其长度是以字节而不是以字符为单位。

另请参见

- “CHAR 数据类型”一节第 76 页
- “VARCHAR 数据类型”一节第 81 页
- “LONG NVARCHAR 数据类型”一节第 77 页

标准和兼容性

- **SQL/2003** 服务商扩充。

NCHAR 数据类型

NCHAR 数据类型存储 Unicode 字符数据，最多可存储 8191 个字符。

语法

NCHAR [(*max-length*)]

参数

- **max-length** 字符串的最大长度（以字符为单位）。该长度必须在 1 到 8191 范围内。如果未指定长度，则值为 1。

注释

字符使用 UTF-8 编码存储。所需的最大存储字节数为 *max-length* 的四倍，尽管所需的实际存储字节数通常要少得多。

也可以将 NCHAR 指定为 NATIONAL CHAR 或 NATIONAL CHARACTER。无论使用哪种语法，都会将数据类型描述为 NCHAR。

当嵌入式 SQL 客户端对 NCHAR 列执行 DESCRIBE 时，返回的数据类型将为 DT_FIXCHAR 或 DT_NFIXCHAR，具体视是否调用了 db_change_nchar_charset 函数而定。请参见“db_change_nchar_charset 函数”一节 《SQL Anywhere 服务器 - 编程》。

此外，当嵌入式 SQL 客户端对 NCHAR 列执行 DESCRIBE 时，返回的长度将是客户端的 NCHAR 字符集中的最大字节长度。例如，对于将西欧字符集 cp1252 用作 NCHAR 字符集的嵌入式 SQL 客户端，NCHAR(10) 列将被描述为长度是 10（10 个字符乘以每个字符的最大字节数一）的类型 DT_NFIXCHAR。对于使用日文字符集 cp932 的嵌入式 SQL 客户端，同一列将被描述为长度为 20（10 个字符乘以每个字符的最大字节数二）的类型 DT_NFIXCHAR。

虽然 NCHAR 和 NVARCHAR 属于不同的类型，但它们在语义上是等同的。在 SQL Anywhere 中，NCHAR 是一种可变长度类型。在其它关系数据库管理系统中，NCHAR 是固定长度类型，将使用空白填充数据来使其达到 *max-length* 个存储字节。SQL Anywhere 不会使用空白填充所存储的字符数据。

对于 ODBC，视是否使用 odbc_distingish_char_and_varchar 选项而定，将 NCHAR 描述为 SQL_WCHAR 或 SQL_WVARCHAR。请参见“odbc_distinguish_char_and_varchar 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

另请参见

- “CHAR 数据类型”一节第 76 页
- “NVARCHAR 数据类型”一节第 80 页
- “LONG NVARCHAR 数据类型”一节第 77 页

标准和兼容性

- **SQL/2003** 服务商扩充。

NTEXT 数据类型

NTEXT 数据类型存储任意长度的 Unicode 字符数据。

语法

NTEXT

注释

NTEXT 是一个域，实现为 LONG NVARCHAR。

另请参见

- “LONG NVARCHAR 数据类型”一节第 77 页
- “TEXT 数据类型”一节第 81 页

标准和兼容性

- **SQL/2003** 服务商扩充。

NVARCHAR 数据类型

NVARCHAR 数据类型存储 Unicode 字符数据，最多可存储 8191 个字符。

语法

NVARCHAR [**(** *max-length* **)**]

参数

- **max-length** 字符串的最大长度（以字符为单位）。该长度必须在 1 到 8191 范围内。如果未指定长度，则值为 1。

注释

字符以 UTF-8 编码存储。所需的最大存储字节数为 *max-length* 的四倍，尽管所需的实际存储字节数通常要少得多。

也可以将 NVARCHAR 指定为 NCHAR VARYING、NATIONAL CHAR VARYING 或 NATIONAL CHARACTER VARYING。无论使用哪种语法，都会将数据类型描述为 NVARCHAR。

当嵌入式 SQL 客户端对 NVARCHAR 列执行 DESCRIBE 时，返回的数据类型将为 DT_VARCHAR 或 DT_NVARCHAR，具体视是否调用了 db_change_nchar_charset 函数而定。请参见“db_change_nchar_charset 函数”一节 《SQL Anywhere 服务器 - 编程》。

此外，当嵌入式 SQL 客户端对 NVARCHAR 列执行 DESCRIBE 时，返回的长度将是客户端的 NCHAR 字符集中的最大字节长度。例如，对于将西欧字符集 cp1252 用作 NCHAR 字符集的嵌入式 SQL 客户端，NVARCHAR(10) 列将被描述为长度是 10（10 个字符乘以每个字符的最大字节数一）的类型 DT_NVARCHAR。对于使用日文字符集 cp932 的嵌入式 SQL 客户端，同一列将被描述为长度为 20（10 个字符乘以每个字符的最大字节数二）的类型 DT_NFIXCHAR。

对于 ODBC，视是否使用 odbc_distingish_char_and_varchar 选项而定，将 NVARCHAR 描述为 SQL_WCHAR 或 SQL_WVARCHAR。请参见“odbc_distinguish_char_and_varchar 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

另请参见

- “NCHAR 数据类型”一节第 78 页
- “LONG NVARCHAR 数据类型”一节第 77 页
- “VARCHAR 数据类型”一节第 81 页

标准和兼容性

- **SQL/2003** 服务商扩充。

TEXT 数据类型

TEXT 数据类型存储任意长度的字符数据。

语法

TEXT

注释

TEXT 是一个域，实现为 LONG VARCHAR。

另请参见

- “LONG VARCHAR 数据类型”一节第 78 页
- “NTEXT 数据类型”一节第 79 页

标准和兼容性

- **SQL/2003** 服务商扩充。

UNIQUEIDENTIFIERSTR 数据类型

UNIQUEIDENTIFIERSTR 数据类型是一个域，实现为 CHAR(36)。

语法

UNIQUEIDENTIFIERSTR

注释

映射 Microsoft SQL Server uniqueidentifier 列时用于远程数据访问。

另请参见

- “数据类型转换：Microsoft SQL Server”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “STRTOUUID 函数 [String]”一节第 304 页

标准和兼容性

- **SQL/2003** 服务商扩充。

VARCHAR 数据类型

VARCHAR 数据类型存储字符数据，最多可存储 32767 个字节。

语法

VARCHAR [**(** *max-length* [**CHAR** | **CHARACTER**] **)**]

参数

- **max-length** 字符串的最大长度。如果使用字节长度语义（不将 CHAR 或 CHARACTER 指定为长度的一部分），则长度将以字节为单位，并且长度必须在 1 到 32767 的范围内。如果未指定长度，则值为 1。

  如果使用字符长度语义（将 CHAR 或 CHARACTER 指定为长度的一部分），则长度将以字符为单位，并且必须指定 *max-length*。使用字符长度语义时，长度与数据库编码中字符最大长度的乘积不得超过 32767 个字节。下表显示所支持字符集类型的最大长度：

字符集	VARCHAR 的最大长度
单字节字符集	32767 个字符
双字节字符集	16383 个字符
UTF-8	8191 个字符

注释

可以将多字节字符存储为 VARCHAR 类型，但声明的长度指的是字节长度而不是字符长度。

也可以将 VARCHAR 指定为 CHAR VARYING 或 CHARACTER VARYING。无论使用哪种语法，都会将数据类型描述为 VARCHAR。

视所使用的接口而定，当客户端应用程序对某列执行 DESCRIBE 时，使用字符长度语义可能会影响返回的值。例如，当某个嵌入式 SQL 客户端应用程序对某列执行通过字节长度语义声明的 DESCRIBE 时，所返回的长度为指定的字节长度。因此，VARCHAR(10) 列将被描述为长度为 10 个字节的类型 DT_VARCHAR。不过，当嵌入式 SQL 客户端应用程序对某列执行通过字符长度语义声明的 DESCRIBE 时，所返回的长度将是在客户端的 CHAR 字符集中指定的最大字节长度。例如，对于将 UTF-8 用作 CHAR 字符集的客户端，VARCHAR(10 CHAR) 列将被描述为长度是 40 个字节的类型 DT_VARCHAR（10 个字符乘以每个字符的最大字节数四）。

另请参见

- “CHAR 数据类型”一节第 76 页
- “LONG VARCHAR 数据类型”一节第 78 页
- “NVARCHAR 数据类型”一节第 80 页

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。字符长度语义是服务商扩充。

XML 数据类型

XML 数据类型存储任意长度的字符数据，也可以用于存储 XML 文档。

语法

XML

注释

最大大小为 2 GB。

当从关系数据中生成元素内容时，XML 类型数据没有被引起来。

您可以在 XML 数据类型与任何其它能够与字符串相互转换的数据类型之间进行转换。请注意，在将字符串转换为 XML 时，不会检查其格式是否正确。

有关在生成 XML 元素时使用 XML 数据类型的信息，请参见“在关系数据库中存储 XML 文档”一节 《SQL Anywhere 服务器 - SQL 的用法》。

嵌入式 SQL 客户端应用程序对 XML 列执行 DESCRIBE 时，会将 XML 列描述为 LONG VARCHAR。

另请参见

- “在数据库中使用 XML” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。

数字数据类型

数字数据类型用于存储数字数据。

NUMERIC 和 DECIMAL 数据类型以及各种 INTEGER 数据类型有时被称为**精确** 数字数据类型，与之相对的是**近似** 数字数据类型 FLOAT、DOUBLE 和 REAL。

精确数字数据类型是指可以指定精度值和小数位数值的数据类型，而近似数字数据类型是指以预定义方式存储的数据类型。*只有精确数字数据能确保在算术运算后精确到指定的最小有效数字。*

不允许小于 1 的数据类型长度和精度。

兼容性

只有小数位数为 0 的 NUMERIC 数据类型可以用于 Transact-SQL identity 列。

使用 NUMERIC 和 DECIMAL 数据类型的缺省精度和小数位数设置时要小心，因为这些设置在其它数据库解决方案中可能是不同的。在 SQL Anywhere 中，缺省精度为 30，而缺省小数位数为 6。

NUMERIC 和 DECIMAL 数据类型应当避免使用缺省精度和小数位数设置，因为在 SQL Anywhere 和 Adaptive Server Enterprise 中这些设置是不同的。在 SQL Anywhere 中，缺省精度为 30，而缺省小数位数为 6。在 Adaptive Server Enterprise 中，缺省精度为 18，而缺省小数位数为 0。

FLOAT (*p*) 数据类型是 REAL 或 DOUBLE 的同义词，具体取决于 *p* 的值。对于 SQL Anywhere，分界值与平台相关，但在所有平台上分界值都大于 15。

有关通过设置数据库选项更改缺省值的信息，请参见“precision 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》和“scale 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

BIGINT 数据类型

BIGINT 数据类型用于存储 BIGINT（需要 8 个存储字节的整数）。

语法

[UNSIGNED] BIGINT

注释

BIGINT 数据类型是一种精确数字数据类型：其精度在算术运算后保持不变。

一个 BIGINT 值需要 8 个存储字节。

有符号 BIGINT 值的范围是 -2^{63} 到 $2^{63} - 1$，即 -9223372036854775808 到 9223372036854775807。

无符号 BIGINT 值的范围是 0 到 $2^{64} - 1$，即 0 到 18446744073709551615。

缺省情况下，数据类型是带符号的。

在将字符串转换为 BIGINT 时，前导空格和尾随空格会被删除。如果前导字符是 '+'，会被忽略。如果前导字符是 '-'，则后面的位会被解释为负数。前导的 '0' 字符将跳过，其余字符将转换为整数值。如果值超出目标数据类型的有效范围、字符串包含非法字符、或者字符串无法解码为整数值，则会返回错误。

另请参见

- “BIT 数据类型”一节第 85 页
- “INTEGER 数据类型”一节第 88 页
- “SMALLINT 数据类型”一节第 90 页
- “TINYINT 数据类型”一节第 91 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 服务商扩充。

BIT 数据类型

BIT 数据类型用于存储位（0 或 1）。

语法

BIT

注释

BIT 是一种可以存储值 0 或 1 的整数类型。

缺省情况下，BIT 数据类型不允许 NULL。

在将字符串转换为 BIT 时，前导空格和尾随空格会被删除。如果前导字符是 '+'，会被忽略。如果前导字符是 '-'，则后面的位会被解释为负数。前导的 '0' 字符将跳过，其余字符将转换为整数值。如果值超出目标数据类型的有效范围、字符串包含非法字符、或者字符串无法解码为整数值，则会返回错误。

另请参见

- “BIGINT 数据类型”一节第 84 页
- “INTEGER 数据类型”一节第 88 页
- “SMALLINT 数据类型”一节第 90 页
- “TINYINT 数据类型”一节第 91 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 服务商扩充。

DECIMAL 数据类型

DECIMAL 数据类型是总位数为 *precision* 且小数点后位数为 *scale* 的小数。

语法

DECIMAL [**(** *precision* [, *scale*] **)**]

参数

- **precision** 一个在 1 到 127 范围内（含 1 和 127）的整数表达式，指定表达式中的位数。缺省设置为 30。
- **scale** 一个在 0 到 127 范围内（含 1 和 127）的整数表达式，指定小数点后的位数。小数位数值应始终小于或等于精度值。缺省设置为 6。

  可以通过设置数据库选项更改缺省值。有关信息，请参见“precision 选项 [数据库]”一节《SQL Anywhere 服务器 - 数据库管理》和“scale 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》 。

注释

DECIMAL 数据类型是精确数字数据类型，其精度在算术运算后保留到最小有效位。

存储小数所需的空间可通过如下方式计算

```
2 + int( (before + 1)/2 ) + int( (after + 1)/2 )
```

函数 int 用于将其参数取整，而 before 和 after 分别是小数点之前和之后的有效位数。存储基于的是所存储的值，而不是列中允许的最大精度和小数位数。

也可以将 DECIMAL 指定为 DEC。无论使用哪种语法，都会将数据类型描述为 DECIMAL。

DECIMAL 在语义上等同于 NUMERIC。

另请参见

- “FLOAT 数据类型”一节第 87 页
- “REAL 数据类型”一节第 90 页
- “DOUBLE 数据类型”一节第 86 页
- “NUMERIC 数据类型”一节第 89 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。

DOUBLE 数据类型

DOUBLE 数据类型用于存储双精度浮点数。

语法

DOUBLE [**PRECISION**]

注释

DOUBLE 数据类型保存双精度浮点数。它是一种近似数字数据类型，在算术运算后容易产生舍入误差。DOUBLE 值的近似特性意味着在比较 DOUBLE 值时通常应该避免使用等式的查询。

DOUBLE 值需要 8 个存储字节。

取值范围是 -1.79769313486231e+308 到 1.79769313486231e+308，最趋近于零的数为 2.22507385850721e-308。DOUBLE 类型的值精确到 15 位有效数字，而超过第十五位就可能会产生舍入误差。

另请参见

- “FLOAT 数据类型”一节第 87 页
- “REAL 数据类型”一节第 90 页
- “DECIMAL 数据类型”一节第 85 页
- “NUMERIC 数据类型”一节第 89 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页
- “数字集之间的转换”一节第 112 页

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。

FLOAT 数据类型

FLOAT 数据类型用于存储单精度浮点数或双精度浮点数。

语法

FLOAT [**(** *precision* **)**]

参数

- **precision** 指定尾数位数的整数表达式。尾数即对数中的小数部分。例如，对数 5.63428 的尾数即为 0.63428。IEEE 标准 754 浮点精度如下：

提供的精度值	数值精度	对应 SQL 数据类型	占用内存
1-24	7 位小数	REAL	4 个字节
25-53	15 位小数	DOUBLE	8 个字节

注释

使用 FLOAT (*precision*) 数据类型创建列时，保证所有平台上的列至少按指定的最小精度保存值。相比之下，REAL 和 DOUBLE 不能保证与平台无关的最小精度。

如果未提供 *precision*，FLOAT 数据类型将是单精度浮点数（等同于 REAL 数据类型），需要 4 个存储字节。

如果提供了 *precision*，则 FLOAT 数据类型将是单精度或双精度型，具体取决于所指定的精度值。REAL 和 DOUBLE 之间的分界值与平台相关。单精度 FLOAT 值需要 4 个存储字节，而双精度 FLOAT 值则需要 8 个存储字节。

FLOAT 数据类型是一种近似数字数据类型。它在算术运算后容易产生舍入误差。FLOAT 值的近似特性意味着在比较 FLOAT 值时通常应该避免使用等式的查询。

另请参见

- “DOUBLE 数据类型”一节第 86 页
- “REAL 数据类型”一节第 90 页
- “DECIMAL 数据类型”一节第 85 页
- “NUMERIC 数据类型”一节第 89 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。

INTEGER 数据类型

INTEGER 数据类型用于存储需要 4 个存储字节的整数。

语法

[UNSIGNED] INTEGER

注释

INTEGER 数据类型是精确数字数据类型，其精度在算术运算后不变。

如果指定 UNSIGNED，永远无法将整数指定为负数。缺省情况下，数据类型是带符号的。

有符号整数的范围是 -2^{31} 到 $2^{31} - 1$，即 -2147483648 到 2147483647。

无符号整数的范围是 0 到 $2^{32} - 1$，或 0 到 4294967295。

也可以将 INTEGER 指定为 INT。无论使用哪种语法，都会将数据类型描述为 INTEGER。

在将字符串转换为 INTEGER 时，前导空格和尾随空格会被删除。如果前导字符是 '+'，会被忽略。如果前导字符是 '-'，则后面的位会被解释为负数。前导的 '0' 字符将跳过，其余字符将转换为整数值。如果值超出目标数据类型的有效范围、字符串包含非法字符、或者字符串无法解码为整数值，则会返回错误。

另请参见

- “BIGINT 数据类型”一节第 84 页
- “BIT 数据类型”一节第 85 页
- “SMALLINT 数据类型”一节第 90 页
- “TINYINT 数据类型”一节第 91 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。UNSIGNED 关键字是服务商扩充。

NUMERIC 数据类型

NUMERIC 数据类型用于存储总位数为 *precision* 且小数点后位数为 *scale* 的小数。

语法

NUMERIC [(*precision* [, *scale*])]

参数

- **precision** 一个在 1 到 127 范围内（含 1 和 127）的整数表达式，指定表达式中的位数。缺省设置为 30。
- **scale** 一个在 0 到 127 范围内（含 1 和 127）的整数表达式，指定小数点后的位数。小数位数值应始终小于或等于精度值。缺省设置为 6。

  可以通过设置数据库选项更改缺省值。有关信息，请参见“precision 选项 [数据库]”一节《SQL Anywhere 服务器 - 数据库管理》和“scale 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

注释

NUMERIC 数据类型是一种精确数字数据类型，其精度在算术运算后保留到最小有效位。

存储小数所需的字节数可通过如下方式计算

```
2 + INT( (BEFORE+1)/2 ) + INT( (AFTER+1)/2 )
```

INT 函数用于将其参数取整，而 BEFORE 和 AFTER 分别是小数点之前和之后的有效数字位数。存储基于的是所存储的值，而不是列中允许的最大精度和小数位数。

NUMERIC 在语义上等同于 DECIMAL。

另请参见

- “FLOAT 数据类型”一节第 87 页
- “REAL 数据类型”一节第 90 页
- “DOUBLE 数据类型”一节第 86 页
- “DECIMAL 数据类型”一节第 85 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页
- “数字集之间的转换”一节第 112 页

标准和兼容性

- **SQL/2003** 如果 scale 选项设置为零，则与 SQL/2003 兼容。

REAL 数据类型

REAL 数据类型用于存储单精度浮点数（存储在 4 个字节中）。

语法

REAL

注释

REAL 数据类型是一种近似数字数据类型，在算术运算后容易产生舍入误差。

取值范围是 -3.402823e+38 到 3.402823e+38，与零的误差可低至 1.175495e-38。保存为 REAL 类型的值精确到 7 个有效数字，但超过第六位可能会产生舍入误差。

REAL 值的近似特性意味着在比较 REAL 值时通常应该避免使用相等运算符的查询。

另请参见

- “DOUBLE 数据类型”一节第 86 页
- “FLOAT 数据类型”一节第 87 页
- “DECIMAL 数据类型”一节第 85 页
- “NUMERIC 数据类型”一节第 89 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。

SMALLINT 数据类型

SMALLINT 数据类型用于存储需要 2 个存储字节的整数。

语法

[UNSIGNED] SMALLINT

注释

SMALLINT 数据类型是一种精确数字数据类型，其精度在算术运算后保持不变。它需要 2 个存储字节。

有符号 SMALLINT 值的范围是 -2^{15} 到 $2^{15} - 1$，即 -32768 到 32767。

无符号 SMALLINT 的取值范围是 0 到 $2^{16} - 1$，即 0 到 65535。

在将字符串转换为 SMALLINT 时，前导空格和尾随空格会被删除。如果前导字符是 '+'，会被忽略。如果前导字符是 '-'，则后面的位会被解释为负数。前导的 '0' 字符将跳过，其余字符将转换为整数值。如果值超出目标数据类型的有效范围、字符串包含非法字符、或者字符串无法解码为整数值，则会返回错误。

另请参见

- “BIGINT 数据类型”一节第 84 页
- “BIT 数据类型”一节第 85 页
- “INTEGER 数据类型”一节第 88 页
- “TINYINT 数据类型”一节第 91 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 与 SQL/2003 兼容。UNSIGNED 关键字是服务商扩充。

TINYINT 数据类型

TINYINT 数据类型用于存储需要 1 个存储字节的无符号整数。

语法

[UNSIGNED] TINYINT

注释

TINYINT 数据类型是一种精确数字数据类型；其精度在算术运算后不变。

可以将 TINYINT 显式指定为 UNSIGNED，但是该类型始终是无符号的，因此 UNSIGNED 修饰符无效。

TINYINT 的值的范围是 0 到 2^8 - 1，即 0 到 255。

在嵌入式 SQL 中，不应将 TINYINT 列读取到定义为 char 或无符号 char 的变量中，因为这样做的结果是系统会尝试将列的值转换为字符串，然后将第一个字节指派给程序中的变量。相反，应该将 TINYINT 列读入 2 个字节或 4 个字节的整数列。此外，要从使用 C 语言编写的应用程序向数据库发送 TINYINT 值，C 变量的类型应该为整数。

在将字符串转换为 TINYINT 时，前导空格和尾随空格会被删除。如果前导字符是 '+'，会被忽略。如果前导字符是 '-'，则后面的位会被解释为负数。前导的 '0' 字符将跳过，其余字符将转换为整数值。如果值超出目标数据类型的有效范围、字符串包含非法字符、或者字符串无法解码为整数值，则会返回错误。

另请参见

- “BIGINT 数据类型”一节第 84 页
- “BIT 数据类型”一节第 85 页
- “INTEGER 数据类型”一节第 88 页
- “SMALLINT 数据类型”一节第 90 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 服务商扩充。

Money 数据类型

货币数据类型用于存储货币数据。

MONEY 数据类型

MONEY 数据类型存储货币数据。

语法

MONEY

注释

MONEY 实现为域，作为 NUMERIC(19,4) 类型。

另请参见

- “SMALLMONEY 数据类型”一节第 92 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 服务商扩充。

SMALLMONEY 数据类型

SMALLMONEY 数据类型用于存储不足一百万货币单位的货币数据。

语法

SMALLMONEY

注释

SMALLMONEY 实现为域，作为 NUMERIC(10,4) 类型。

另请参见

- “MONEY 数据类型”一节第 92 页
- “数字函数”一节第 123 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 服务商扩充。

位数组数据类型

位数组用于存储位数据（0 和 1）。

位数组 数据类型用于存储位数组。SQL Anywhere 支持的位数组数据类型包括 VARBIT 和 LONG VARBIT。

LONG VARBIT 数据类型

LONG VARBIT 数据类型用于存储任意长度的位数组。

语法

LONG VARBIT

注释

用于存储任意长度的位（1 和 0）数组或长度超过 32767 位的位数组。

也可以将 LONG VARBIT 指定为 LONG BIT VARYING。无论使用哪种语法，都会将数据类型描述为 LONG VARBIT。

另请参见

- “BIT 数据类型”一节第 85 页
- “VARBIT 数据类型”一节第 93 页
- “转换位数组”一节第 110 页
- “位数组函数”一节第 119 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 服务商扩充。

VARBIT 数据类型

VARBIT 数据类型用于存储长度小于 32767 位的位数组。

语法

VARBIT [**(***max-length* **)**]

参数

- **max-length** 位数组的最大长度（以位为单位）。该长度必须在 1 到 32767 范围内。如果未指定长度，则值为 1。

注释

也可以将 VARBIT 指定为 BIT VARYING。无论使用哪种语法，都会将数据类型描述为 VARBIT。

另请参见

- “BIT 数据类型”一节第 85 页
- “LONG VARBIT 数据类型”一节第 93 页
- “转换位数组”一节第 110 页
- “位数组函数”一节第 119 页
- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 服务商扩充。

日期和时间数据类型

以下列表提供了日期处理方式的概览：

- SQL Anywhere 总是对任何合法的日期算术和逻辑运算返回正确的值，不论计算出的值是否跨越不同的世纪。
- SQL Anywhere 内部存储的日期始终显式地包括年份值的世纪部分。
- SQL Anywhere 的操作不受任何返回值（包括当前日期）的影响。
- 日期值总是能够以完整的世纪格式输出。

日期的存储方式

包含年份值的日期以下面的数据类型之一在内部使用，并以该类型存储在 SQL Anywhere 数据库中：

数据类型	包含	存储为	可接受值的范围
DATE	日历日期（年、月、日）	4 个字节	0001-01-01 到 9999-12-31
TIMESTAMP	时间戳（年、月、日、小时、分钟、秒和精确到 6 个小数位百万分之一秒）	8 个字节	0001-01-01 到 9999-12-31（TIMESTAMP 时间部分早于 1600-02-28 23:59:59 和晚于 7911-01-01 00:00:00 的精度将被删除）

有关 SQL Anywhere 日期和时间数据类型的详细信息，请参见“日期和时间数据类型”一节 第 95 页。

向数据库发送日期和时间

可以使用以下方式之一向数据库发送日期和时间：

- 使用任何接口，以字符串形式
- 使用 ODBC，以 TIMESTAMP 结构形式
- 使用嵌入式 SQL，以 SQLDATETIME 结构形式

将时间以字符串（对于 TIME 数据类型）或字符串的一部分（对于 TIMESTAMP 或 DATE 数据类型）形式发送到数据库时，小时、分钟和秒必须按照 *hh*:*mm*:*ss*.*sss* 格式以冒号分隔，但可以出现在字符串中的任何位置。以下是用于指定时间的有效且明确的字符串：

```
21:35 -- 24 hour clock if no am or pm specified 10:00pm -- pm specified, so
interpreted as 12 hour clock 10:00 -- 10:00am in the absence of pm
10:23:32.234 -- seconds and fractions of a second included
```

将日期以字符串形式发送到数据库时，其到 DATE 或 TIMESTAMP 数据类型的转换是自动进行的。可以用以下两种方式之一提供字符串：

- 以 *yyyy/mm/dd* 或 *yyyy-mm-dd* 格式的字符串形式，数据库对该格式有明确解释。
- 以按照 date_order 数据库选项解释的字符串形式。请参见“date_order 选项 [数据库]”一节《SQL Anywhere 服务器 - 数据库管理》。

Transact-SQL 字符串到日期/时间转换

将字符串转换为日期和时间数据类型。

如果将只包含时间值（不包含日期）的字符串转换为日期/时间数据类型，SQL Anywhere 会使用当前日期。

如果时间的小数部分小于 3 位，则无论该值的前面是句点还是冒号，SQL Anywhere 都会以相同的方式对该值进行解释：一位表示十分位，两位表示百分位，三位表示千分位。

示例

无论分隔符为何，SQL Anywhere 都会以相同方式转换毫秒值。

```
12:34:56.7 to 12:34:56.700 12:34:56:7 to 12:34:56.700 12.34.56.78 to
12:34:56.780 12.34.56:78 to 12:34:56.780 12:34:56.789 to 12:34:56.789
12:34:56:789 to 12:34:56.789
```

从数据库中检索日期和时间

可以使用以下方式之一从数据库中检索日期和时间：

- 使用任何接口，以字符串形式
- 使用 ODBC，以 TIMESTAMP 结构形式
- 使用嵌入式 SQL，以 SQLDATETIME 结构形式

以字符串形式检索日期或时间时，按照数据库选项 date_format、time_format 和 timestamp_format 所指定的格式进行检索。有关这些选项的说明，请参见“SET OPTION 语句”一节第 702 页。

有关用于处理日期和时间的函数的信息，请参见“日期和时间函数”一节第 120 页。允许对日期使用以下算术运算符：

- **timestamp + integer** 向日期或时间戳中添加指定的天数。
- **timestamp - integer** 从日期或时间戳中减去指定的天数。
- **date - date** 计算两个日期或时间戳之间的天数。
- **date + time** 创建将给定日期和时间组合在一起的时间戳。

闰年

SQL Anywhere 使用全球通用的算法来确定哪些年是闰年。使用这种算法，当年份可以被四整除时即视为闰年，但年份为世纪日期时（如 1900 年）除外，此时年份只有在可以被 400 整除时才是闰年。

SQL Anywhere 会正确地处理所有闰年。例如，以下 SQL 语句产生的返回值为 "Tuesday"：

```
SELECT DAYNAME('2000-02-29');
```

SQL Anywhere 接受 2000 年 —闰年— 2 月 29 日作为日期，并用这个日期确定星期几。

然而，以下语句会遭到 SQL Anywhere 拒绝：

```
SELECT DAYNAME('2001-02-29');
```

此语句会导致错误（无法将 '2001-02-29' 转换为日期），因为 2001 年不存在 2 月 29 日。

比较日期和时间

缺省情况下，存储为 DATE 类型的值没有任何小时或分钟值，因此比较日期非常简单。

DATE 数据类型也可以包含时间，这会使日期的比较变得复杂。如果在将日期输入数据库时没有指定时间，缺省时间将是 0:00 或 12:00am（午夜）。带有此选项设置的任何日期比较都会比较时间和日期。数据库日期值 1999-05-23 10:00 与常量 1999-05-23 并不相等。可以使用 DATEFORMAT 函数或某个其它日期函数来比较日期和时间字段的各个部分。例如，

```
DATEFORMAT(invoice_date,'yyyy/mm/dd') = '1999/05/23';
```

如果数据库列只需要日期，客户端应用程序应确保在将数据输入到数据库中时不指定时间。这样，只包含日期字符串的比较才会按预期方式进行。

如果要将某个日期*作为字符串*与某个字符串进行比较，则必须在比较之前使用 DATEFORMAT 函数或 CAST 函数将日期转换为字符串。

使用明确的日期和时间

yyyy/mm/dd 或 *yyyy-mm-dd* 格式的日期总是会被明确地识别为日期，而与 date_order 设置无关。可以使用其它字符代替 '/' 或 '-' 作为分隔符；例如 '?'、空格字符或 ','。在任何不同的用户可能会采用不同 date_order 设置的环境中，都应使用此格式。例如，在存储过程中，使用明确的日期格式可以防止按照用户的 date_order 设置错误地解释日期。

此外，*hh:mm:ss:ssss* 格式的字符串将明确地解释为时间。

对于日期和时间的组合，任何明确的日期和任何明确的时间都会产生明确的日期时间值。同样，*yyyy-mm-dd hh.mm.ss.sss* 格式也是明确的日期时间值。句点只能在与日期组合的时间中使用。

在其它环境中，可以使用更灵活的日期格式。SQL Anywhere 可以将多种字符串解释为日期。具体的解释取决于数据库选项 date_order 的设置。date_order 数据库选项的值可以是 *MDY*、*YMD* 或 *DMY*（请参见“SET OPTION 语句”一节第 702 页）。例如，以下语句将 date_order 选项设置为 *DMY*：

```
SET OPTION date_order = 'DMY' ;
```

缺省的 date_order 设置为 YMD。每当建立连接时，ODBC 驱动程序就会将 date_order 选项设置为 YMD。仍然可以使用 SET TEMPORARY OPTION 语句更改该值。

数据库选项 date_order 决定数据库将字符串 10/11/12 解释为 2010 年 11 月 12 日、2012 年 10 月 11 日还是 2012 年 11 月 10 日。日期字符串的年、月、日应该用某个字符（/、- 或空格）分隔，并应按 date_order 选项指定的顺序显示。

年份可以 2 位或 4 位形式提供。nearest_century 选项的值影响对 2 位年份的解释：对于小于 nearest_century 的值，将为其加上 2000；对于所有其它值，将为其加上 1900。此选项的缺省值为 50。因此，缺省情况下会将 50 解释为 1950，而将 49 解释为 2049。

月份可以是月份的名称或编号。小时和分钟以冒号分隔，但可以出现在字符串中的任何位置。

注意

- 建议始终使用 4 位格式指定年份。
- 只要 date_order 的设置正确，以下字符串便都是有效日期：

```
99-05-23 21:35 99/5/23 1999/05/23 May 23 1999 23-May-1999 Tuesday May 23,
1999 10:00pm
```

- 如果字符串只包含部分日期规范，将使用缺省值填写日期。使用以下缺省值：
  - **year**　今年
  - **month**　没有缺省值
  - **day**　1（对月份字段很有用；例如，1999 年 5 月解释为日期 1999-05-01 00:00）
  - **小时、分钟、秒、小数**　0

DATE 数据类型

DATE 数据类型用于存储日历日期，如年、月和日。

语法

DATE

注释

年份可以从 0001 年到 9999 年。SQL Anywhere 中的最小日期是 0001-01-01 00:00:00。

由于历史原因，DATE 列还可以包含小时和分钟。对于任何包含小时和分钟的数据，建议使用 TIMESTAMP 数据类型。

应用程序检索 DATE 值所使用的格式由 date_format 设置控制。例如，表示 2010 年 7 月 19 日的日期值可能会以 `2010/07/19` 或 `Jul 19, 2010` 的形式返回给应用程序。

数据库服务器将字符串解释为日期的方式由 date_order 选项控制。例如，视 date_order 的设置而定，应用程序为 DATE 值提供的值 `02/05/2002` 在数据库中可能被解释为 5 月 2 日或 2 月 5 日。

DATE 值需要 4 个存储字节。

另请参见

- “date_format 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “date_order 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “DATETIME 数据类型”一节第 99 页
- “SMALLDATETIME 数据类型”一节第 99 页
- “TIMESTAMP 数据类型”一节第 100 页
- “日期和时间函数”一节第 120 页

标准和兼容性

- **SQL/2003** 服务商扩充。

DATETIME 数据类型

DATETIME 数据类型是一个域，实现为 TIMESTAMP，用于存储日期和时间信息。

语法

DATETIME

另请参见

- “DATE 数据类型”一节第 98 页
- “SMALLDATETIME 数据类型”一节第 99 页
- “TIMESTAMP 数据类型”一节第 100 页
- “日期和时间函数”一节第 120 页

标准和兼容性

- **SQL/2003** 服务商扩充。

SMALLDATETIME 数据类型

SMALLDATETIME 数据类型是一个域，实现为 TIMESTAMP，用于存储日期和时间信息。

语法

SMALLDATETIME

另请参见

- “DATE 数据类型”一节第 98 页
- “DATETIME 数据类型”一节第 99 页
- “TIMESTAMP 数据类型”一节第 100 页
- “日期和时间函数”一节第 120 页

标准和兼容性

- **SQL/2003** 服务商扩充。

TIME 数据类型

TIME 数据类型用于存储某天的时间，包括小时、分钟、秒和秒的小数。

语法

TIME

注释

小数存储到 6 个小数位。TIME 值需要 8 个存储字节。（ODBC 标准将 TIME 数据类型限制为精确到秒。由于这个原因，在依赖高于秒的精度的 WHERE 子句比较中，不应使用 TIME 数据类型。）

另请参见

- “TIMESTAMP 数据类型”一节第 100 页
- “日期和时间函数”一节第 120 页

标准和兼容性

- **SQL/2003** 服务商扩充。

TIMESTAMP 数据类型

TIMESTAMP 数据类型用于存储时间点，包括年、月、日、小时、分钟、秒和秒的小数。

语法

TIMESTAMP

注释

小数存储到 6 个小数位。TIMESTAMP 值需要 8 个存储字节。

尽管 TIMESTAMP 数据类型的可接受日期范围与 DATE 类型相同（从 0001 年到 9999 年），但 TIMESTAMP 日期类型的有用范围是从 1600-02-28 23:59:59 到 7911-01-01 00:00:00。早于或晚于此范围的 TIMESTAMP 的时间部分可能不完整。

另请参见

- “TIME 数据类型”一节第 100 页
- “日期和时间函数”一节第 120 页

标准和兼容性

- **SQL/2003** 服务商扩充。

二进制数据类型

二进制数据类型用于存储二进制数据，包括图像和数据库不进行解释的其它类型信息。

BINARY 数据类型

BINARY 数据类型用于存储具有指定最大长度（以字节为单位）的二进制数据。

语法

BINARY [(*max-length*)]

参数

- **max-length** 值的最大长度（以字节为单位）。该长度必须在 1 到 32767 范围内。如果未指定长度，则值为 1。

注释

在比较过程中，将会精确地逐个字节地对 BINARY 值进行比较。与之不同的是，CHAR 数据类型的值是使用数据库的归类序列进行比较。如果一个二进制字符串是另一个二进制字符串的前缀，则认为较短的字符串小于较长的字符串。

与 CHAR 值不同，BINARY 值在字符集转换期间不会进行转换。

BINARY 在语义上等同于 VARBINARY。它是一种可变长度类型。在其它数据库管理系统中，BINARY 是一种固定长度类型。

另请参见

- “VARBINARY 数据类型”一节第 103 页
- “LONG BINARY 数据类型”一节第 102 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

IMAGE 数据类型

IMAGE 数据类型用于存储任意长度的二进制数据。

语法

IMAGE

注释

IMAGE 实现为域，作为 LONG BINARY 类型。

另请参见

- “LONG BINARY 数据类型”一节第 102 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

LONG BINARY 数据类型

LONG BINARY 数据类型用于存储任意长度的二进制数据。

语法

LONG BINARY

注释

最大大小为 2 GB。

另请参见

- “BINARY 数据类型”一节第 101 页
- “VARBINARY 数据类型”一节第 103 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

UNIQUEIDENTIFIER 数据类型

UNIQUEIDENTIFIER 数据类型用于存储 UUID（又称 GUID）值。

语法

UNIQUEIDENTIFIER

注释

UNIQUEIDENTIFIER 数据类型通常用于主键或其它唯一列，以保存唯一标识行的 UUID（通用唯一标识符）值。NEWID 函数以这样一种方式来生成 UUID 值：在一台计算机上生成的值与在另一台计算机上生成的 UUID 不匹配。因此，使用 NEWID 生成的 UNIQUEIDENTIFIER 值可以在同步环境中用作键。

例如：

```
CREATE TABLE T1 (
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    c1 INT );
```

UUID 值又称 GUID（全局唯一标识符）。UUID 值包含连字符，因此它们可与其它 RDBMS 兼容。

SQL Anywhere 会根据需要自动将 UNIQUEIDENTIFIER 值在字符串和二进制值之间进行转换。

UNIQUEIDENTIFIER 值存储为 BINARY(16)，但对客户端应用程序则描述为 BINARY(36)。此描述确保了客户端将该值作为字符串读取时已为结果分配了足够的空间。对于 ODBC 客户端应用程序，uniqueidentifier 值显示为 SQL_GUID 类型。

另请参见

- “NEWID 缺省值”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “NEWID 函数 [Miscellaneous]”一节第 242 页
- “UUIDTOSTR 函数 [String]”一节第 322 页
- “STRTOUUID 函数 [String]”一节第 304 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

VARBINARY 数据类型

VARBINARY 数据类型用于存储具有指定最大长度（以字节为单位）的二进制数据。

语法

VARBINARY [**(** *max-length* **)**]

参数

- **max-length** 值的最大长度（以字节为单位）。该长度必须在 1 到 32767 范围内。如果未指定长度，则值为 1。

注释

在比较过程中，将会精确地逐个字节地对 VARBINARY 值进行比较。与之不同的是，CHAR 数据类型的值是使用数据库的归类序列进行比较。如果一个二进制字符串是另一个二进制字符串的前缀，则认为较短的字符串小于较长的字符串。

与 CHAR 值不同，VARBINARY 值在字符集转换期间不会进行转换。

也可以将 VARBINARY 指定为 BINARY VARYING。无论使用哪种语法，都会将数据类型描述为 VARBINARY。

另请参见

- “BINARY 数据类型”一节第 101 页
- “LONG BINARY 数据类型”一节第 102 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

域

域是内置数据类型的别名，包括适用的精度值和小数位数值，还可以包括 DEFAULT 值和 CHECK 条件。SQL Anywhere 中预先定义了一些域（如货币数据类型），但您也可以添加更多自己的域。

域（也称为**用户定义的数据类型**）使整个数据库中的列都可以自动定义成相同的数据类型，具有相同的 NULL 或 NOT NULL 条件、相同的 DEFAULT 设置和相同的 CHECK 条件。域有利于整个数据库的一致性，并且可以消除一些错误类型。

简单的域

域使用 CREATE DOMAIN 语句进行创建。有关语法的完整说明，请参见“CREATE DOMAIN 语句”一节第 424 页。

以下语句创建一个名为 street_address 的数据类型，这是一个有 35 个字符的字符串。

```
CREATE DOMAIN street_address CHAR( 35 );
```

CREATE DATATYPE 可代替 CREATE DOMAIN 使用，但不建议这样做。

创建数据类型时需要资源权限。创建某个数据类型后，执行 CREATE DOMAIN 语句的用户 ID 就是该数据类型的所有者。任何用户都可以使用该数据类型。与其它数据库对象不同的是，所有者名称从来不用作数据类型名的前缀。

在定义列时，street_address 数据类型的使用方式与任何其它数据类型完全相同。例如，在以下有两列的表中，第二列为 street_address 列：

```
CREATE TABLE twocol (
    id INT,
    street street_address
);
```

域可以被其所有者或具有 DBA 权限的用户使用 DROP DOMAIN 语句进行删除：

```
DROP DOMAIN street_address;
```

只有当数据库中的任何表中都未使用该数据类型时，才能执行此语句。如果尝试删除正在使用的域，会显示消息 [表 'SYSUSERTYPE' 中行的主键正在另一个表中被引用]。

域的约束和缺省值

可以在域中构建许多与列关联的属性，如允许 NULL 值、具有 DEFAULT 值等。以该数据类型定义的任何列都会自动继承 NULL 设置、CHECK 条件和 DEFAULT 值。这样就能以一种类似的含义在整个数据库的列中构建一致性。

例如，SQL Anywhere 示例数据库中的许多主键列是保存 ID 号的整数列。以下语句创建的数据类型可能会对此类列有帮助：

```
CREATE DOMAIN id INT
NOT NULL
DEFAULT AUTOINCREMENT
CHECK( @col > 0 );
```

缺省情况下，使用 id 数据类型创建的列不允许为 NULL，其缺省值为自动递增值，而且必须保存正数。在 *@col* 变量中，可以使用任何标识符取代 *col*。

可以通过显式地提供列的属性来替换该数据类型的属性。无论 id 数据类型的设置为何，以数据类型 id 创建并显式地允许 NULL 值的列都允许使用 NULL 值。

兼容性

- **命名约束和缺省值** 在 SQL Anywhere 中，创建的域具有基本数据类型，也可以具有 NULL 或 NOT NULL 条件、缺省值和 CHECK 条件。不支持命名约束和命名缺省值。
- **创建数据类型** 在 SQL Anywhere 中，可以使用 sp_addtype 系统过程添加域，也可以使用 CREATE DOMAIN 语句。

数据类型转换

类型转换可以自动进行，也可以使用 CAST 或 CONVERT 函数显式地请求进行。还可使用以下函数强制进行类型转换：

- **DATE 函数** 将表达式转换为日期，并删除任何小时、分钟或秒。可能会报告转换错误。
- **STRING 函数** 此函数等同于 `CAST( value AS LONG VARCHAR)`。
- **VALUE+0.0** 等同于 CAST (value AS DECIMAL)。

下面列出了自动数据类型转换的概括性规则：

- 如果字符串在数字表达式中使用或作为需要数字参数的函数参数使用，则字符串将转换为数字。
- 如果数字在字符串表达式中使用或作为字符串函数参数使用，则在使用前会转换为字符串。
- 所有日期常量都指定为字符串。字符串在使用前会自动转换为日期。

在某些情况下不适合进行自动数据库转换。例如，在下面的示例中，自动数据类型转换失败。

```
'12/31/90' + 5
'a' > 0
```

另请参见

- “数据类型转换函数”一节第 119 页
- “DATE 函数 [Date and time]”一节第 164 页
- “STRING 函数 [String]”一节第 303 页
- “CAST 函数 [Data type conversion]”一节第 141 页

数据类型之间的比较

在具有不同数据类型的参数之间执行比较（如 =）时，必须转换其中的一个或多个参数，以便使用一种数据类型进行比较操作。

一些规则可能会导致转换失败或意外的比较结果。在这些情况下，应该使用 CAST 或 CONVERT 显式地转换其中一个参数。

通过将参数显式地转换为其它类型，可以忽略上述转换规则。例如，如果要按 CHAR 来比较 DATE 和 CHAR，则需要将 DATE 显式地转换为 CHAR。请参见“CAST 函数 [Data type conversion]”一节第 141 页。

替换字符

如果字符无法使用要转换后的字符集表示，则改为使用替换字符表示。此类型的转换视为**有损耗**；如果原始字符不能以目标字符集表示，则原始字符丢失。

另外，不仅不同的字符集可能有不同的替换字符，而且一个字符集中的替换字符在另一个字符集也可能是非替换字符。对一个字符执行多次转换时，了解这些信息很重要，因为最后的字符可能不显示为预期的目标字符集的替换字符。

例如，假定客户端字符集为 Windows-1252，而数据库字符集为 ISO_8859-1:1987（美国一些 Unix 版本的缺省设置）。接着，假定非 Unicode 客户端应用程序（例如，嵌入式 SQL）试图将欧元符号插入 CHAR、VARCHAR 或 LONG VARCHAR 列。由于 CHAR 字符集中没有此字符，所以插入 ISO_8859-1:1987 的替换字符 0x1A。

现在，如果将上面的 ISO_8859-1:1987 替换字符读取为 Unicode（例如，通过在 ODBC 中执行 `SELECT * FROM t` 到 SQL_C_WCHAR 绑定列的操作），此字符变为 Unicode 代码点 U+001A。（在 Unicode 中，代码点 U+001A 是记录分隔符控制字符。）但是，Unicode 中的替换字符是代码点 U+FFFD。此示例说明，即使数据包含替换字符，由于执行多次转换，这些字符也可能没有转换为目标字符集的替换字符。

因此，了解和测试在多个字符集之间进行转换时如何使用替换字符很重要。

on_charset_conversion_failure 选项有助于确定在转换期间字符不能以目标字符集表示时的行为。请参见“on_charset_conversion_failure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

另请参见

- “数据类型转换”一节第 106 页
- “CHAR 和 NCHAR 之间的比较”一节第 107 页
- “on_charset_conversion_failure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

CHAR 和 NCHAR 之间的比较

当比较 CHAR 类型（CHAR、VARCHAR、LONG VARCHAR）和 NCHAR 类型（NCHAR、NVARCHAR、LONG NVARCHAR）的值时，SQL Anywhere 将使用推导规则来决定执行比较时所使用的类型。通常，如果一个值基于列引用而另一个值不是，则将以包含列引用的值的类型执行比较。

推导规则主要是判断一个值是否基于列引用建立。如果一个值是不基于列引用的变量、主机变量、文字常量或复杂表达式而另一个值基于列引用，则基于常量的值将隐式地转换为基于列的值的类型。

下面按适用顺序列出推导规则：

- 如果 NCHAR 值基于列引用，则 CHAR 将隐式地转换为 NCHAR，并以 NCHAR 执行比较。这包括 NCHAR 和 CHAR 都基于列引用的情况。
- 如果 NCHAR 值不基于列引用，但 CHAR 值基于列引用，则 NCHAR 值将隐式地转换为 CHAR，并且以 CHAR 执行比较。

  如果您预计有 NCHAR 到 CHAR 的转换，考虑 on_charset_conversion_failure 选项的设置很重要，因为此选项控制不能在 CHAR 字符集中表示 NCHAR 字符时的行为。有关详细说明，请参见“将 NCHAR 转换为 CHAR”一节第 109 页。
- 如果两者的值都不基于列引用，则 CHAR 将隐式地转换为 NCHAR，并以 NCHAR 执行比较。

示例

条件 `Employees.GivenName = N'Susan'` 将 CHAR 列 (Employees.GivenName) 与文字 N'Susan' 作比较。值 N'Susan' 会先转换为 CHAR，然后再执行比较，就如同将比较写为：

```
Employees.GivenName  = CAST( N'Susan' AS CHAR )
```

或者，在条件 `Employees.GivenName = T.nchar_column` 中，值 T.nchar_column 不会转换为 CHAR。比较的执行就好像按如下方法编写比较（不能使用 Employees.GivenName 上的索引）：

```
CAST( Employees.GivenName AS NCHAR ) = T.nchar_column;
```

另请参见

- “将 NCHAR 转换为 CHAR”一节第 109 页
- “替换字符”一节第 106 页
- “CAST 函数 [Data type conversion]”一节第 141 页
- “CONVERT 函数 [Data type conversion]”一节第 152 页
- “CAST 函数 [Data type conversion]”一节第 141 页
- “on_charset_conversion_failure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

数字数据类型之间的比较

SQL Anywhere 在比较数字数据类型时使用以下规则（按列出顺序检查规则，并使用适用的第一条规则）：

1. 如果一个参数是 TINYINT，而另一个是 INTEGER，则将两个参数都转换为 INTEGER，然后再进行比较。
2. 如果一个参数是 TINYINT，而另一个是 SMALLINT，则将两个参数都转换为 SMALLINT，然后再进行比较。
3. 如果一个参数是 UNSIGNED SMALLINT，而另一个是 INTEGER，则将两个参数都转换为 INTEGER，然后再进行比较。
4. 如果参数的数据类型有公用的超类型，则转换为公用超类型，然后进行比较。超类型是以下每个列表中的最后一个数据类型：
   - BIT » TINYINT » UNSIGNED SMALLINT » UNSIGNED INTEGER » UNSIGNED BIGINT » NUMERIC
   - SMALLINT » INTEGER » BIGINT » NUMERIC
   - REAL » DOUBLE
   - CHAR » LONG VARCHAR
   - BINARY » LONG BINARY

   例如，如果两个参数的数据类型为 BIT 和 TINYINT，则将它们都转换为 NUMERIC。

时间和日期数据类型之间的比较

SQL Anywhere 在比较时间和日期数据类型时使用以下规则（按列出顺序检查规则，并使用适用的第一条规则）：

1. 如果其中一个参数的数据类型为 TIME，则将两个参数都转换为 TIME，然后再进行比较。
2. 如果其中一个数据类型为 DATE 或 TIMESTAMP，则将两个参数都转换为 TIMESTAMP，然后再进行比较。

   例如，如果两个参数的数据类型为 REAL 和 DATE，则将它们都转换为 TIMESTAMP。
3. 如果一个参数的数据类型为 NUMERIC，而另一个参数的数据类型为 FLOAT，则将两个参数的数据类型都转换为 DOUBLE，然后再进行比较。

其它比较

1. 如果数据类型是 CHAR（如 CHAR、VARCHAR、LONG VARCHAR 等，但不包括 NCHAR 类型）的混合，则转换为 LONG VARCHAR，然后再进行比较。
2. 如果其中一个参数的数据类型是 UNIQUEIDENTIFIER，则转换为 UNIQUEIDENTIFIER，然后再进行比较。
3. 如果其中一个参数的数据类型是位数组（VARBIT 或 LONG VARBIT），则转换为 LONG VARBIT，然后再进行比较。
4. 如果一个参数的数据类型为 CHARACTER，而另一个参数的数据类型为 BINARY，则转换为 BINARY，然后再进行比较。
5. 如果一个参数是 CHAR 类型，而另一个参数是 NCHAR 类型，则使用预定义的推导规则。请参见“CHAR 和 NCHAR 之间的比较”一节第 107 页。
6. 如果没有规则，则转换为 NUMERIC，然后再进行比较。

   例如，如果两个参数的数据类型为 REAL 和 CHAR，则将它们的数据类型都转换为 NUMERIC。

将 NCHAR 转换为 CHAR

NCHAR 到 CHAR 的转换可发生在比较 CHAR 和 NCHAR 数据的过程中，也可以在专门请求时执行。这类转换是有损耗的，因为根据 CHAR 字符集的不同，有些 NCHAR 字符可能无法以 CHAR 字符显示。当 NCHAR 字符无法转换为 CHAR 字符时，将使用 CHAR 字符集的替换字符来代替。对于单字节字符集，该字符通常是十六进制 1A。

根据 on_charset_conversion_failure 选项的设置，不能转换字符时，会发生以下情况之一：

- 使用替换字符，不发出警告
- 使用替换字符，发出警告
- 返回错误

因此，从 NCHAR 转换到 CHAR 时考虑此选项很重要。请参见“on_charset_conversion_failure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

另请参见

- “CHAR 和 NCHAR 之间的比较”一节第 107 页
- “on_charset_conversion_failure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

将 NULL 常量转换为 NUMERIC 和字符串类型

将 NULL 常量转换为 NUMERIC 或字符串类型（如 CHAR、VARCHAR、LONG VARCHAR、BINARY、VARBINARY 和 LONG BINARY）时，大小将设置为 0。例如：

`SELECT CAST( NULL AS CHAR )` 返回 CHAR(0)

`SELECT CAST( NULL AS NUMERIC )` 返回 NUMERIC(1,0)

将日期转换为字符串

SQL Anywhere 提供了几个用于将 SQL Anywhere 日期和时间值转换为各种字符串和其它表达式的函数。在将日期值转换为字符串时，可能会将年份部分缩减为一个表示该年份的两位数字，从而丢失日期中的世纪部分。

错误的世纪值

假定有以下语句，尽管没有发生数据库错误，该语句仍错误地将一个表示日期 2000 年 1 月 1 日的字符串转换为表示日期 1900 年 1 月 1 日的字符串。

```
SELECT DATEFORMAT (
            DATEFORMAT('2000-01-01', 'Mmm dd/yy' ),
            'yyyy-Mmm-dd' )
    AS Wrong_year;
```

SQL Anywhere 会自动将明确的日期字符串 2000-01-01 正确地转换为日期值。然而，内部（或称嵌套）DATEFORMAT 函数的 'Mmm dd/yy' 格式在将日期转换回字符串并传递给外部 DATEFORMAT 函数时会删除日期的世纪部分。

因为在此情况下数据库选项 nearest_century 设置为 0，所以外部 DATEFORMAT 函数会将用两位年份值表示日期的字符串转换为 1900 和 1999 之间的年份。

有关日期和时间函数的详细信息，请参见“日期和时间函数”一节第 120 页。

转换位数组

将整数转换成位数组

将整数转换成位数组时，位数组的长度是整数类型中的位数，而位数组的值是该整数的二进制表示。该整数的最高有效位成为数组的第一位。

示例

`SELECT CAST( CAST( 1 AS BIT ) AS VARBIT )` 返回包含 1 的 VARBIT(1)。

`SELECT CAST( CAST( 8 AS TINYINT ) AS VARBIT )` 返回包含 00001000 的 VARBIT(8)。

`SELECT CAST( CAST( 194 AS INTEGER ) AS VARBIT )` 返回包含 00000000000000000000000011000010 的 VARBIT(32)。

将二进制转换成位数组

将长度为 *n* 的二进制类型转换成位数组时，该数组的长度为 *n* * 8 位。位数组的第一个 8 位成为该二进制值的第一个字节。该二进制值的最高有效位成为该数组的第一位。位数组的下一个 8 位成为该二进制值的第二个字节，以此类推。

示例

`SELECT CAST( 0x8181 AS VARBIT )` 返回包含 1000000110000001 的 VARBIT(16)。

将字符转换成位数组

将长度为 *n* 的字符数据类型转换成位数组时，该数组的长度为 *n* 位。每个字符都必须是 '0' 或 '1'，并且数组的相应位被指派值 0 或 1。

示例

`SELECT CAST( '001100' AS VARBIT )` 返回包含 001100 的 VARBIT(6)。

将位数组转换成整数

将位数组转换为整数数据类型时，将根据该整数类型的存储格式首先使用最高有效位解释位数组的二进制值。

示例

`SELECT CAST( CAST( '11000010' AS VARBIT ) AS INTEGER )` 返回 194 (11000010_2 = 0xC2 = 194)。

将位数组转换成二进制值

将位数组转换成二进制时，数组的第一个 8 位成为该二进制值的第一个字节。数组的第一位成为该二进制值的最高有效位。下一个 8 位用作第二个字节，以此类推。如果位数组的长度不是 8 的倍数，则用附加零填充该二进制值最后一个字节的最低有效位。

示例

`SELECT CAST( CAST( '1111' AS VARBIT ) AS BINARY )` 返回 0xF0（1111_2 变为 11110000_2 = 0xF0）。

`SELECT CAST( CAST( '0011000000110001' AS VARBIT ) AS BINARY )` 返回 0x3031 (0011000000110001_2 = 0x3031)。

将位数组转换成字符

将长度为 *n* 位的位数组转换成字符数据类型时，结果的长度将是 *n* 个字符。结果中的每个字符都必须为 '0' 或 '1'，对应于数组中的位。

示例

SELECT CAST(CAST('01110' AS VARBIT) AS VARCHAR) 返回字符串 '01110'。

数字集之间的转换

将 DOUBLE 类型转换为 NUMERIC 类型时，将保留前 15 个有效数字的精度。

另请参见

- “CAST 函数 [Data type conversion]”一节第 141 页
- “CONVERT 函数 [Data type conversion]”一节第 152 页
- “CAST 函数 [Data type conversion]”一节第 141 页

不明确的字符串到日期的转换

当需要日期值时，即使年份在字符串中仅用两位来表示，SQL Anywhere 也会自动将字符串转换为日期。

如果年份值中的世纪部分被省略，转换方法将取决于 nearest_century 数据库选项。

nearest_century 数据库选项是一个数字，充当 19YY 日期值和 20YY 日期值之间的断点。

小于 nearest_century 值的两位年份值将转换为 20yy，而大于或等于该值的年份将转换为 19yy。

如果未设置此选项，将采用缺省设置 50。因此，两位年份字符串将被理解为是指介于 1950 到 2049 之间的年份。

此 nearest_century 选项是在 SQL Anywhere 5.5 版中引入的。在 5.5 版中，缺省设置为 0。

不明确的日期转换示例

以下语句创建了一个表，该表可用来说明 SQL Anywhere 中不明确日期信息的转换。

```
CREATE TABLE T1 (C1 DATE);
```

表 T1 包含一列 C1，其数据类型为 DATE。

以下语句将日期值插入列 C1 中。SQL Anywhere 会自动转换含有不明确年份值的字符串，这种字符串用两位表示年份，但没有指示世纪。

```
INSERT INTO T1 VALUES('00-01-01');
```

缺省情况下，nearest_century 选项设置为 50，因此 SQL Anywhere 会将上述字符串转换为日期 2000-01-01。以下语句校验此插入的结果。

```
SELECT * FROM T1;
```

用以下语句更改 nearest_century 选项会改变转换过程。

```
SET OPTION nearest_century = 0;
```

当 nearest_century 选项设置为 0 时，用同样的语句执行先前的插入会产生不同的日期值：


```
INSERT INTO T1 VALUES('00-01-01');
```

上述语句此时插入的日期将是 1900-01-01。请使用以下语句校验结果。

```
SELECT * FROM T1;
```

Java 和 SQL 数据类型转换

Java 存储过程和 JDBC 应用程序都需要在 Java 类型和 SQL 类型之间进行数据类型转换。Java 到 SQL 和 SQL 到 Java 数据类型转换是按照 JDBC 标准执行的。以下各表对这些转换做了说明。

Java 到 SQL 的数据类型转换

Java 类型	SQL 类型
String	CHAR
String	VARCHAR
String	TEXT
java.math.BigDecimal	NUMERIC
java.math.BigDecimal	MONEY
java.math.BigDecimal	SMALLMONEY
boolean	BIT
byte	TINYINT
Short	SMALLINT
Int	INTEGER
long	INTEGER
float	REAL
double	DOUBLE
byte[ ]	VARBINARY
byte[ ]	IMAGE
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.lang.Double	DOUBLE

Java 类型	SQL 类型
java.lang.Float	REAL
java.lang.Integer	INTEGER
java.lang.Long	INTEGER

SQL 到 Java 的数据类型转换

SQL 类型	Java 类型
CHAR	String
VARCHAR	String
TEXT	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
MONEY	java.math.BigDecimal
SMALLMONEY	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[ ]
VARBINARY	byte[ ]
LONG VARBINARY	byte[ ]

SQL 类型	Java 类型
IMAGE	byte[ ]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

SQL 函数

目录

函数用于从数据库返回信息。在任何可以使用表达式的地方都可以使用函数。

除非在文档中另有指定，否则如果有任何参数为 NULL，函数都将返回 NULL。

函数使用的语法约定与 SQL 语句使用的语法约定相同。有关语法约定的完整列表，请参见“语法约定”一节第 341 页。

函数类型

本节将按类型对可用的函数进行分组。

集合函数

集合函数对数据库中一组行的数据进行汇总。这些组是使用 SELECT 语句的 GROUP BY 子句构成的。只允许在选择列表和 SELECT 语句的 HAVING 和 ORDER BY 子句中使用集合函数。

函数列表

以下是可用的集合函数：

位数组函数

可以通过位数组函数对位数组执行任务。可以使用以下位数组函数：

- “BIT_AND 函数 [Aggregate]”一节第 136 页
- “BIT_OR 函数 [Aggregate]”一节第 137 页
- “BIT_XOR 函数 [Aggregate]”一节第 139 页
- “BIT_LENGTH 函数 [Bit array]”一节第 137 页
- “BIT_SUBSTR 函数 [Bit array]”一节第 138 页
- “COUNT_SET_BITS 函数 [Bit array]”一节第 158 页
- “GET_BIT 函数 [Bit array]”一节第 200 页
- “SET_BIT 函数 [Bit array]”一节第 288 页
- “SET_BITS 函数 [Aggregate]”一节第 289 页

有关位运算符的信息，请参见“位运算符”一节第 14 页。

另请参见“sa_get_bits 系统过程”一节第 827 页。

秩函数

可以通过秩函数根据在查询中指定的顺序为结果集中的每一行计算排序值。

- “CUME_DIST 函数 [Ranking]”一节第 162 页
- “DENSE_RANK 函数 [Ranking]”一节第 181 页
- “PERCENT_RANK 函数 [Ranking]”一节第 252 页
- “RANK 函数 [Ranking]”一节第 261 页

数据类型转换函数

数据类型转换函数用于将参数从一种数据类型转换为另一种数据类型，或测试它们是否可以进行转换。

函数列表

以下是可用的数据类型转换函数：

- “CAST 函数 [Data type conversion]”一节第 141 页
- “CONVERT 函数 [Data type conversion]”一节第 152 页
- “HEXTOINT 函数 [Data type conversion]”一节第 206 页
- “INTTOHEX 函数 [Data type conversion]”一节第 220 页
- “ISDATE 函数 [Data type conversion]”一节第 221 页
- “ISNUMERIC 函数 [Miscellaneous]”一节第 222 页

日期和时间函数

日期和时间函数对日期和时间数据类型执行操作，或者返回日期或时间信息。

在本章中，**日期时间** 一词用来表示日期或时间，或者时间戳。具体的数据类型 DATETIME 表示为 DATETIME。

有关日期时间数据类型的详细信息，请参见“日期和时间数据类型”一节第 95 页。

函数列表

以下是可用的日期和时间函数：

日期部分

许多日期函数都使用从**日期部分** 生成的日期。下表显示了允许的日期部分的值。

日期部分	缩写	值
年份	yy	1-9999
季度	qq	1-4
Month	mm	1-12
Week	wk	1-54. 一周从星期日开始。
Day	dd	1-31
Dayofyear	dy	1-366
Weekday	dw	1-7（星期日 = 1，……，星期六 = 7）
Hour	hh	0-23
Minute	mi	0-59
Second	ss	0-59
Millisecond	ms	0-999
Calyearofweek	cyr	整数。一周开始的年份。包含一年头几天的周可能在前一年已经开始，具体取决于这一年从星期几开始。从星期一到星期四开始的年没有属于前一年的天，但从星期五到星期日开始的年在这一年的第一个星期一开始第一周。
Calweekofyear	cwk	1-54. 一年内包含指定日期的周的顺序号。
Caldayofweek	cdw	1-7. （星期一 = 1，……，星期日 = 7）

用户定义的函数

用户定义的函数（或 UDF）由程序或环境用户创建。用户定义的函数与内置在程序或环境中的函数相对应。

在 SQL Anywhere 中有两种创建用户定义函数的机制。您可以使用 SQL 语言编写函数，也可以使用 Java。

SQL 中用户定义的函数

您可以使用“CREATE FUNCTION 语句（Web 服务）”一节第 445 页在 SQL 中实现自己的函数。CREATE FUNCTION 语句内的 RETURN 语句决定函数的数据类型。

创建了 SQL 用户定义的函数后，可以在任何使用相同数据类型的内置函数的地方使用该函数。

有关创建 SQL 函数的详细信息，请参见“使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》。

Java 中用户定义的函数

Java 类提供了更加强大灵活的用于实现用户定义函数的方法，而且还有一个额外的优点：如果您愿意，可以将它们从数据库服务器移动到客户端应用程序。

已安装的 Java 类的任何类方法都可以作为用户定义的函数，在任何使用相同数据类型的内置函数的地方使用。

实例方法与类的特定实例相关，所以其行为与标准用户定义的函数不同。

有关创建 Java 类及类方法的详细信息，请参见“创建类”一节 《SQL Anywhere 服务器 - 编程》。

确定是要创建用户定义的函数还是过程

函数与过程相似。确定是要创建函数还是过程取决于您要得到的返回结果，以及将要调用的对象。当确定是要创建 UDF 还是过程时，请考虑它们独特的特性，如下面所列。

函数：

- 可以返回任意类型的单值，且允许使用 RETURNS 子句声明所返回的类型
- 可以在能够使用表达式的大多数地方使用
- 可用于只定义 IN 参数

过程：

- 可以使用 INOUT 或 OUT 参数返回多个值
- 可以返回结果集
- 可以在查询的 FROM 子句中引用，或使用 CALL 语句，或使用 Transact-SQL EXECUTE 语句
- 可以使用命名参数调用

杂项函数

杂类函数对算术、字符串或日期/时间表达式（包括其它函数的返回值）执行运算。

函数列表

以下是可用的杂类函数：

数字函数

数字函数执行数字数据类型的数学运算或者返回数字信息。

函数列表

以下是可用的数字函数：

Web 服务函数

HTTP 函数帮助在 Web 服务内部处理 HTTP 请求。同样，SOAP 函数帮助在 Web 服务内部处理 SOAP 请求。

提供以下函数：

- “HTML_DECODE 函数 [Miscellaneous]”一节第 209 页
- “HTML_ENCODE 函数 [Miscellaneous]”一节第 210 页
- “HTTP_BODY 函数 [HTTP]”一节第 214 页
- “HTTP_DECODE 函数 [HTTP]”一节第 211 页
- “HTTP_ENCODE 函数 [HTTP]”一节第 212 页
- “HTTP_HEADER 函数 [HTTP]”一节第 214 页
- “HTTP_VARIABLE 函数 [HTTP]”一节第 216 页
- “NEXT_HTTP_HEADER 函数 [HTTP]”一节第 245 页
- “NEXT_HTTP_VARIABLE 函数 [HTTP]”一节第 246 页
- “NEXT_SOAP_HEADER 函数 [SOAP]”一节第 247 页
- “SOAP_HEADER 函数 [SOAP]”一节第 292 页

针对 web 服务还提供了许多系统过程。请参见“Web 服务系统过程”一节第 786 页。

字符串函数

字符串函数执行字符串的转换、抽取或处理操作，或者返回有关字符串的信息。

使用多字节字符集工作时，请仔细检查所使用的函数是否返回有关字符或字节的信息。

函数列表

以下是可用的字符串函数：

- “TRIM 函数 [String]”一节第 315 页
- “UCASE 函数 [String]”一节第 317 页
- “UNICODE 函数 [String]”一节第 318 页
- “UNISTR 函数 [String]”一节第 319 页
- “UPPER 函数 [String]”一节第 320 页
- “UUIDTOSTR 函数 [String]”一节第 322 页
- “XMLCONCAT 函数 [String]”一节第 330 页
- “XMLELEMENT 函数 [String]”一节第 331 页
- “XMLFOREST 函数 [String]”一节第 333 页
- “XMLGEN 函数 [String]”一节第 334 页

系统函数

系统函数用于返回系统信息。

函数列表

以下是可用的系统函数：

- “CONNECTION_EXTENDED_PROPERTY 函数 [String]”一节第 149 页
- “CONNECTION_PROPERTY 函数 [System]”一节第 150 页
- “DATALENGTH 函数 [System]”一节第 164 页
- “DB_ID 函数 [System]”一节第 175 页
- “DB_NAME 函数 [System]”一节第 176 页
- “DB_EXTENDED_PROPERTY 函数 [System]”一节第 173 页
- “DB_PROPERTY 函数 [System]”一节第 177 页
- “EVENT_CONDITION 函数 [System]”一节第 189 页
- “EVENT_CONDITION_NAME 函数 [System]”一节第 191 页
- “EVENT_PARAMETER 函数 [System]”一节第 191 页
- “NEXT_CONNECTION 函数 [System]”一节第 243 页
- “NEXT_DATABASE 函数 [System]”一节第 244 页
- “PROPERTY 函数 [System]”一节第 255 页
- “PROPERTY_DESCRIPTION 函数 [System]”一节第 256 页
- “PROPERTY_NAME 函数 [System]”一节第 257 页
- “PROPERTY_NUMBER 函数 [System]”一节第 258 页
- “SUSER_ID 函数 [System]”一节第 308 页
- “SUSER_NAME 函数 [System]”一节第 308 页
- “TSEQUAL 函数 [System]（不再支持）”一节第 316 页
- “USER_ID 函数 [System]”一节第 321 页
- “USER_NAME 函数 [System]”一节第 321 页

注意

- 某些系统函数在 SQL Anywhere 中是作为存储过程来实现的。
- db_id、db_name 和 datalength 函数是作为内置函数来实现的。

下表描述了已实现的系统函数。

系统函数	说明
COL_LENGTH(*table-name*, *column-name* **)**	返回定义的列长度
COL_NAME(*table-id*, *column-id* [, *database-id*] **)**	返回列名称
INDEX_COL (*table-name*, *index-id*, *key_#* [, *userid*] **)**	返回索引列的名称
OBJECT_ID (*object-name* **)**	返回对象 ID
OBJECT_NAME (*object-id* [, *database-id*] **)**	返回对象名称

文本和图像函数

文本和图像函数对文本和图像数据类型进行操作。SQL Anywhere 仅支持 textptr 文本和图像函数。

函数列表

以下是可用的文本和图像函数：

- “TEXTPTR 函数 [Text and image]”一节第 310 页

SQL 函数 (A-D)

列出了每个函数，并在它旁边指出函数类型（数字、字符等等）。

有关指向给定类型所有函数的链接，请参见“函数类型”一节第 118 页。

另请参见

- “SQL 函数 (E-O)”一节第 185 页
- “SQL 函数 (P-Z)”一节第 251 页

ABS 函数 [Numeric]

返回数字表达式的绝对值。

语法

ABS(*numeric-expression* **)**

参数

- **numeric-expression** 要返回其绝对值的数字。

返回值

数字表达式的绝对值。

数字表达式数据类型	返回值
INT	INT
FLOAT	FLOAT
DOUBLE	DOUBLE
NUMERIC	NUMERIC

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性。

示例

以下语句返回值 66。

```
SELECT ABS( -66 );
```

ACOS 函数 [Numeric]

返回数字表达式的反余弦值（以弧度表示）。

语法

ACOS(*numeric-expression* **)**

参数

- **numeric-expression** 角度的余弦值。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

另请参见

- “ASIN 函数 [Numeric]”一节第 131 页
- “ATAN 函数 [Numeric]”一节第 132 页
- “ATAN2 函数 [Numeric]”一节第 133 页
- “COS 函数 [Numeric]”一节第 156 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 0.52 的反余弦值。

```
SELECT ACOS( 0.52 );
```

ARGN 函数 [Miscellaneous]

从参数列表中返回所选参数。

语法

ARGN(*integer-expression*, *expression* [, ...] **)**

参数

- **integer-expression** 表达式列表中的参数位置。
- **expression** 任何传递给函数的数据类型的表达式。提供的所有表达式都必须属于相同的数据类型。

返回值

使用 *integer-expression* 的值作为 n，从其余参数列表中返回第 n 个参数（从 1 开始）。

注释

虽然表达式可以是任意数据类型，但它们必须具有相同的数据类型。整数表达式必须是从 1 到列表中的表达式数目，否则返回 NULL。多个表达式之间以逗号分隔。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 6。

```
SELECT ARGN( 6, 1,2,3,4,5,6 );
```

ASCII 函数 [String]

返回字符串表达式中第一个字节的整数 ASCII 值。

语法

ASCII(*string-expression* **)**

参数

- **string-expression** 字符串。

返回值

SMALLINT

注释

如果字符串为空，则 ASCII 返回零。文字字符串必须用引号括起来。如果数据库字符集为多字节字符集，并且参数字符串的第一个字符包含一个以上的字节，则结果为 NULL。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 90。

```
SELECT ASCII( 'Z' );
```

ASIN 函数 [Numeric]

返回一个数字的反正弦值（以弧度表示）。

语法

ASIN(*numeric-expression* **)**

参数

- **numeric-expression** 角度的正弦值。

返回值

DOUBLE

注释

SIN 和 ASIN 函数互为逆运算。

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

另请参见

- “ACOS 函数 [Numeric]”一节第 129 页
- “ATAN 函数 [Numeric]”一节第 132 页
- “ATAN2 函数 [Numeric]”一节第 133 页
- “SIN 函数 [Numeric]”一节第 292 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 0.52 的反正弦值。

```
SELECT ASIN( 0.52 );
```

ATAN 函数 [Numeric]

返回一个数字的反正切值（以弧度表示）。

语法

ATAN(*numeric-expression* **)**

注释

ATAN 和 TAN 函数互为逆运算。

参数

- **numeric-expression** 角度的正切值。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

另请参见

- “ACOS 函数 [Numeric]”一节第 129 页
- “ASIN 函数 [Numeric]”一节第 131 页
- “ATAN2 函数 [Numeric]”一节第 133 页
- “TAN 函数 [Numeric]”一节第 309 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 0.52 的反正切值。

```
SELECT ATAN( 0.52 );
```

ATAN2 函数 [Numeric]

返回两个数字的比率的反正切值（以弧度表示）。

语法

{ **ATN2** | **ATAN2** }**(** *numeric-expression-1*, *numeric-expression-2* **)**

参数

- **numeric-expression-1** 要计算其反正切值的比率中的分子。
- **numeric-expression-2** 要计算其反正切值的比率中的分母。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

另请参见

- “ACOS 函数 [Numeric]”一节第 129 页
- “ASIN 函数 [Numeric]”一节第 131 页
- “ATAN 函数 [Numeric]”一节第 132 页
- “TAN 函数 [Numeric]”一节第 309 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 0.52 与 0.60 的比率的反正切值。

```
SELECT ATAN2( 0.52, 0.60 );
```


AVG 函数 [Aggregate]

计算一组行、一个数字表达式或一组唯一值的平均值。

语法 1

AVG(*numeric-expression* | **DISTINCT** *numeric-expression* **)**

语法 2

AVG(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **numeric-expression** 要在一组行上计算其平均值的表达式。
- **DISTINCT 子句** 计算输入中的唯一数字值的平均值。

返回值

如果组不包含任何行，则返回 NULL 值。

如果参数为 DOUBLE，则返回 DOUBLE；否则返回 NUMERIC。

注释

此平均值不包括 *numeric-expression* 为 NULL 值的行。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “SUM 函数 [Aggregate]”一节第 307 页
- “COUNT 函数 [Aggregate]”一节第 157 页

标准和兼容性

- **SQL/2003** 核心特性。语法 2 为特性 T611。

示例

以下语句返回值 49988.623200。

```
SELECT AVG( Salary ) FROM Employees;
```

以下语句返回 Products 表中的平均产品价格：

```
SELECT AVG( DISTINCT UnitPrice ) FROM Products;
```


BASE64_DECODE 函数 [String]

使用 MIME base64 格式进行数据解码，并将字符串以 LONG VARCHAR 类型返回。

语法

BASE64_DECODE(*string-expression* **)**

参数

- **string-expression** 要解码的字符串。请注意，该字符串必须采用 base64 编码格式。

返回值

LONG VARCHAR

另请参见

- “BASE64_ENCODE 函数 [String]”一节第 135 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句在嵌入式 SQL 程序中将一个图像插入到图像表中。输入数据（主机变量）必须采用 base64 编码格式。

```
EXEC SQL INSERT INTO images ( image_data ) VALUES ( BASE64_DECODE ( :img ) );
```

BASE64_ENCODE 函数 [String]

使用 MIME base64 格式对数据编码并返回 7 位 ASCII 字符串类型的值。

语法

BASE64_ENCODE(*string-expression* **)**

参数

- **string-expression** 要编码的字符串。

返回值

LONG VARCHAR

另请参见

- “BASE64_DECODE 函数 [String]”一节第 135 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句从包含图像的表中检索数据，并以 ASCII 格式返回数据。所产生的字符串可以嵌入到电子邮件消息中，随后由接收者进行解码以检索原始图像。

```
SELECT BASE64_ENCODE( image_data ) FROM IMAGES;
```

BIT_AND 函数 [Aggregate]

处理 *n* 个位数组并使用以下逻辑返回其参数的逐位 AND 运算结果：对于所比较的每个位，如果所有位都为 1，则返回 1；否则，返回 0。

语法

BIT_AND(*bit-expression* **)**

参数

- **expression** 要确定值的表达式。这通常是列名。

返回值

LONG VARBIT

另请参见

- “BIT_OR 函数 [Aggregate]”一节第 137 页
- “BIT_XOR 函数 [Aggregate]”一节第 139 页
- “位运算符”一节第 14 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例生成包含 CHAR 列的四个行，然后将值转换为 VARBIT。

```
SELECT BIT_AND( CAST(row_value AS VARBIT) )
FROM dbo.sa_split_list('0001,0111,0100,0011')
```

得出结果 0000 的过程如下：

1. 在第 1 行 (0001) 和第 2 行 (0111) 进行逐位与运算，得到 0001（这两个值的第四位都是 1）。
2. 再将上面的比较结果 (0001) 与第 3 行 (0100) 进行逐位与运算，得到 0000（两个值没有同一位都是 1 的情况）。
3. 然后将上面的比较结果 (0000) 与第 4 行 (0011) 进行逐位与运算，得到 0000（两个值没有同一位都是 1 的情况）。

BIT_LENGTH 函数 [Bit array]

返回数组中存储的位数。

语法

BIT_LENGTH(*bit-expression* **)**

参数

- **bit-expression** 要确定长度的位表达式。

返回值

INT

另请参见

- “BIT_LENGTH 函数 [Bit array]”一节第 137 页
- “CHAR_LENGTH 函数 [String]”一节第 145 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 8：

```
SELECT BIT_LENGTH( '01101011' );
```

BIT_OR 函数 [Aggregate]

处理 *n* 个位数组并使用以下逻辑返回其参数的逐位 OR 运算结果：对于所比较的每个位，如果所有位都为 1，则返回 1；否则，返回 0。

语法

BIT_OR(*bit-expression* **)**

参数

- **expression** 要确定值的表达式。这通常是列名。

返回值

LONG VARBIT

另请参见

- “BIT_AND 函数 [Aggregate]”一节第 136 页
- “BIT_XOR 函数 [Aggregate]”一节第 139 页
- “位运算符”一节第 14 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例生成包含 CHAR 列的四个行，然后将值转换为 VARBIT。

```
SELECT BIT_OR( CAST(row_value AS VARBIT) )
FROM dbo.sa_split_list('0001,0111,0100,0011')
```

得出结果 0111 的过程如下：

1. 将第 1 行 (0001) 与第 2 行 (0111) 进行逐位或运算，得到 0111。
2. 再将上面的比较结果 (0111) 与第 3 行 (0100) 进行逐位或运算，得到 0111。
3. 然后将上面的比较结果 (0111) 与第 4 行 (0011) 进行逐位或运算，得到 0111。

BIT_SUBSTR 函数 [Bit array]

返回位数组的子数组。

语法

BIT_SUBSTR(*bit-expression* [, *start* [, *length*]] **)**

参数

- **bit-expression** 从中抽取子数组的位数组。
- **start** 要返回的子数组的开始位置。如果开始位置为负值，则指定从数组结尾处（而不是开始处）算起的位数。数组中的第一位在位置 1 处。
- **length** 要返回的子数组的长度。如果长度为正值，则指定子数组在开始位置右侧第 *length* 位处结束；如果长度为负值，则至多返回开始位置左侧（包括开始位置）的 *length* 位。

返回值

LONG VARBIT

注释

start 和 *length* 可以是正数也可以是负数。通过使用适当的负数和正数组合，可以从字符串开始处或结束处获取子数组。使用负数作为 *length* 不会影响在子数组中返回的位的顺序。

如果指定了 *length*，则子数组不能超过该长度。如果 start 为零且 length 为非负数，则使用 1 作为 start 的值。如果 start 为零且 length 为负数，则使用 -1 作为 start 的值。

如果未指定 *length*，将继续进行选择，直至到达数组的尾部。

BIT_SUBSTR 函数等同于以下语句，但执行速度更快：

```
CAST( SUBSTR( CAST( bit-expression AS VARCHAR ),
start [, length ] )
AS VARBIT )
```


另请参见

- “SUBSTRING 函数 [String]”一节第 305 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 1101：

```
SELECT BIT_SUBSTR( '001101', 3 );
```

以下语句返回 10110：

```
SELECT BIT_SUBSTR( '01011011101111011111', 2, 5 );
```

以下语句返回 11111：

```
SELECT BIT_SUBSTR( '01011011101111011111', -5, 5 );
```

BIT_XOR 函数 [Aggregate]

处理 *n* 个位数组并使用以下逻辑返回其参数的逐位互斥 OR 运算结果：对于所比较的每个位，如果置位的参数个数是奇数（奇数奇偶校验），则返回 1；否则，返回 0。

语法

BIT_XOR(*bit-expression* **)**

参数

- **bit-expression** 要确定值的表达式。这通常是列名。

返回值

LONG VARBIT

另请参见

- “BIT_AND 函数 [Aggregate]”一节第 136 页
- “BIT_OR 函数 [Aggregate]”一节第 137 页
- “位运算符”一节第 14 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例生成包含 CHAR 列的四个行，然后将值转换为 VARBIT。

```
SELECT BIT_XOR( CAST(row_value AS VARBIT) )
FROM dbo.sa_split_list('0001,0111,0100,0011')
```

得出结果 0001 的过程如下：

1. 将第 1 行 (0001) 和第 2 行 (0111) 进行逐位异或 (XOR) 运算，得到 0110。
2. 再将上面的比较结果 (0110) 与第 3 行 (0100) 进行逐位 XOR 运算，得到 0010。
3. 然后将上面的比较结果 (0010) 与第 4 行 (0011) 进行逐位 XOR 运算，得到 0001。

BYTE_LENGTH 函数 [String]

返回字符串中的字节数。

语法

BYTE_LENGTH(*string-expression* **)**

参数

- **string-expression** 要计算其长度的字符串。

返回值

INT

注释

返回的长度中包括 *string-expression* 中的尾随空格字符。

NULL 字符串的返回值为 NULL。

如果字符串使用的是多字节字符集，则 BYTE_LENGTH 值可能不同于 CHAR_LENGTH 返回的字符数。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “CHAR_LENGTH 函数 [String]”一节第 145 页
- “DATALENGTH 函数 [System]”一节第 164 页
- “LENGTH 函数 [String]”一节第 227 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 12。

```
SELECT BYTE_LENGTH( 'Test Message' );
```

BYTE_SUBSTR 函数 [String]

返回字符串的子串。子串是用字节而不是字符来计算的。

语法

BYTE_SUBSTR(*string-expression*, *start* [, *length*] **)**

参数

- **string-expression** 从中获取子串的字符串。
- **start** 表示子串开始位置的整数表达式。正整数表示从字符串开始处开始，第一个字符位置为 1。负整数表示子串从字符串结尾处开始，最后一个字符的位置为 -1。
- **length** 表示子串长度的整数表达式。正 *length* 表示从开始位置*向右*的字节数。负 *length* 表示至多返回开始位置左侧（包括开始位置）的 *length* 个字节。

返回值

返回的值取决于 *string-expression* 的类型。此外，所指定的参数决定返回的值是否为 LONG。例如，当您为长度指定小于 32K 的常量时，不会返回 LONG。

BINARY

VARCHAR

NVARCHAR

注释

如果指定了 *length*，则子串不能超过该字节数。*start* 和 *length* 可以是正数也可以是负数。通过使用适当的负数和正数组合，可以从字符串开始处或结束处获取子串。

如果 *start* 为零且 length 为非负数，则使用 1 作为 *start* 的值。如果 *start* 为零且 *length* 为负数，则使用 -1 作为 start 值。

另请参见

- “SUBSTRING 函数 [String]”一节第 305 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 Test。

```
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

CAST 函数 [Data type conversion]

返回转换为提供的数据类型的表达式的值。

CAST、CONVERT、HEXTOINT、和 INTTOHEX 函数可用于十六进制值与其它值之间的相互转换。有关使用这些函数的详细信息，请参见“转换至/自十六进制值”一节第 10 页。

语法

CAST(*expression* **AS** *datatype* **)**

参数

- **expression** 要转换的表达式。
- **data type** 目标数据类型。

返回值

指定的数据类型。

注释

如果不指示字符串类型的长度，数据库服务器会选择适当的长度。如果没有为 DECIMAL 转换指定精度和小数位数，数据库服务器会选择适当的值。

如果使用 CAST 函数来截断字符串，则必须将 string_rtruncation 数据库选项设置为 OFF；否则会出错。建议您使用 LEFT 函数来截断字符串。

另请参见

- “CONVERT 函数 [Data type conversion]”一节第 152 页
- “LEFT 函数 [String]”一节第 226 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

以下函数确保字符串被用作日期：

```
SELECT CAST( '2000-10-31' AS DATE );
```

计算表达式 `1 + 2` 的值，并将结果转换为单字符字符串。

```
SELECT CAST( 1 + 2 AS CHAR );
```

CEILING 函数 [Numeric]

返回第一个大于或等于给定值的整数。对于正数，这被称为向上舍入。

语法

CEILING(*numeric-expression* **)**

参数

- **numeric-expression** 要计算其上限的数字。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

另请参见

- “FLOOR 函数 [Numeric]”一节第 199 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 60。

```
SELECT CEILING( 59.84567 );
```

CHAR 函数 [String]

返回数字所表示的 ASCII 码值的字符。

语法

CHAR(*integer-expression* **)**

参数

- **integer-expression** 要转换成 ASCII 字符的数字。该数字必须在 0 到 255 范围内（含 0 和 255）。

返回值

VARCHAR

注释

根据二进制排序顺序，返回的字符与当前数据库的字符集中提供的数字表达式相对应。

对于值大于 255 或小于 0 的整数表达式，CHAR 返回 NULL。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 Y。

```
SELECT CHAR( 89 );
```

CHARINDEX 函数 [String]

返回一个字符串在另一个字符串中的位置。

语法

CHARINDEX(*string-expression-1*, *string-expression-2* **)**

参数

- **string-expression-1** 要搜索的字符串。
- **string-expression-2** 被搜索的字符串。

返回值

INT

注释

string-expression-1 的第一个字符标识为 1。如果被搜索的字符串包含一个以上另一字符串的实例，则 CHARINDEX 函数返回第一个实例的位置。

如果被搜索的字符串不包含另一个字符串，则 CHARINDEX 函数返回 0。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “SUBSTRING 函数 [String]”一节第 305 页
- “REPLACE 函数 [String]”一节第 278 页
- “LOCATE 函数 [String]”一节第 230 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句仅在姓氏中包含字母 K 时才会从 Employees 表中的 Surname 和 GivenName 列返回姓和名：

```
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX( 'K', Surname ) = 1;
```

返回的结果：

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

CHAR_LENGTH 函数 [String]

返回字符串中的字符数。

语法

CHAR_LENGTH (*string-expression* **)**

参数

- **string-expression** 要计算其长度的字符串。

返回值

INT

注释

返回的长度中包括尾随空格字符。

NULL 字符串的返回值为 NULL。

如果字符串使用的是多字节字符集，则 CHAR_LENGTH 函数返回的值可能不同于 BYTE_LENGTH 函数返回的字节数。

> **注意**
> 对于 CHAR、VARCHAR LONG VARCHAR 以及 NCHAR 数据类型，可以交替使用 CHAR_LENGTH 函数和 LENGTH 函数。不过，对于 BINARY 和位数组数据类型，必须使用 LENGTH 函数。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “BYTE_LENGTH 函数 [String]”一节第 140 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

以下语句返回值 8。

```
SELECT CHAR_LENGTH( 'Chemical' );
```

COALESCE 函数 [Miscellaneous]

返回列表中的第一个非 NULL 表达式。此函数与 ISNULL 函数完全相同。

语法

COALESCE(*expression*, *expression* [, ...] **)**

参数

- **expression** 任意表达式。

  必须至少给此函数传递两个表达式，并且所有表达式必须可以进行比较。

返回值

ANY

注释

仅当所有参数都为 NULL 时结果才是 NULL。

参数可以是任何标量类型，但不必是同一类型。

有关数据库服务器对此函数处理方法的详细说明，请参见“ISNULL 函数 [Miscellaneous]”一节第 222 页。

另请参见

- “ISNULL 函数 [Miscellaneous]”一节第 222 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

以下语句返回值 34。

```
SELECT COALESCE( NULL, 34, 13, 0 );
```

COMPARE 函数 [String]

用于根据可选归类规则比较两个字符串。

语法

COMPARE(
string-expression-1,
string-expression-2
[, { *collation-id*
| *collation-name*[**(***collation-tailoring-string***)**] }]
)

参数

- **string-expression-1** 第一个字符串表达式。
- **string-expression-2** 第二个字符串表达式。

  字符串表达式只能包含使用数据库的字符集编码的字符。

- **collation-id** 指定要使用的排序顺序的变量或整数常量。只能对内置归类使用 *collation-id*。请参见“SORTKEY 函数 [String]”一节第 293 页。

  如果不指定归类名或 ID，缺省值将是 Default Unicode multilingual。

- **collation-name** 指定要使用的归类名称的字符串或字符变量。也可以指定 char_collation 或 db_collation（例如，`COMPARE( 'abc', 'ABC', 'char_collation' );`），以使用数据库的 CHAR 归类。类似地，可以指定 nchar_collation，以使用数据库的 NCHAR 归类。有关有效归类名的列表，请参见“SORTKEY 函数 [String]”一节第 293 页。

- **collation-tailoring-string** 还可以指定归类定制选项 (*collation-tailoring-string*)，以便能够对字符比较进行更多的控制。这些选项采用归类名后包括在圆括号内的 "关键字=值" 对的形式。例如，`'UCA(locale=es;case=LowerFirst;accent=respect)'`。指定这些选项的语法与为 CREATE DATABASE 语句的 COLLATION 子句定义的语法完全相同。请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

  注意
  指定 UCA 归类时，支持所有归类定制选项。对于所有其它归类，仅支持区分大小写定制选项。

返回值

INTEGER，以您选择的归类规则为基础：

值	含义
1	*string-expression-1* 大于 *string-expression-2*
0	*string-expression-1* 等于 *string-expression-2*
-1	*string-expression-1* 小于 *string-expression-2*

注释

即使数据库启用了空格填充，COMPARE 函数也不会将空字符串和仅包含空格的字符串视为相等。COMPARE 函数使用 SORTKEY 函数生成用于比较的归类关键字。因此，空字符串、含有一个空格的字符串和含有两个空格的字符串的比较结果不会是相等。

如果 *string-expression-1* 或 *string-expression-2* 中有一个为 NULL，则结果为 NULL。

另请参见

- “SORTKEY 函数 [String]”一节第 293 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例使用 COMPARE 函数执行三个比较：

```
SELECT COMPARE( 'abc','ABC','UCA(case=LowerFirst)' ),
       COMPARE( 'abc','ABC','UCA(case=Ignore)' ),
       COMPARE( 'abc','ABC','UCA(case=UpperFirst)' );
```

返回的三个值为 -1、0 和 1，分别表示每个比较的结果。第一个比较返回的结果为 -1，表示 *string-expression-2* ('ABC') 小于 *string-expresssion-1* ('abc')。这是因为在第一个 COMPARE 语句中将区分大小写设置为 LowerFirst。

COMPRESS 函数 [String]

压缩字符串，并返回 LONG BINARY 类型的值。

语法

COMPRESS(*string-expression* [, '*compression-algorithm-alias*'])

参数

- **string-expression** 要压缩的字符串。也可以将二进制值传递给此函数。此参数区分大小写，即使是在不区分大小写的数据库中也是如此。
- **compression-algorithm-alias** 用于压缩的算法的别名。支持的值为 zip 和 gzip（两者均基于相同的算法，但使用不同的标头和报尾）。

  Zip 是一种受到广泛支持的压缩算法。Gzip 与 Unix 上的 gzip 实用程序兼容，而 zip 算法则与相应的实用程序不兼容。

  必须以相同算法执行解压缩。

  有关详细信息，请参见“DECOMPRESS 函数 [String]”一节第 177 页。

返回值

LONG BINARY

注释

COMPRESS 返回的值不是人工可读的。如果返回值比原始串长，则其最大增加尺寸将不会超过原始串长度 + 12 字节长度的 0.1%。可以使用 DECOMPRESS 函数来解压缩经过压缩的 *string-expression*。

如果您在表中存储压缩后的值，则列类型应设为 BINARY 或 LONG BINARY 以避免对该数据执行字符集转换操作。

另请参见

- “DECOMPRESS 函数 [String]”一节第 177 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例返回通过使用 gzip 算法压缩字符串 'Hello World' 而创建的二进制字符串的长度。想要确定某个值压缩后的长度是否变短时，此示例会很有用。

```
SELECT LENGTH( COMPRESS( 'Hello world', 'gzip' ) );
```

CONNECTION_EXTENDED_PROPERTY 函数 [String]

返回给定属性的值。允许指定一个可选的特定于属性的字符串参数。

语法

CONNECTION_EXTENDED_PROPERTY(
{ *property-id* | *property-name* }
[, *property-specific-argument* [, *connection-id*]]
)

参数

- **property-id** 连接属性 ID。
- **property-name** 连接属性名称。属性名可以是 CharSet 和 NcharCharSet。
- **property-specific-argument** 与以下连接属性关联的可选属性特定字符串参数。
  - **CharSet** 返回指定标准所知的连接的 CHAR 字符集标签。可以接受的值包括：ASE、IANA、MIME、JAVA、WINDOWS、UTR22、IBM 和 ICU。缺省值为 IANA，但如果数据库连接是通过 TDS 建立的，则缺省值为 ASE。
  - **NcharCharSet** 返回指定标准所知的连接的 NCHAR 字符集标签。可以接受的值与以上列出的 CharSet 的可接受值相同。
- **connection-id** 数据库连接的连接 ID 号。如果未指定值，则使用当前连接的 ID 号。

返回值

返回扩展的连接属性。返回值为 VARCHAR。

注释

CONNECTION_EXTENDED_PROPERTY 函数类似于 CONNECTION_PROPERTY 函数，不同之处是它允许指定一个可选的属性特定的字符串参数。属性特定参数的解释取决于在第一个参数中指定的属性 ID 或属性名。

可以使用 CONNECTION_EXTENDED_PROPERTY 函数返回任何连接属性的值。但是，扩展信息只可用于扩展属性。

另请参见

- “连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CONNECTION_PROPERTY 函数 [System]”一节第 150 页
- “DB_EXTENDED_PROPERTY 函数 [System]”一节第 173 页
- “DB_PROPERTY 函数 [System]”一节第 177 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例返回 Java 标准所知的当前连接的 CHAR 字符集：

```
SELECT CONNECTION_EXTENDED_PROPERTY( 'charset', 'Java' );
```

CONNECTION_PROPERTY 函数 [System]

以字符串形式返回给定连接属性的值。

语法

CONNECTION_PROPERTY(
{ *integer-expression-1* | *string-expression* }
[, *integer-expression-2*] **)**

参数

- **integer-expression-1** 大多数情况下，提供字符串表达式作为第一个参数更为方便。如果您确实提供了整数表达式，则它将是连接属性 ID。可以使用 PROPERTY_NUMBER 函数来确定这一点。
- **string-expression** 连接属性名。必须指定属性 ID 或属性名两者中的一个。

  有关连接属性的列表，请参见“连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **integer-expression-2** 当前数据库连接的连接 ID。如果省略此参数，将使用当前连接。

返回值

VARCHAR

注释

如果省略第二个参数，则使用当前连接。

另请参见

- “连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “PROPERTY_NUMBER 函数 [System]”一节第 258 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回所维护的预准备语句的数目。

```
SELECT CONNECTION_PROPERTY( 'PrepStmt' );
```


CONFLICT 函数 [Miscellaneous]

指示一个列是否是在 SQL Remote 环境下统一数据库中执行的 UPDATE 操作的冲突溯源。

语法

CONFLICT(*column-name* **)**

参数

- **column-name** 要测试是否冲突的列的名称。

返回值

如果列出现在 SQL Remote 消息代理执行的 UPDATE 语句的 VERIFY 列表中，以及要更新的行中列的原始值与该语句的 VALUES 列表中提供的值不匹配，则返回 TRUE。否则，返回 FALSE。

另请参见

- “CREATE TRIGGER 语句”一节第 511 页
- “更新冲突的缺省解决方法”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

CONFLICT 函数旨在用于 SQL Remote RESOLVE UPDATE 触发器中，以避免出现错误消息。为了说明 CONFLICT 函数的使用，请参考下面的表：

```
CREATE TABLE Admin (
  PKey bigint NOT NULL DEFAULT GLOBAL AUTOINCREMENT,
  TextCol CHAR(20) NULL, PRIMARY KEY ( PKey ) );
```

假设统一数据库和远程数据库在 Admin 表中都有以下行：

```
1, 'Initial'
```

现在，在统一数据库中，按以下所示更新行：

```
UPDATE Admin SET TextCol = 'Consolidated Update' WHERE PKey = 1;
```

在远程数据库中，按以下所示将行更新为不同的值：

```
UPDATE Admin SET TextCol = 'Remote Update' WHERE PKey = 1;
```

接下来，在远程数据库中运行 dbremote。该命令将生成一个消息文件，在消息文件中包括要在统一数据库中执行的以下语句：

```
UPDATE Admin SET TextCol='Remote Update'
VERIFY ( TextCol )
VALUES ( 'Initial' )
WHERE PKey=1;
```

当 SQL Remote 消息代理在统一数据库中运行并应用此 UPDATE 语句时，SQL Anywhere 会使用 VERIFY 和 VALUES 子句来确定是否将会触发 RESOLVE UPDATE 触发器。RESOLVE UPDATE

触发器仅当从 SQL Remote 消息代理中执行对统一数据库的更新时才会被触发。以下是一个 RESOLVE UPDATE 触发器的示例：

```
CREATE TRIGGER ResolveUpdateAdmin
RESOLVE UPDATE ON Admin
REFERENCING OLD AS OldConsolidated
    NEW AS NewRemote
    REMOTE as OldRemote
FOR EACH ROW BEGIN
  MESSAGE 'OLD';
  MESSAGE OldConsolidated.PKey || ',' || OldConsolidated.TextCol;
  MESSAGE 'NEW';
  MESSAGE NewRemote.PKey || ',' || NewRemote.TextCol;
  MESSAGE 'REMOTE';
  MESSAGE OldRemote.PKey || ',' || OldRemote.TextCol;
END;
```

由于统一数据库中 TextCol 列的当前值 (`'Consolidated Update'`) 与 VALUES 子句中相应列的值 (`'Initial'`) 不匹配，因此 RESOLVE UPDATE 触发器将会触发。

由于 PKey 列在远程执行的 UPDATE 语句中未被更改，因而在触发器中无法访问 OldRemote.PKey 值，所以触发器运行错误。

CONFLICT 函数通过返回以下值来帮助避免出现此错误：

- 如果 OldRemote.PKey 值不存在，则返回 FALSE。
- 如果 OldRemote.PKey 值存在，但与 OldConsolidated.PKey 匹配，则返回 FALSE。
- 如果 OldRemote.PKey 值存在，且不同于 OldConsolidated.PKey，则返回 TRUE。

可以使用 CONFLICT 函数按以下方式重写触发器，并避免出现该错误：

```
CREATE TRIGGER ResolveUpdateAdmin
RESOLVE UPDATE ON Admin
REFERENCING OLD AS OldConsolidated
    NEW AS NewRemote
    REMOTE as OldRemote
FOR EACH ROW BEGIN
  message 'OLD';
  message OldConsolidated.PKey || ',' || OldConsolidated.TextCol;
  message 'NEW';
  message NewRemote.PKey || ',' || NewRemote.TextCol;
  message 'REMOTE';
  if CONFLICT( PKey ) then
    message OldRemote.PKey;
  end if;
  if CONFLICT( TextCol ) then
    message OldRemote.TextCol;
  end if;
END;
```

CONVERT 函数 [Data type conversion]

返回转换成提供的数据类型的表达式。

CAST、CONVERT、HEXTOINT、和 INTTOHEX 函数可用于十六进制值与其它值之间的相互转换。有关使用这些函数的详细信息，请参见“转换至/自十六进制值”一节第 10 页。

语法

CONVERT(*datatype*, *expression* [, *format-style*] **)**

参数

- **datatype** 表达式将要转换成的数据类型。
- **expression** 要转换的表达式。
- **format-style** 要应用于输出值的样式代码。请在将字符串转换为日期或时间数据类型或将日期或时间数据类型转换为字符串时使用此参数。下表显示了所支持的样式代码，其后为该样式代码产生的输出格式的表示。样式代码分为两列，具体要视输出格式中是否包括世纪而定（例如，不包括世纪时为 06，包括世纪时则为 2006）。

不含世纪 (yy) 的样式代码	含世纪 (yyyy) 的样式代码	输出格式
-	0 或 100	Mmm dd yyyy hh:nnAA
1	101	mm/dd/yy[yy]
2	102	[yy]yy.mm.dd
3	103	dd/mm/yy[yy]
4	104	dd.mm.yy[yy]
5	105	dd-mm-yy[yy]
6	106	dd Mmm yy[yy]
7	107	Mmm dd, yy[yy]
8	108	hh:nn:ss
-	9 或 109	Mmm dd yyyy hh:nn:ss:sssAA
10	110	mm-dd-yy[yy]
11	111	[yy]yy/mm/dd
12	112	[yy]yymmdd
-	13 或 113	dd Mmm yyyy hh:nn:ss:sss（24 小时制，欧洲缺省时间 + 毫秒，4 位数年份）
-	14 或 114	hh:nn:ss:sss（24 小时制）
-	20 或 120	yyyy-mm-dd hh:nn:ss（24 小时制，ODBC 规范，4 位数年份）

不含世纪 (yy) 的样式代码	含世纪 (yyyy) 的样式代码	输出格式
-	21 或 121	yyyy-mm-dd hh:nn:ss.sss（24 小时制，ODBC 规范加毫秒，4 位数年份）

返回值

指定的数据类型。

注释

如果不提供 *format-style* 参数，将使用样式代码 0。

有关每种输出符号（如 Mmm）生成的样式的说明，请参见“date_format 选项 [数据库]”一节《SQL Anywhere 服务器 - 数据库管理》。

另请参见

- “CAST 函数 [Data type conversion]”一节第 141 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句举例说明了格式样式的用法。

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 104 ) FROM SalesOrders;
```

OrderDate
16.03.2000
20.03.2000
23.03.2000
25.03.2000
...

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 7 ) FROM SalesOrders;
```

OrderDate
Mar 16, 00
Mar 20, 00
Mar 23, 00

OrderDate
Mar 25, 00
...

以下语句举例说明了到整数的转换，并返回值 5。

```
SELECT CONVERT( integer, 5.2 );
```

CORR 函数 [Aggregate]

返回一组数字对的相关系数。

语法

CORR(*dependent-expression*, *independent-expression* **)**

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

dependent-expression 和 *independent-expression* 都是数字类型。该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。然后执行以下计算：

COVAR_POP (*y*, *x*) / **STDDEV_POP** (*y*) * **STDDEV_POP** (*x*)

其中，*y* 代表 *dependent-expression*，而 *x* 代表 *independent-expression*。

另请参见

- “集合函数”一节第 118 页
- “COVAR_POP 函数 [Aggregate]”一节第 159 页
- “STDDEV_POP 函数 [Aggregate]”一节第 299 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性。

示例

以下示例执行相关操作，用来查找收入水平是否与年龄相关联。此函数返回值 0.44022675645996。

```
SELECT CORR( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) ) FROM Employees;
```

COS 函数 [Numeric]

返回由其参数所指定的角度（以弧度表示）的余弦值。

语法

COS(*numeric-expression* **)**

参数

- **numeric-expression** 角度（以弧度为单位）。

返回值

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。如果参数为 NULL，则结果为 NULL。

另请参见

- “ACOS 函数 [Numeric]”一节第 129 页
- “COT 函数 [Numeric]”一节第 156 页
- “SIN 函数 [Numeric]”一节第 292 页
- “TAN 函数 [Numeric]”一节第 309 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 0.52 弧度角度的余弦值。

```
SELECT COS( 0.52 );
```

COT 函数 [Numeric]

返回由其参数所指定的角度（以弧度表示）的余切值。

语法

COT(*numeric-expression* **)**

参数

- **numeric-expression** 角度（以弧度为单位）。

返回值

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。如果参数为 NULL，则结果为 NULL。

另请参见

- “COS 函数 [Numeric]”一节第 156 页
- “SIN 函数 [Numeric]”一节第 292 页
- “TAN 函数 [Numeric]”一节第 309 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 0.52 的余切值。

```
SELECT COT( 0.52 );
```

COUNT 函数 [Aggregate]

根据指定的参数计算组中的行数。

语法 1

COUNT(
*
| *expression*
| **DISTINCT** *expression*
)

语法 2

COUNT(
{ * | *expression* }
) OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- ***** 返回每组中的行数。
- **expression** 要返回行数的表达式。
- **DISTINCT expression** 要返回非重复行数的表达式。

返回值

INT

注释

计数中不包括值为 NULL 的行。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “AVG 函数 [Aggregate]”一节第 134 页
- “SUM 函数 [Aggregate]”一节第 307 页

标准和兼容性

- **SQL/2003** 核心特性。语法 2 为特性 T611。

示例

以下语句返回每个唯一的城市和含有该城市值的行数。

```
SELECT City, COUNT( * ) FROM Employees GROUP BY City;
```

COUNT_SET_BITS 函数 [Bit array]

返回数组中设为 1 (TRUE) 的位数的计数。

语法

COUNT_SET_BITS(*bit-expression* **)**

参数

- **bit-expression** 要确定置位的位数组。

返回值

UNSIGNED INT

注释

如果 *bit-expression* 为 NULL，则返回 NULL。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 4：

```
SELECT COUNT_SET_BITS( '00110011' );
```

以下语句返回值 12：

```
SELECT COUNT_SET_BITS( '0011001111111111' );
```


COVAR_POP 函数 [Aggregate]

返回一组数字对的总体协方差。

语法 1

COVAR_POP(*dependent-expression*, *independent-expression* **)**

语法 2

COVAR_POP(*dependent-expression*, *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

dependent-expression 和 *independent-expression* 都是数字类型。该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为空的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。然后执行以下计算：

```
( SUM( y * x ) - SUM( x ) * SUM( y ) / n ) / n
```

其中，*y* 代表 *dependent-expression*，而 *x* 代表 *independent-expression*。

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “COVAR_SAMP 函数 [Aggregate]”一节第 160 页
- “SUM 函数 [Aggregate]”一节第 307 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例衡量雇员的年龄与薪水之间的相关程度。此函数返回值 73785.84005866646。

```
SELECT COVAR_POP( Salary, ( YEAR( NOW( ) ) - YEAR( BirthDate ) ) )
FROM Employees;
```

COVAR_SAMP 函数 [Aggregate]

返回一组数值对的样本协方差。

语法 1

COVAR_SAMP(*dependent-expression*, *independent-expression* **)**

语法 2

COVAR_SAMP(*dependent-expression*, *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

dependent-expression 和 *independent-expression* 都是数字类型。该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为空的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “COVAR_POP 函数 [Aggregate]”一节第 159 页
- “SUM 函数 [Aggregate]”一节第 307 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例返回值 74782.9460054052。

```
SELECT COVAR_SAMP( Salary, ( 2008 - YEAR( BirthDate ) ) )
FROM Employees;
```

CSCONVERT 函数 [String]

在不同字符集之间转换字符串。

语法

CSCONVERT(
string-expression,
target-charset-string
[, *source-charset-string*] [, *options*]])

参数

- **string-expression** 字符串。
- **target-charset-string** 目标字符集。*target-charset-string* 可以是以下值之一：
  - **os_charset** 作为数据库服务器的主机的操作系统所使用的字符集的别名。
  - **char_charset** 数据库使用的 CHAR 字符集的别名。
  - **nchar_charset** 数据库使用的 NCHAR 字符集的别名。
  - **任何其它受支持的字符集标签** 您可以指定 SQL Anywhere 支持的任何一个字符集标签。
  - **source-charset** 原始 *string-expression* 使用的字符集。缺省值为 db_charset（数据库字符集）。*source-charset-string* 可以是以下值之一：
    - **os_charset** 操作系统使用的字符集的别名。
    - **char_charset** 数据库使用的 CHAR 字符集的别名。
    - **nchar_charset** 数据库使用的 NCHAR 字符集的别名。
    - **任何其它受支持的字符集标签** 您可以指定 SQL Anywhere 支持的任何一个字符集标签。
  - **options** 您可以指定以下各选项之一：
    - **读/写 BOM** 缺省情况下，该值设置为 **read_bom= on** 和 **write_bom=off**。可将值更改为 **read_bom=off** 和 **write_bom=on**。

返回值

LONG BINARY

注释

可以运行以下命令来查看 SQL Anywhere 所支持的字符集列表：

```
dbinit -le
```

有关此函数可使用的字符集标签的详细信息，请参见“支持的字符集”一节 《SQL Anywhere 服务器 - 数据库管理》。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的代码段将 mytext 列从繁体中文字符集转换为简体中文字符集：

```
SELECT CSCONVERT( mytext, 'cp936', 'cp950' )
FROM mytable;
```

下面的代码段将 mytext 列的字符集从数据库字符集转换为简体中文字符集：

```
SELECT CSCONVERT( mytext, 'cp936' )
FROM mytable;
```

如果文件名存储在数据库中，则它会以数据库的字符集存储。如果服务器要从名称存储在数据库中（例如，存储在外部存储过程中）的文件读取内容或向该文件写入内容，则必须将其文件名显式地转换为操作系统的字符集，然后才能访问该文件。在数据库中存储并由客户端检索的文件名会自动转换为客户端的字符集，因此不需要进行显式转换。

以下代码段将 filename 列中值的字符集从数据库字符集转换为操作系统字符集：

```
SELECT CSCONVERT( filename, 'os_charset' )
FROM mytable;
```

某个表中包含一个文件名的列表。一个外部存储过程将此表中的一个文件名用作参数，直接从该文件读取信息。可以在不需要进行字符集转换时使用以下语句：

```
SELECT MYFUNC( filename )
FROM mytable;
```

mytable 子句指示带文件名列的表。但是，如果您需要将文件名的字符集转换为操作系统的字符集，则应使用以下语句。

```
SELECT MYFUNC( csconvert( filename, 'os_charset' ) )
FROM mytable;
```

CUME_DIST 函数 [Ranking]

计算某个值在一组行中的相对位置。

语法

CUME_DIST( ) OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分的说明

返回值

介于 0 和 1 之间的 DOUBLE 值。

注释

目前不允许在 CUME_DIST 函数中使用组合排序键。您可以在任何其它秩函数中使用组合排序键。

可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。作为窗口函数使用时，必须指定 ORDER BY 子句，还可以指定 PARTITION BY 子句，但不能指定 ROWS 或 RANGE 子句。请参见“WINDOW 子句”一节 第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “DENSE_RANK 函数 [Ranking]”一节第 181 页
- “PERCENT_RANK 函数 [Ranking]”一节第 252 页
- “RANK 函数 [Ranking]”一节第 261 页

标准和兼容性

- **SQL/2003** SQL/OLAP 特性 T612。

示例

以下示例返回一个包含居住在 California 的职员的薪水累计分布的结果集。

```
SELECT DepartmentID, Surname, Salary,
CUME_DIST() OVER (PARTITION BY DepartmentID
ORDER BY Salary DESC) "Rank"
FROM Employees
WHERE State IN ('CA');
```

以下是该结果集：

DepartmentID	Surname	Salary	Rank
200	Savarino	72300.000	0.333333333333333
200	Clark	45000.000	0.666666666666667
200	Overbey	39300.000	1

DATALENGTH 函数 [System]

返回表达式结果的基础存储的长度（以字节为单位）。

语法

DATALENGTH(*expression* **)**

参数

- **expression** 通常是列的名称。如果 *expression* 是字符串常量，则必须将它用引号括起来。

返回值

UNSIGNED INT

注释

DATALENGTH 函数的返回值如下：

数据类型	DATALENGTH
SMALLINT	2
INTEGER	4
DOUBLE	8
CHAR	数据的长度
BINARY	数据的长度

此函数支持 NCHAR 输入和输出。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 CompanyName 列中最长字符串的长度。

```
SELECT MAX( DATALENGTH( CompanyName ) )
FROM Customers;
```

以下语句返回字符串 '8sdofinsv8s7a7s7gehe4h' 的长度：

```
SELECT DATALENGTH( '8sdofinsv8s7a7s7gehe4h' );
```

DATE 函数 [Date and time]

将表达式转换为日期，并删除任何小时、分钟或秒。

有关控制日期格式的解释的信息，请参见“date_order 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

语法

DATE(*expression* **)**

返回值

DATE

参数

- **expression** 要转换为日期格式的值，通常为字符串。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 1999-01-02 作为日期。

```
SELECT DATE( '1999-01-02 21:20:53' );
```

以下语句返回 SYSOBJECT 系统视图中列出的所有对象的创建日期：

```
SELECT DATE( creation_time ) FROM SYSOBJECT;
```

DATEADD 函数 [Date and time]

返回通过将多个日期部分添加到日期中而产生的日期。

语法

DATEADD(*date-part*, *numeric-expression*, *date-expression* **)**

date-part :
year
| **quarter**
| **month**
| **week**
| **day**
| **dayofyear**
| **hour**
| **minute**
| **second**
| **millisecond**

参数

- **date-part** 要添加到日期中的日期部分。有关日期部分的详细信息，请参见“日期部分”一节 第 120 页。

- **numeric-expression** 要添加到日期中的日期部分的数量。*numeric-expression* 可以是任何数字类型，但其值将截断为整数。
- **date-expression** 要修改的日期。

返回值

TIMESTAMP

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值： 1995-11-02 00:00:00.000.

```
SELECT DATEADD( month, 102, '1987/05/02' );
```

DATEDIFF 函数 [Date and time]

返回两个日期之间的间隔。

语法

DATEDIFF(*date-part*, *date-expression-1*, *date-expression-2* **)**

date-part :
year
| **quarter**
| **month**
| **week**
| **day**
| **dayofyear**
| **hour**
| **minute**
| **second**
| **millisecond**

参数

- **date-part** 指定要测量间隔的日期部分。

  选择上面列出的日期对象之一。有关日期部分的完整列表，请参见“日期部分”一节第 120 页。
- **date-expression-1** 间隔的起始日期。从 *date-expression-2* 中减去此值，以返回这两个参数之间 *date-part* 的数量。
- **date-expression-2** 间隔的结束日期。从此值中减去 *Date-expression-1*，以返回这两个参数之间 *date-part* 的数量。

返回值

INT

注释

此函数计算两个指定日期之间日期部分的数量。结果是等于 date2 与 date1 之差的有符号整数值（以日期部分为单位）。

当结果不是日期部分的偶数倍时，将会截断而不是舍入 DATEDIFF 函数的结果。

当使用 **day** 作为日期部分时，DATEDIFF 函数返回两个指定时间之间（包括第二个日期但不包括第一个日期）的午夜数。

当使用 **month** 作为日期部分时，DATEDIFF 函数返回两个日期之间（包括第二个日期但不包括第一个日期）出现的月第一天的个数。

当使用 **week** 作为日期部分时，DATEDIFF 函数返回两个日期之间（包括第二个日期但不包括第一个日期）的星期日的个数。

对于更小的时间单位存在溢出值：

- **millisecond** 24 天
- **seconds** 68 年
- **minute** 4083 年
- **others** 没有溢出限制

如果超出这些限制，此函数将返回溢出错误。

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下语句返回 1。

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' );
```

以下语句返回 102。

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' );
```

以下语句返回 0。

```
SELECT DATEDIFF( day, '00:00', '23:59' );
```

以下语句返回 4。

```
SELECT DATEDIFF( day,
   '1999/07/19 00:00',
   '1999/07/23 23:59' );
```

以下语句返回 0。

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' );
```

以下语句返回 1。

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' );
```

DATEFORMAT 函数 [Date and time]

返回以指定格式表示日期表达式的字符串。

语法

DATEFORMAT(*datetime-expression*, *string-expression* **)**

参数

- **datetime-expression** 要转换的日期时间。
- **string-expression** 转换后的日期格式。

  有关日期格式说明的信息，请参见“timestamp_format 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  此函数支持 NCHAR 输入和/或输出。

返回值

VARCHAR

注释

任何允许的日期格式都可用于字符串表达式。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 Jan 01, 1989。

```
SELECT DATEFORMAT( '1989-01-01', 'Mmm dd, yyyy' );
```

DATENAME 函数 [Date and time]

以字符串形式返回日期时间值中指定部分的名称（如月份 June）。

语法

DATENAME(*date-part*, *date-expression* **)**

参数

- **date-part** 要指定的日期部分。

  有关允许的日期部分的完整列表，请参见“日期部分”一节第 120 页。

- **date-expression** 要返回日期部分名的日期。该日期必须包含所请求的 *date-part*。

返回值

VARCHAR

注释

DATENAME 函数返回字符串，即使结果是数字类型（如表示日期的 23）。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 May。

```
SELECT DATENAME( month, '1987/05/02' );
```

DATEPART 函数 [Date and time]

返回日期时间值中一部分的值。

语法

DATEPART(*date-part*, *date-expression* **)**

参数

- **date-part** 要返回的日期部分。

  有关允许的日期部分的完整列表，请参见“日期部分”一节第 120 页。

- **date-expression** 要返回的日期部分的日期。

返回值

INT

注释

该日期必须包含 *date-part* 字段。

与星期几对应的数字取决于 first_day_of_week 选项的设置。缺省情况下星期日对应数字 7。

另请参见

- “first_day_of_week 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SET 语句 [T-SQL]”一节第 697 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 5。

```
SELECT DATEPART( month , '1987/05/02' );
```

下面的示例将创建表 TableStatistics，并向其中插入 SalesOrders 表中所存储的每年销售订单总数：

```
CREATE TABLE TableStatistics (
   ID INTEGER NOT NULL DEFAULT AUTOINCREMENT,
   Year INT,
   NumberOrders INT );
INSERT INTO TableStatistics ( Year, NumberOrders )
   SELECT DATEPART( Year, OrderDate ), COUNT(*)
   FROM SalesOrders
   GROUP BY DATEPART( Year, OrderDate );
```

DATETIME 函数 [Date and time]

将表达式转换成时间戳。

语法

DATETIME(*expression* **)**

参数

- **expression** 要转换的表达式。通常是字符串。

返回值

TIMESTAMP

注释

试图转换数字值时会返回错误。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值为 1998-09-09 12:12:12.000 的时间戳。

```
SELECT DATETIME( '1998-09-09 12:12:12.000' );
```

DAY 函数 [Date and time]

返回 1 到 31 之间的一个整数。

语法

DAY(*date-expression* **)**

参数

- **date-expression** 日期。

返回值

SMALLINT

注释

整数 1 到 31 对应于日期中的月号。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 12。

```
SELECT DAY( '2001-09-12' );
```

DAYNAME 函数 [Date and time]

返回日期中的星期几的名称。

语法

DAYNAME(*date-expression* **)**

参数

- **date-expression** 日期。

返回值

VARCHAR

注释

返回的英语名如下：Sunday、Monday、Tuesday、Wednesday、Thursday、Friday、Saturday。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 Saturday。

```
SELECT DAYNAME ( '1987/05/02' );
```

DAYS 函数 [Date and time]

计算天数的函数。有关具体详细信息，请参见此函数的用法。

语法 1：integer

DAYS([*datetime-expression*,] *datetime-expression* **)**

语法 2：timestamp

DAYS(*datetime-expression*, *integer-expression* **)**

参数

- **datetime-expression** 日期和时间。
- **integer-expression** 要添加到 *datetime-expression* 中的天数。如果 *integer-expression* 是负数，则从时间戳中减去相应的天数。如果提供整数表达式，则必须将 *datetime-expression* 显式地转换为日期或时间戳。

  有关转换数据类型的信息，请参见“CAST 函数 [Data type conversion]”一节第 141 页。

返回值

INT，如果指定两个日期时间表达式。

TIMESTAMP，如果指定的第二个参数为整数。

注释

此函数的行为会视所指定信息的不同而变化：

- 如果提供一个日期，此函数返回自 0000-02-29 以来的天数。

  注意
  0000-02-29 并不表示实际日期，它是日期算法使用的日期。

- 如果提供两个日期，此函数返回它们之间的整数天数。替代方法为使用 DATEDIFF 函数。
- 如果提供一个日期和一个整数，此函数会为指定日期添加整数天数。替代方法为使用 DATEADD 函数。

此函数会忽略小时、分钟和秒。

另请参见

- “DATEDIFF 函数 [Date and time]”一节第 166 页
- “DATEADD 函数 [Date and time]”一节第 165 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回整数 729889。

```
SELECT DAYS( '1998-07-13 06:07:12' );
```

以下语句返回整数值 -366，表示第二个日期比第一个日期早 366 天。建议使用第二个示例 (DATEDIFF)。

```
SELECT DAYS( '1998-07-13 06:07:12',
             '1997-07-12 10:07:12' );

SELECT DATEDIFF( day,
   '1998-07-13 06:07:12',
   '1997-07-12 10:07:12' );
```

以下语句返回时间戳 14.07.99 00:00:000.000。建议使用第二个示例 (DATEADD)。


```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 );

SELECT DATEADD( day, 366, '1998-07-13' );
```

DB_EXTENDED_PROPERTY 函数 [System]

返回给定属性的值。允许指定一个可选的特定于属性的字符串参数。

语法

DB_EXTENDED_PROPERTY(
{ *property-id* | *property-name* }
[, *property-specific-argument*
[, *database-id* | *database-name*]]
)

参数

- **property-id** 要查询的数据库属性 ID。
- **property-name** 要查询的数据库属性名称。

  有关数据库属性的完整列表，请参见“数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **property-specific-argument** 以下数据库属性允许指定其它参数，以便返回关于属性的特定信息，如下所述。
  - **CharSet 属性** 指定标准的名称，以便获得标准的缺省 CHAR 字符集标签。可指定的值为：ASE、IANA、MIME、JAVA、WINDOWS、UTR22、IBM 和 ICU。如果未指定任何标准，将使用 IANA 作为缺省标准，但如果数据库连接是通过 TDS 建立的，则缺省标准将是 ASE。
  - **CatalogCollation、Collation 和 NcharCollation 属性** 查询这些属性时，可将以下值指定为 *property-specific-argument*，以便返回特定于归类的信息：
    - **AccentSensitive** 指定 AccentSensitive，以便获得归类的区分重音设置。例如，以下语句返回 NCHAR 归类的区分重音设置：

      ```
      SELECT DB_EXTENDED_PROPERTY( 'NcharCollation', 'AccentSensitive');
      ```

      可能的返回值为：Ignore、Respect 和 French。有关这些值的说明，请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
    - **CaseSensitivity** 指定 CaseSensitivity，以便获得归类的区分大小写设置。可能的返回值为：Ignore、Respect、UpperFirst 和 LowerFirst。有关这些值的说明，请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
    - **PunctuationSensitivity** 指定 PunctuationSensitivity，以便获得归类的区分标点符号设置。可能的返回值为：Ignore、Primary 和 Quaternary。有关这些值的说明，请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
    - **Properties** 指定 Properties，以便获得包含为归类指定的所有定制选项的字符串。有关返回的字符串中的关键字和值的说明，请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **Specification** 指定 Specification，以便获得包含用于归类的完整归类说明的字符串。有关返回的字符串中的关键字和值的说明，请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

  - **DriveType 属性** 指定 dbspace 的名称或文件 ID，以便获得 dbspace 的驱动器类型。返回值是以下值之一：CD、FIXED、RAMDISK、REMOTE、REMOVABLE 或 UNKNOWN。如果未指定任何内容，则返回系统 dbspace 的驱动器类型。如果指定的 dbspace 不存在，则属性函数返回 NULL。如果指定了 dbspace 的名称并且还指定了一个数据库（非当前连接的数据库）的 ID，则该函数也会返回 NULL。
  - **File 属性** 指定 dbspace 名称，以获得数据库根文件的文件名，包括路径。如果未指定任何内容，则返回系统 dbspace 的信息。如果指定的文件不存在，则函数返回 NULL。
  - **FileSize 属性** 指定 dbspace 的名称或文件 ID，以便获得指定文件的大小。另外，可以指定 temporary，以便返回临时 dbspace 的大小；还可以指定 translog，以便返回日志文件的大小。如果未指定任何东西，则返回系统 dbspace 的大小。如果指定的文件不存在，则函数返回 NULL。
  - **FreePages 属性** 指定 dbspace 的名称或文件 ID，以便获得空闲页数。另外，可以指定 temporary，以便返回临时 dbspace 中的空闲页数；还可以指定 translog，以便返回日志文件中的空闲页数。如果未指定任何内容，则返回系统 dbspace 中的空闲页数。如果指定的文件不存在，则函数返回 NULL。
  - **IOParallelism 属性** 指定 dbspace 名称，以便获得 dbspace 支持的并发 I/O 操作数估计值。如果未指定 dbspace，则使用当前系统的 dbspace。
  - **NextScheduleTime 属性** 指定事件名称，以便获得事件的下一调度执行时间。

- **database-id** 数据库 ID 号，即 DB_ID 函数返回值。通常使用数据库名。
- **database-name** 数据库的名称，即 DB_NAME 函数返回值。

返回值

返回 VARCHAR 值。如果省略第三个参数，则使用当前数据库。

注释

DB_EXTENDED_PROPERTY 函数类似于 DB_PROPERTY 函数，不同之处是它允许指定一个可选的 *property-specific-argument* 字符串参数。对 *property-specific-argument* 的解释取决于在第一个参数中指定的属性 ID 或属性名。

比较目录字符串（如表名和过程名）时，数据库服务器使用 CHAR 归类。对于 UCA 归类，目录归类与 CHAR 归类相同，仅定制更改为不区分大小写、不区分重音，以及在主级别中排序标点符号等几点例外。对于传统归类，目录归类与 CHAR 归类相同，仅定制更改为不区分大小写例外。虽然不能显式指定用于目录归类的定制，但可以查询 Specification 属性，以便获得数据库服务器比较目录字符串时使用的完整归类说明。如果需要利用 CHAR 归类和目录归类之间的差异，查询 Specification 属性很有用。例如，假定有不区分标点符号的 CHAR 归类并要执行升级脚本，该升级脚本定义名为 my_procedure 的过程，还设法删除名为 myprocedure 的旧版本。使用 CHAR 归类，以下语句不能获得所需结果，因为 my_procedure 等效于 myprocedure：

```
CREATE PROCEDURE my_procedure( ) ...;
IF EXISTS ( SELECT * FROM SYS.SYSPROCEDURE WHERE proc_name = 'myprocedure' )
```



```
THEN DROP PROCEDURE myprocedure
END IF;
```

但可以执行以下语句以获得所需结果：

```
CREATE PROCEDURE my_procedure( ) ...;
IF EXISTS ( SELECT * FROM SYS.SYSPROCEDURE
   WHERE COMPARE( proc_name, 'myprocedure',
DB_EXTENDED_PROPERTY( 'CatalogCollation', 'Specification' ) ) = 0 )
THEN DROP PROCEDURE myprocedure
END IF;
```

另请参见

- “DB_ID 函数 [System]”一节第 175 页
- “DB_NAME 函数 [System]”一节第 176 页
- “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CONNECTION_PROPERTY 函数 [System]”一节第 150 页
- “CONNECTION_EXTENDED_PROPERTY 函数 [String]”一节第 149 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回当前数据库的位置：

```
SELECT DB_EXTENDED_PROPERTY( 'File' );
```

以下语句返回以页为单位的系统 dbspace 文件大小。

```
SELECT DB_EXTENDED_PROPERTY( 'FileSize' );
```

以下语句返回事务日志的文件大小（以页为单位）。

```
SELECT DB_EXTENDED_PROPERTY( 'FileSize', 'translog' );
```

以下语句返回 NCHAR 归类的区分大小写设置：

```
SELECT DB_EXTENDED_PROPERTY( 'NcharCollation',' CaseSensitivity' );
```

以下语句返回为数据库 CHAR 归类指定的定制选项：

```
SELECT DB_EXTENDED_PROPERTY ( 'Collation', 'Properties' );
```

以下语句返回为数据库 NCHAR 归类指定的完整归类说明：

```
SELECT DB_EXTENDED_PROPERTY( 'NCharCollation', 'Specification' );
```

DB_ID 函数 [System]

返回数据库 ID 号。

语法

DB_ID([*database-name*] **)**

参数

- **database-name** 包含数据库名的字符串。如果不提供 *database-name*，则返回当前数据库的 ID 号。

返回值

INT

另请参见

- “global_database_id 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

执行对象是作为服务器上唯一数据库的 SQL Anywhere 示例数据库时，该语句返回值 0。

```
SELECT DB_ID( 'demo' );
```

执行对象是唯一处于运行状态的数据库时，以下语句返回值 0。

```
SELECT DB_ID( );
```

DB_NAME 函数 [System]

返回具有给定 ID 号的数据库的名称。

语法

DB_NAME([*database-id*] **)**

参数

- **database-id** 数据库的 ID。*database-id* 必须是数字表达式。

返回值

VARCHAR

注释

如果未提供数据库 ID，则返回当前数据库的名称。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

执行对象是作为服务器上唯一数据库的 SQL Anywhere 示例数据库时，该语句返回数据库名 demo。

```
SELECT DB_NAME( 0 );
```

DB_PROPERTY 函数 [System]

返回给定属性的值。

语法

DB_PROPERTY(
{ *property-id* | *property-name* }
[, *database-id* | *database-name*]
)

参数

- **property-id** 数据库属性 ID。
- **property-name** 数据库属性名称。
- **database-id** 数据库 ID 号，即 DB_ID 函数返回值。通常使用数据库名。
- **database-name** 数据库的名称，即 DB_NAME 函数返回值。

返回值

VARCHAR

注释

返回字符串。如果省略第二个参数，则使用当前数据库。

另请参见

- “DB_ID 函数 [System]”一节第 175 页
- “DB_NAME 函数 [System]”一节第 176 页
- “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回当前数据库的页大小（以字节为单位）。

```
SELECT DB_PROPERTY( 'PageSize' );
```

DECOMPRESS 函数 [String]

解压缩字符串，并返回 LONG BINARY 类型的结果值。

语法

DECOMPRESS(*string-expression* **[,** *compression-algorithm-alias***])**

参数

- **string-expression** 要解压缩的字符串。也可以将二进制值传递给此函数。此参数区分大小写，即使是在不区分大小写的数据库中也是如此。
- **compression-algorithm-alias** 用于解压缩的算法的别名（字符串）。支持的值为 zip 和 gzip（两者均基于相同的算法，但使用不同的标头和报尾）。

  Zip 是一种受到广泛支持的压缩算法。Gzip 与 Unix 上的 gzip 实用程序兼容，而 zip 算法则与相应的实用程序不兼容。

  如果未指定任何算法，函数将尝试检测用于压缩字符串的算法。如果指定的算法不正确，或无法检测到正确的算法，则不会解压缩字符串。

  有关压缩的详细信息，请参见“COMPRESS 函数 [String]”一节第 148 页。

返回值

LONG BINARY

注释

此函数可用来解压缩使用 COMPRESS 函数压缩的值。

不需要对存储在已压缩的列中的值使用 DECOMPRESS 函数。已压缩列中的值的压缩与解压缩由数据库服务器自动进行处理。请参见“选择是否压缩列”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “COMPRESS 函数 [String]”一节第 148 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例使用 DECOMPRESS 函数来解压缩虚构表 TableA 的 Attachment 列中的值：

```
SELECT DECOMPRESS ( Attachment, 'gzip' )
FROM TableA;
```

由于 DECOMPRESS 返回二进制值，因此如果原始值为字符类型（如 LONG VARCHAR），则可以应用 CAST 来返回人工可读的值：

```
SELECT CAST ( DECOMPRESS ( Attachment, 'gzip' )
AS LONG VARCHAR ) FROM TableA;
```


DECRYPT 函数 [String]

使用所提供的密钥对字符串解密，并返回 LONG BINARY 类型的值。

语法

DECRYPT(*string-expression*, *key*
[, *algorithm*]
)

algorithm :
'AES'
| **'AES256'**
| **'AES_FIPS'**
| **'AES256_FIPS'**

参数

- **string-expression** 要解密的字符串。也可以将二进制值传递给此函数。此参数区分大小写，即使是在不区分大小写的数据库中也是如此。
- **key** 对 *string-expression* 进行解密所需的加密密钥（字符串）。此密钥必须与用来对 *string-expression* 进行加密的密钥相同，才能获得被加密的原始值。此参数区分大小写，即使是在不区分大小写的数据库中也是如此。

  小心
  对于高度加密的数据库，请务必将密钥的副本保存在安全的位置。如果丢失了加密密钥，则无办法访问数据，即使有技术支持人员协助也如此。此时必须放弃该数据库并创建一个新的数据库。

- **algorithm** 此可选参数指定最初用来加密 *string-expression* 所使用的算法。

返回值

LONG BINARY

注释

有关所支持加密算法的详细信息，请参见“ENCRYPT 函数 [String]”一节第 185 页。

可以使用 DECRYPT 函数对用 ENCRYPT 函数加密的 *string-expression* 进行解密。此函数返回与输入字符串字节数相同的 LONG BINARY 类型值。

要成功解密 *string-expression*，您必须使用对数据进行加密时所使用的密钥。如果指定的加密密钥不正确，将会生成错误。丢失密钥将导致数据无法访问，而无法访问的数据是无法进行恢复的。

注意
并非所有平台上都可以使用 FIPS。有关支持的平台的列表，请参见 http://www.sybase.com/detail?id=1062623。

另请参见

- “ENCRYPT 函数 [String]”一节第 185 页
- “加密数据库的某些部分”一节 《SQL Anywhere 服务器 - 数据库管理》
- “字符串函数”一节第 125 页
- “-fips 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例对 user_info 表中的用户口令进行解密。由于 DECRYPT 函数会将值转换为无法阅读的 LONG BINARY 数据类型，因此使用 CAST 函数将口令转换回 CHAR 数据类型。

```
SELECT CAST( DECRYPT( user_pwd, '8U3dkA' ) AS CHAR(100) ) FROM user_info;
```

DEGREES 函数 [Numeric]

将数字从弧度转换为度数。

语法

DEGREES(*numeric-expression* **)**

参数

- **numeric-expression** 以弧度表示的角度。

返回值

返回由 *numeric-expression* 所指定的角的度数。

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果参数为 NULL，则结果为 NULL。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 29.79380534680281。

```
SELECT DEGREES( 0.52 );
```


DENSE_RANK 函数 [Ranking]

计算值在分区中的排位。对于并列的值，DENSE_RANK 函数不会在排名序列中留出空位。

语法

DENSE_RANK( ) OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分的说明

返回值

INTEGER

注释

可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。作为窗口函数使用时，必须指定 ORDER BY 子句，还可以指定 PARTITION BY 子句，但不能指定 ROWS 或 RANGE 子句。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “CUME_DIST 函数 [Ranking]”一节第 162 页
- “PERCENT_RANK 函数 [Ranking]”一节第 252 页
- “RANK 函数 [Ranking]”一节第 261 页

标准和兼容性

- **SQL/2003** SQL/OLAP 特性 T612。

示例

以下示例返回一个结果集，它提供 Utah 和 New York 州雇员薪水的排位。尽管在返回的结果集中包含 19 条记录，但只列出了 18 个排位，因为在列表中第 7 名职员与第 8 名具有相同的薪水而并列第 7 位。因为 DENSE_RANK 函数不会在排位中留出空位，所以第 9 名雇员的排位为 '8'，而不是 '9'。

```
SELECT DepartmentID, Surname, Salary, State,
DENSE_RANK() OVER (ORDER BY Salary DESC) AS SalaryRank
FROM Employees
WHERE State IN ('NY','UT');
```

以下是该结果集：

DepartmentID	Surname	Salary	State	SalaryRank
100	Shishov	72995.000	UT	1
100	Wang	68400.000	UT	2

DepartmentID	Surname	Salary	State	SalaryRank
100	Cobb	62000.000	UT	3
400	Morris	61300.000	UT	4
300	Davidson	57090.000	NY	5
200	Martel	55700.000	NY	6
100	Blaikie	54900.000	NY	7
400	Diaz	54900.000	UT	7
100	Driscoll	48023.000	UT	8
400	Hildebrand	45829.000	UT	9
100	Whitney	45700.000	NY	10
100	Guevara	42998.000	NY	11
100	Soo	39075.000	NY	12
200	Goggin	37900.000	UT	13
400	Wetherby	35745.000	NY	14
400	Ahmed	34992.000	NY	15
500	Rebeiro	34576.000	UT	16
300	Bigelow	31200.000	UT	17
500	Lynch	24903.000	UT	18

DIFFERENCE 函数 [String]

返回两个字符串表达式之间 SOUNDEX 值的差。

语法

DIFFERENCE (*string-expression-1*, *string-expression-2* **)**

参数

- **string-expression-1** 第一个 SOUNDEX 参数。
- **string-expression-2** 第二个 SOUNDEX 参数。

返回值

SMALLINT

注释

DIFFERENCE 函数对两个字符串的 SOUNDEX 值进行比较，并计算它们之间的相似程度，返回 0 到 4 之间的值，4 表示最佳匹配。

此函数总是会返回某个值。仅当其中一个参数是 NULL 时，结果才是 NULL。

另请参见

- “SOUNDEX 函数 [String]”一节第 296 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 3。

```
SELECT DIFFERENCE( 'test', 'chest' );
```

DOW 函数 [Date and time]

返回 1 到 7 之间的一个数字，表示某个日期是星期几，其中 1 表示星期日，2 表示星期一，依此类推。

语法

DOW(*date-expression* **)**

参数

- **date-expression** 要计算的日期。

返回值

SMALLINT

注释

DOW 函数不受为 first_day_of_week 数据库选项指定的值的影响。例如，即使将 first_day_of_week 设置为 Monday，对于 Monday，DOW 函数仍会返回 2。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 5。

```
SELECT DOW( '1998-07-09' );
```

以下语句查询 Employees 表，然后返回雇员的 StartDate，以与星期几对应的数字表示：

```
SELECT DOW( StartDate ) FROM Employees;
```


SQL 函数 (E-O)

列出了每个函数，并在它旁边指出函数类型（数字、字符等等）。

有关指向给定类型所有函数的链接，请参见“函数类型”一节第 118 页。

另请参见

- “SQL 函数 (A-D)”一节第 129 页
- “SQL 函数 (P-Z)”一节第 251 页

ENCRYPT 函数 [String]

使用所提供的加密密钥对指定值进行加密，并返回 LONG BINARY 类型的值。

语法

ENCRYPT(*string-expression*, *key*
[, *algorithm*]
)

algorithm :
'AES'
| **'AES256'**
| **'AES_FIPS'**
| **'AES256_FIPS'**

参数

- **string-expression** 要加密的数据。也可以将二进制值传递给此函数。此参数区分大小写，即使是在不区分大小写的数据库中也是如此。
- **key** 用来对 *string-expression* 进行加密的加密密钥。解密时必须使用同一密钥才能获得原始值。此参数区分大小写，即使是在不区分大小写的数据库中也是如此。

  与大多数口令一样，最好选择无法被轻易猜到的密钥值。建议选择满足以下条件的密钥值：长度至少为 16 个字符，混合使用大小写并包含数字、字母和特殊字符。每次要对数据进行解密时，都需要使用此密钥。

  > **小心**
  > 对于高度加密的数据库，请务必将密钥的副本保存在安全的位置。如果丢失了加密密钥，就没有办法访问数据，即使有技术支持人员协助也不行。此时必须放弃该数据库并创建一个新的数据库。

- **algorithm** 此可选参数指定在对 *string-expression* 进行加密时所使用的算法。用于高度加密的算法是 Rijndael：它是一种数据块加密算法，美国国家标准与技术协会（National Institute of Standards and Technology，简称 NIST）选择它作为新的数据块编码器高级加密标准（Advanced Encryption Standard，简称 AES）。

  在支持 FIPS 的平台上，可以为 *algorithm* 指定一种 FIPS 算法。

如果不指定 *algorithm*，将使用缺省的 AES。如果启动数据库服务器时使用了 -fips 服务器选项，则将使用 AES_FIPS 作为缺省值。

返回值

LONG BINARY

注释

此函数返回的 LONG BINARY 值的长度至多比输入 *string-expression* 的长度长 31 个字节。该函数的返回值不是人工可读的。可以使用 DECRYPT 函数对用 ENCRYPT 函数加密的 *string-expression* 进行解密。要成功解密 *string-expression*，您必须使用对数据进行加密时所使用的加密密钥和算法。如果指定的加密密钥不正确，将会生成错误。丢失密钥将导致数据无法访问，而无法访问的数据是无法进行恢复的。

如果您在表中存储加密后的值，则列类型应设为 BINARY 或 LONG BINARY 以避免对该数据执行字符集转换操作。

注意
并非所有平台上都可以使用 FIPS。有关受支持的平台的列表，请参见 http://www.sybase.com/detail?id=1062623。

另请参见

- “DECRYPT 函数 [String]”一节第 179 页
- “加密数据库的某些部分”一节 《SQL Anywhere 服务器 - 数据库管理》
- “-fips 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性。

示例

以下触发器对 user_info 表的 user_pwd 列加密。此列包含用户口令，每当更改口令值时便会触发触发器。

```
CREATE TRIGGER encrypt_updated_pwd
BEFORE UPDATE OF user_pwd
ON user_info
REFERENCING NEW AS new_pwd
FOR EACH ROW
BEGIN
    SET new_pwd.user_pwd=ENCRYPT( new_pwd.user_pwd, '8U3dkA' );
END;
```

ERRORMSG 函数 [Miscellaneous]

提供当前错误或者指定的 SQLSTATE 或 SQLCODE 值的错误消息。

语法

ERRORMSG([*sqlstate* | *sqlcode*] **)**

sqlstate: *string*

sqlcode: *integer*

参数

- **sqlstate** 要返回其错误消息的 SQLSTATE 值。
- **sqlcode** 要返回其错误消息的 SQLCODE 值。

返回值

包含错误消息的字符串。

VARCHAR

注释

如果未提供参数，则提供当前状态的错误消息。将进行任何替代（如表名和列名）。

如果提供了参数，则返回所提供的 SQLSTATE 或 SQLCODE 的错误消息，而不进行替代。表名和列名以占位符形式 (%1) 提供。

另请参见

- “按 SQLSTATE 排序的 SQL Anywhere 错误消息”一节 《错误消息》
- “按 SQLCODE 排序的 SQL Anywhere 错误消息”一节 《错误消息》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 SQLCODE -813 的错误消息。

```
SELECT ERRORMSG( -813 );
```

ESTIMATE 函数 [Miscellaneous]

以百分比的形式返回选择性估计，该百分比由查询优化程序根据指定的参数计算得出。

语法

ESTIMATE(*column-name* [, *value* [, *relation-string*]] **)**

参数

- **column-name** 在估计中使用的列。
- **value** 要与列进行比较的值。缺省值为 NULL。
- **relation-string** 用于比较的比较运算符（用单引号括起来）。此参数可接受的值为：'=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' '='、'>'、'<'、'>='、'<='、'<>'、'!='、'!<' 和 '!>'。缺省值为 '='。

返回值

REAL

注释

如果 *value* 为 NULL，则关系字符串 '=' 和 '!=' 将分别被解释为 IS NULL 和 IS NOT NULL 条件。

另请参见

- “INDEX_ESTIMATE 函数 [Miscellaneous]”一节第 219 页
- “ESTIMATE_SOURCE 函数 [杂类]”一节第 188 页
- “EXPERIENCE_ESTIMATE 函数 [Miscellaneous]”一节第 194 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回据估计大于 200 的 EmployeeID 值的百分比。精度值取决于您对数据库执行的操作。

```
SELECT FIRST ESTIMATE( EmployeeID, 200, '>' )
   FROM Employees
   ORDER BY 1;
```

ESTIMATE_SOURCE 函数 [杂类]

提供查询优化程序使用的选择性估计源。

语法

ESTIMATE_SOURCE(
column-name
[, *value*
[, *relation-string*]]
)

参数

- **column-name** 要调查的列的名称。
- **value** 要与列进行比较的值。缺省值为 NULL。
- **relation-string** 用于比较的比较运算符（用单引号括起来）。此参数可接受的值为：'=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' '=、'>'、'<'、'>='、'<='、'<>'、'!='、'!<' 和 '!>'。缺省值为 '='。

返回值

选择性估计源，可以是下列值之一：

- **统计** 当您指定了一个值，并且有可用来估计列中值的平均选择性的存储统计时，使用此来源。仅当值的选择性是统计中存储的有效数字时，统计才可用。目前，如果值至少出现在 1% 的行中，即视其为有效。

- **列** 类似于 Statistics，不同的是值的选择性出现在不到 1% 的行中。这种情况下，使用的选择性是已在统计中存储的、出现在不到 1% 的行中的所有值的平均值。
- **Guess** 当没有相关的索引可以使用，并且没有为列收集统计时，将返回此来源。这种情况下，将使用内置推测。
- **Column-column** 当使用的估计是连接的选择性时，返回此来源。这种情况下，估计的计算方法是：连接结果集中的行数与两个表笛卡尔乘积中的行数的商。
- **索引** 当没有可用于估计选择性的统计，但存在可供探测以估计选择性的索引时，使用此来源。
- **用户** 当存在用户提供的估计，并且 user_estimates 数据库选项未设置为 Disabled 时，返回此来源。

  有关详细信息，请参见“user_estimates 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **Computed** 当优化程序根据其它信息计算统计时，返回此来源。例如，SQL Anywhere 不维护多个列的统计，因此，如果想要得到多个列等式（如 x=5 和 y=10）的估计，并且存在列 x 和 y 的统计，则优化程序会通过将每一列的估计选择性相乘来创建估计。
- **始终** 当测试根据定义为真时，使用此来源。例如，如果值是 1=1。
- **Combined** 当优化程序使用上述多个来源并且合并使用时，使用此来源。
- **Bounded** 它可以限定其它来源之一。这表示 SQL Anywhere 已对估计设置了上限和/或下限。优化程序这样做是为了将估计保持在逻辑范围内。例如，它确保估计不大于 100%，或选择性不小于一行。

注释

如果 *value* 为 NULL，则关系字符串 '=' 和 '!=' 将分别被解释为 IS NULL 和 IS NOT NULL 条件。

另请参见

- “ESTIMATE 函数 [Miscellaneous]”一节第 187 页
- “INDEX_ESTIMATE 函数 [Miscellaneous]”一节第 219 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 Index，这表示查询优化程序已探测索引来估计选择性。

```
SELECT FIRST ESTIMATE_SOURCE( EmployeeID, 200, '>' )
   FROM Employees
   ORDER BY 1;
```

EVENT_CONDITION 函数 [System]

指定何时触发事件处理程序。

语法

EVENT_CONDITION(*condition-name* **)**

参数

- **condition-name** 触发事件的条件。可能的值已经在数据库中预设，并且不区分大小写。每个条件仅对某些特定的事件类型有效。下表列出了一些条件以及它们所适用的事件：

条件名称	单位	适用于……	注释
DBFreePercent	无	DBDiskSpace	
DBFreeSpace	MB	DBDiskSpace	
DBSize	MB	GrowDB	
ErrorNumber	无	RAISERROR	
IdleTime	seconds	ServerIdle	
间隔	seconds	全部	自上次执行处理程序以来的时间
LogFreePercent	无	LogDiskSpace	
LogFreeSpace	MB	LogDiskSpace	
LogSize	MB	GrowLog	
RemainingValues	integer	GlobalAutoincrement	剩余值的数目
TempFreePercent	无	TempDiskSpace	
TempFreeSpace	MB	TempDiskSpace	
TempSize	MB	GrowTemp	

返回值

INT

注释

EVENT_CONDITION 函数不是从事件进行调用时会返回 NULL。

另请参见

- “CREATE EVENT 语句”一节第 429 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下事件定义使用了 EVENT_CONDITION 函数：

```
CREATE EVENT LogNotifier
TYPE LogDiskSpace
WHERE event_condition( 'LogFreePercent' ) < 50
HANDLER
BEGIN
   MESSAGE 'LogNotifier message'
END;
```

EVENT_CONDITION_NAME 函数 [System]

列出 EVENT_CONDITION 的可能的参数。

语法

EVENT_CONDITION_NAME(*integer* **)**

参数

- **integer** 必须大于或等于零。

返回值

VARCHAR

注释

可以使用 EVENT_CONDITION_NAME 函数获得用于 EVENT_CONDITION 函数的所有参数的列表，方法是在整数中循环，直到函数返回 NULL。

EVENT_CONDITION_NAME 函数不是从事件进行调用时会返回 NULL。

另请参见

- “CREATE EVENT 语句”一节第 429 页

标准和兼容性

- **SQL/2003** 服务商扩充。

EVENT_PARAMETER 函数 [System]

提供事件处理程序的上下文信息。

语法

EVENT_PARAMETER(*context-name* **)**

context-name:
AppInfo
| **ConnectionID**

| **DisconnectReason**
| **EventName**
| **Executions**
| **MirrorServerName**
| **NumActive**
| **ScheduleName**
| **SQLCODE**
| **TableName**
| **User**
| *condition-name*

参数

- **context-name** 其中一个预设字符串。这些字符串必须放在引号中，它们区分大小写，并传递以下信息：
  - **AppInfo** 导致事件被触发的连接的 AppInfo 连接属性的值。使用以下语句查看事件上下文外部的属性的值：

    ```
    SELECT CONNECTION_PROPERTY( 'AppInfo' );
    ```

    此参数对 Connect、Disconnect、ConnectFailed、BackupEnd 和 RAISERROR 事件有效。AppInfo 字符串包含了为嵌入式 SQL、ODBC、OLE DB、ADO.NET 和 iAnywhere JDBC 驱动程序连接建立客户端连接的计算机名和应用程序名。
  - **ConnectionId** 导致事件被触发的连接的连接 ID。
  - **DisconnectReason** 表示连接终止原因的字符串。此参数仅对 Disconnect 事件有效。可以接受的结果包括：
    - **abnormal** 由于客户端应用程序在与数据库断开连接之前异常中止而导致断开连接，或者是在客户端和服务器计算机之间发生通信故障而导致断开连接。
    - **connect failed** 连接尝试失败。
    - **drop connection** 执行了一条 DROP CONNECTION 语句。
    - **from client** 由客户端应用程序断开连接。
    - **inactive** 在由 -ti 服务器选项所指定的时间内没有接收到请求。
    - **liveness** 在由 -tl 服务器选项所指定的时间内没有接收到活动包。
  - **EventName** 已触发的事件的名称。
  - **Executions** 事件处理程序执行的次数。
  - **MirrorServerName** 失去与数据库镜像系统中主服务器连接的镜像服务器或仲裁服务器的名称。
  - **NumActive** 事件处理程序的活动实例数。如果想限制事件处理程序，以便在任何给定时间仅执行一个实例，则此参数会有用处。
  - **ScheduleName** 导致事件被触发的调度的名称。如果该事件使用 TRIGGER EVENT 手工触发或者为系统事件，则结果为空字符串。如果在创建调度时没有显式地为其指派名称，则其名称将是事件的名称。
  - **SQLCODE** 连接失败期间所发生错误的 SQLCODE。此参数仅对 ConnectFailed 事件有效。

- **TableName** 表名，与 RemainingValues 一起使用。
- **用户** 导致事件被触发的用户的用户 ID。

此外，可以从 EVENT_PARAMETER 函数访问 EVENT_CONDITION 函数的任何有效 *condition-name* 参数。

下列表列出了 context-name 值及对其有效的系统事件类型。

Context-name 值	有效系统事件类型
AppInfo	BackupEnd、"Connect"、ConnectFailed、"Disconnect"、"RAISERROR"、用户事件
ConnectionID	BackupEnd、"Connect"、"Disconnect"、Global Autoincrement、"RAISERROR"、用户事件
DisconnectReason	"Disconnect"
EventName	all
Executions	all
NumActive	all
SQLCODE	ConnectFailed
TableName	GlobalAutoincrement
用户	BackupEnd、"Connect"、ConnectFailed、"Disconnect"、GlobalAutoincrement、"RAISERROR"、用户事件

返回值

VARCHAR

注释

传递给事件的最大值大小受到服务器最大页面大小（-gp 服务器选项）的限制。过长的值将被截断成小于最大页面大小。

另请参见

- “EVENT_CONDITION 函数 [System]”一节第 189 页
- “CREATE EVENT 语句”一节第 429 页
- “TRIGGER EVENT 语句”一节第 726 页
- “-gp 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例说明如何将字符串参数传递给一个事件。该事件显示其在数据库服务器消息窗口中触发的时间。

```
CREATE EVENT ev_PassedParameter
HANDLER
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' ||
event_parameter( 'time' );
END;
TRIGGER EVENT ev_PassedParameter( "Time"=string(current timestamp ) );
```

EXP 函数 [Numeric]

返回指数函数，即 e 的指定数字次乘方。

语法

EXP(*numeric-expression* **)**

参数

- **numeric-expression** 指数。

返回值

DOUBLE

注释

EXP 函数返回由 *numeric-expression* 指定的值的指数。

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。如果参数为 NULL，则结果为 NULL。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 3269017.3724721107。

```
SELECT EXP( 15 );
```

EXPERIENCE_ESTIMATE 函数 [Miscellaneous]

以百分比的形式返回频率表中的选择性估计，该百分比由查询优化程序根据指定的参数计算得出。

语法

EXPERIENCE_ESTIMATE(
column-name
[, *value*

[, *relation-string*]]
)

参数

- **column-name** 要调查的列的名称。
- **value** 要与列进行比较的值。
- **relation-string** 用于比较的比较运算符。此参数可接受的值为：'=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' '='、'>'、'<'、'>='、'<='、'<>'、'!='、'!<' 和 '!>'。缺省值为 '='。

返回值

REAL

注释

如果 *value* 为 NULL，则关系字符串 = 和 != 将分别被解释为 IS NULL 和 IS NOT NULL 条件。

另请参见

- “ESTIMATE 函数 [Miscellaneous]”一节第 187 页
- “INDEX_ESTIMATE 函数 [Miscellaneous]”一节第 219 页
- “ESTIMATE_SOURCE 函数 [杂类]”一节第 188 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句返回 NULL。

```
SELECT DISTINCT EXPERIENCE_ESTIMATE( EmployeeID, 200, '>' )
FROM Employees;
```

EXPLANATION 函数 [Miscellaneous]

返回 SQL 语句的计划优化策略。

语法

EXPLANATION(
string-expression
[, *cursor-type*]
[, *update-status*]
)

参数

- **string-expression** SQL 语句，通常是 SELECT 语句，但也可以是 UPDATE 或 DELETE 语句。

- **cursor-type** 一种游标类型，以字符串表示。可能的值为 asensitive、insensitive、sensitive 或 keyset-driven。如果未指定 *cursor-type*，则缺省情况下使用 asensitive。
- **update-status** 字符串参数，它采用下列值之一来指示优化程序如何处理给定的游标：

值	说明
READ-ONLY	游标是只读的。
READ-WRITE（缺省值）	可以读取或写入游标。
FOR UPDATE	可以读取或写入游标。这与 READ-WRITE 相同。

返回值

LONG VARCHAR

注释

优化以字符串形式返回。

这些信息可以帮助决定要添加哪些索引或如何构建数据库，以获得更好的性能。

另请参见

- “UltraLite 中的执行计划”一节 《UltraLite - 数据库管理和参考》
- “读取执行计划”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “PLAN 函数 [Miscellaneous]”一节第 254 页
- “GRAPHICAL_PLAN 函数 [Miscellaneous]”一节第 202 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句将 SELECT 语句作为字符串参数传递，并返回查询的执行计划。

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

以下语句将返回一个字符串，其中包含 'select * from Departments where ....' 查询 INSENSITIVE 游标的文本计划的缩写形式。

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100',
   'insensitive', 'read-only' );
```

EXPRTYPE 函数 [Miscellaneous]

返回标识一个表达式的数据类型的字符串。

语法

EXPRTYPE(*string-expression*, *integer-expression* **)**

参数

- **string-expression** SELECT 语句。要查询其数据类型的表达式必须出现在选择列表中。如果该字符串不是有效的 SELECT 语句，则返回 NULL。
- **integer-expression** 所求的表达式在选择列表中的位置。选择列表中的第一项为 1。如果没有与整数表达式值相对应的 SELECT 列表项，则返回 NULL。

返回值

LONG VARCHAR

另请参见

- “SQL 数据类型”第 75 页
- “sa_describe_query 系统过程”一节第 819 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句在针对 SQL Anywhere 示例数据库后会返回 `smallint`。

```
SELECT EXPRTYPE( 'SELECT LineID FROM SalesOrderItems', 1 );
```

FIRST_VALUE 函数 [Aggregate]

从窗口第一行返回值。

语法

FIRST_VALUE(*expression* [{ **RESPECT** | **IGNORE** } **NULLS**])
OVER (*window-spec* **)**

window-spec：请参见下面的“注释”部分

参数

- **expression** 要计算的表达式。例如，列名。

返回值

参数的数据类型。

注释

FIRST_VALUE 函数使您不用自连接就能（依照某种排序）选择表中的第一个值。如果您希望使用第一个值作为计算的基准，此函数很有价值。

FIRST_VALUE 函数从窗口提取第一个记录。然后针对第一个记录计算 *expression* 并返回结果。

如果指定 IGNORE NULL，则返回 *expression* 的第一个非 NULL 值。如果指定 RESPECT NULLS（缺省值），则会返回第一个值，无论其是否为 NULL。

FIRST_VALUE 函数与其它大多数集合函数的不同之处在于：该函数只能与窗口说明一起使用。

可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “窗口集合函数”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “LAST_VALUE 函数 [Aggregate]”一节第 223 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例返回每个雇员的工资和同一部门中最近雇用的雇员的工资之间的关系，以百分比表示：

```
SELECT DepartmentID, EmployeeID,
       100 * Salary / ( FIRST_VALUE( Salary ) OVER (
                          PARTITION BY DepartmentID  ORDER BY StartDate
DESC ) )
          AS percentage
    FROM Employees;
```

DepartmentID	EmployeeID	percentage
500	1658	100
500	1615	110.4284624
500	1570	138.8427097
500	1013	109.5851905
500	921	167.4497049
500	868	113.2393688
500	750	137.7344095
500	703	222.8679276
500	191	119.6642975
400	1751	100
400	1740	99.705647

DepartmentID	EmployeeID	percentage
400	1684	130.969936
400	1643	83.9734797
400	1607	175.1828989
400	1576	197.0164609
...	...	...

雇员 1658 是部门 500 的第一行，表示他们是该部门中最近雇用的雇员，且其百分比为 100%。相对于雇员 1658 的百分比计算部门 500 中其余雇员的百分比。例如，雇员 1570 的收入大约是雇员 1658 的收入的 139%。

如果同一部门中其他雇员的工资与最近雇用的雇员的工资相同，则他们的百分比也是 100%。

FLOOR 函数 [Numeric]

返回不大于某个数字的最大整数。

语法

FLOOR(*numeric-expression* **)**

参数

- **numeric-expression** 值，通常为 FLOAT。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

另请参见

- “CEILING 函数 [Numeric]”一节第 142 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 Floor 值 123。

```
SELECT FLOOR (123);
```

以下语句返回 Floor 值 123。

```
SELECT FLOOR (123.45);
```

以下语句返回 Floor 值 -124。

```
SELECT FLOOR (-123.45);
```

GET_BIT 函数 [Bit array]

返回位数组中指定位的值（1 或 0）。

语法

GET_BIT(*bit-expression*, *position* **)**

参数

- **bit-expression** 包含该位的位数组。
- **position** 要返回状态的位的位置。

返回值

BIT

注释

数组中的位置从左侧开始计数（从 1 开始）。

如果 *position* 超出数组的长度，则返回 0 (false)。

另请参见

- “位运算符”一节第 14 页
- “SET_BIT 函数 [Bit array]”一节第 288 页
- “SET_BITS 函数 [Aggregate]”一节第 289 页
- “sa_get_bits 系统过程”一节第 827 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 1：

```
SELECT GET_BIT( '00110011' , 4 );
```

以下语句返回值 0：

```
SELECT GET_BIT( '00110011' , 5 );
```

GET_IDENTITY 函数 [Miscellaneous]

将值分配给自动增量列。这是使用自动增量生成编号的替代方法。

语法

GET_IDENTITY(*table_name* [**,** *number_to_allocate*] **)**

参数

- **table_name** 表示表名的字符串，可以选择包括所有者名称。
- **number_to_allocate** 为标识分配的开始数字。缺省值为 1。

返回值

UNSIGNED BIGINT

注释

使用自动增量或全局自动增量仍是生成 ID 的最有效方法，但仍提供了此函数作为一种替代方法。此函数假设表定义了自动增量列。缺省情况下，它返回将为表的自动增量列生成的下一个可用值，并保留该值以便没有其它连接可以使用它。

如果未找到表，此函数会返回错误；如果表没有自动增量列，则返回 NULL。如果有多个自动增量列，将使用所找到的第一个自动增量列。

number_to_allocate 是保留值的个数。如果 *number_to_allocate* 大于 1，此函数还保留剩余值。下一次分配将使用当前数字加上 *number_to_allocate* 的值。这可以降低应用程序执行 GET_IDENTITY 函数的频率。

执行 GET_IDENTITY 函数之后不需要执行 COMMIT，因此可以使用用于插入行的同一连接调用 GET_IDENTITY 函数。如果需要几个表的 ID 值，可以使用一个包括多个 GET_IDENTITY 函数调用的 SELECT 语句来获得这些值，如示例中所示。

GET_IDENTITY 函数是非确定型函数，因此对其进行连续调用可能会返回不同的值。优化程序不会对 GET_IDENTITY 函数的结果进行高速缓存。

有关非确定型函数的详细信息，请参见“函数高速缓存”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “CREATE TABLE 语句”一节第 497 页
- “ALTER TABLE 语句”一节第 373 页
- “NUMBER 函数 [Miscellaneous]”一节第 249 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 Customers 表自动增量列 (ID) 的下一个可用值。返回的数字和以下九个值均会保留：

```
SELECT GET_IDENTITY( 'Customers', 10 );
```

GETDATE 函数 [Date and time]

返回当前的年、月、日、小时、分钟、秒和秒的小数部分。

语法

GETDATE( )

返回值

TIMESTAMP

注释

其准确性受系统时钟准确性的限制。

GETDATE 函数返回的信息与 NOW 函数返回的信息及 CURRENT TIMESTAMP 特殊值相同。

另请参见

- “NOW 函数 [Date and time]”一节第 248 页
- “CURRENT TIMESTAMP 特殊值”一节第 57 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回系统日期和时间。

```
SELECT GETDATE( );
```

GRAPHICAL_PLAN 函数 [Miscellaneous]

以字符串的形式返回 XML 格式的 SQL 语句的计划优化策略。

语法

GRAPHICAL_PLAN(
string-expression
[**,** *statistics-level*
[**,** *cursor-type*
[**,** *update-status*]]] **)**

参数

- **string-expression** SQL 语句，通常是 SELECT 语句，但也可以是 UPDATE 或 DELETE 语句。
- **statistics-level** 一个整数。*Statistics-level* 可以是以下值之一：

值	说明
0	仅包括优化程序估计（缺省值）。
2	包括节点统计在内的详细统计。
3	详细统计。

cursor-type 一种游标类型，以字符串表示。可能的值为：asensitive、insensitive、sensitive 或 keyset-driven。如果未指定 *cursor-type*，则缺省情况下使用 asensitive。

update-status 字符串参数，它采用下列值之一来指示优化程序如何处理给定的游标：

值	说明
READ-ONLY	游标是只读的。
READ-WRITE（缺省值）	可以读取或写入游标。
FOR UPDATE	可以读取或写入游标。这与 READ-WRITE 完全相同。

返回值

LONG VARCHAR

另请参见

- “PLAN 函数 [Miscellaneous]”一节第 254 页
- “EXPLANATION 函数 [Miscellaneous]”一节第 195 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下 Interactive SQL 示例将 SELECT 语句作为字符串参数进行传递，并返回执行该查询的计划。它将计划保存到可使用 Interactive SQL 打开并读取的文件 *plan.saplan* 中。

```
SELECT GRAPHICAL_PLAN(  'SELECT * FROM Departments WHERE DepartmentID >
100' );
OUTPUT TO 'plan.saplan' FORMAT TEXT QUOTE '' HEXADECIMAL ASIS;
```

以下语句返回一个字符串，其中包含 `SELECT * FROM Departments WHERE DepartmentID > 100` 查询中由键集驱动的可更新游标的图形式计划。它还使服务器能够以实际执行统计以及优化程序使用的估计统计为计划添加批注。

```
SELECT GRAPHICAL_PLAN(
   'SELECT * FROM Departments WHERE DepartmentID > 100',
   2,
   'keyset-driven', 'for update' );
```

GREATER 函数 [Miscellaneous]

返回两个参数值中的较大参数值。

语法

GREATER(*expression-1*, *expression-2* **)**

参数

- **expression-1** 要比较的第一个参数值。
- **expression-2** 要比较的第二个参数值。

返回值

ANY

注释

如果两个参数相等，则返回第一个。

另请参见

- “LESSER 函数 [Miscellaneous]”一节第 228 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 10。

```
SELECT GREATER( 10, 5 ) FROM dummy;
```

GROUPING 函数 [Aggregate]

标识 GROUP BY 操作结果集中的某一列是否由于其为小计行的一部分或由于基础数据的缘故而为 NULL。

语法

GROUPING(*group-by-expression* **)**

参数

- **group-by-expression** 一个表达式，作为使用 GROUP BY 子句的查询结果集中的分组列出现。此函数可用于标识通过 ROLLUP 或 CUBE 操作添加到结果集中的小计行。

返回值

- **1** 表示 *group-by-expression* 由于其是小计行的一部分而为 NULL。该列不是该行的前缀列。
- **0** 表示 *group-by-expression* 为小计行的前缀列。

另请参见

- “使用 ROLLUP”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “使用 CUBE”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “GROUP BY GROUPING SETS”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “SELECT 语句”一节第 689 页
- “使用 GROUPING 函数检测占位符 NULL”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T611)。

示例

有关正在使用的此函数的示例，请参见“使用 GROUPING 函数检测占位符 NULL”一节 《SQL Anywhere 服务器 - SQL 的用法》。

HASH 函数 [String]

以散列形式返回指定值。

语法

HASH(*string-expression*[, *algorithm*] **)**

参数

- **string-expression** 要散列的值。此参数区分大小写，即使是在不区分大小写的数据库中也是如此。
- **algorithm** 用于散列的算法。可接受的值包括：MD5、SHA1、SHA1_FIPS、SHA256、SHA256_FIPS。缺省情况下使用 MD5 算法。

> **注意**
> FIPS 算法仅供在使用 Certicom 提供的经 FIPS 140-2 认证的软件的系统上使用。

返回值

VARCHAR

注释

使用散列函数可将每个传递到函数的值转换成唯一的字节序列。

如果使用 -fips 选项启动服务器，则使用的算法或行为可能会不同，如下所述：

- 如果指定 SHA1，则使用 SHA1_FIPS
- 如果指定 SHA256，则使用 SHA256_FIPS
- 如果指定 MD5，则返回错误

以下为返回类型，它们取决于所使用的算法：

- MD5 返回 VARCHAR(32)
- SHA1 返回 VARCHAR(40)
- SHA1_FIPS 返回 VARCHAR(40)
- SHA256 返回 VARCHAR(40)
- SHA256_FIPS 返回 VARCHAR(40)

小心
所有这些算法都是单向散列算法。无法通过散列值重新创建原始字符串。

另请参见

- “字符串函数”一节第 125 页
- “-fips 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例创建了一个称为 user_info 的表，以存储应用程序用户的信息（包括用户 ID 和口令）。还会向该表中插入一行。通过使用 HASH 函数和 SHA256 算法求出口令的散列值。如果不想以明文存储口令，而某个外部应用程序需要对口令进行比较，则以此方式存储散列值形式的口令可能会有用处。

```
CREATE TABLE user_info (
  employee_id   INTEGER NOT NULL PRIMARY KEY,
  user_name CHAR(80),
  user_pwd CHAR(80) );
INSERT INTO user_info
  VALUES ( '1', 's_phillips', HASH( 'mypass', 'SHA256' ) );
```

HEXTOINT 函数 [Data type conversion]

返回十六进制字符串的等效十进制整数。

CAST、CONVERT、HEXTOINT、和 INTTOHEX 函数可用于十六进制值与其它值之间的相互转换。有关使用这些函数的详细信息，请参见“转换至/自十六进制值”一节第 10 页。

语法

HEXTOINT(*hexadecimal-string* **)**

参数

- **hexadecimal-string** 要转换成整数的字符串。

返回值

HEXTOINT 函数返回十六进制字符串的等效平台无关 SQL INTEGER。如果从右侧数第 8 位数是数字 8-9 以及大写或小写字母 A-F 中的某一个，并且前面的前导位都是大写或小写字母 F，则该十六

进制值代表负整数。以下语句中对 HEXTOINT 的使用无效，因为该参数表示的正整数值无法表示为有符号 32 位整数：

```
SELECT HEXTOINT( '0x0080000001' );
```

INT

注释

HEXTOINT 函数只接受由数字以及大写或小写字母 A-F 组成的字符串文字或字符串变量（带有或不带有 0x 前缀）。以下都是有效的 HEXTOINT 用法：

```
SELECT HEXTOINT( '0xFFFFFFFF' );
SELECT HEXTOINT( '0x00000100' );
SELECT HEXTOINT( '100' );
SELECT HEXTOINT( '0xffffffff80000001' );
```

存在 0x 前缀时，HEXTOINT 函数会将其删除。如果数据超过 8 位数，则它所表示的值必须可以表示为有符号 32 位整数值。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “INTTOHEX 函数 [Data type conversion]”一节第 220 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 420。

```
SELECT HEXTOINT( '1A4' );
```

HOUR 函数 [Date and time]

返回日期时间的小时部分。

语法

HOUR(*datetime-expression* **)**

参数

- **datetime-expression** 日期时间。

返回值

SMALLINT

注释

返回的值是 0 到 23 之间的某个数字，对应于日期时间的小时。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 21：

```
SELECT HOUR( '1998-07-09 21:12:13' );
```

HOURS 函数 [Date and time]

计算小时的函数。有关具体详细信息，请参见此函数的用法。

语法 1：integer

HOURS ([*datetime-expression*,] *datetime-expression* **)**

语法 2：timestamp

HOURS (*datetime-expression*, *integer-expression* **)**

参数

- **datetime-expression** 日期和时间。
- **integer-expression** 要添加到 *datetime-expression* 中的小时数。如果 *integer-expression* 是负数，则从日期时间中减去相应的小时数。如果提供整数表达式，则必须将 *datetime-expression* 显式地转换为 DATETIME 数据类型。

  有关转换数据类型的信息，请参见“CAST 函数 [Data type conversion]”一节第 141 页。

返回值

INT

TIMESTAMP

注释

所指定的信息会更改此函数的行为：

- 如果提供一个日期，此函数返回自 0000-02-29 以来的小时数。

  注意
  0000-02-29 并不表示实际日期，它是日期算法使用的日期。

- 如果提供两个时间戳，此函数会返回它们之间的整数小时数。替代方法为使用 DATEDIFF 函数。
- 如果提供一个日期和一个整数，此函数会为指定时间戳添加整数小时数。替代方法为使用 DATEADD 函数。

另请参见

- “DATEDIFF 函数 [Date and time]”一节第 166 页
- “DATEADD 函数 [Date and time]”一节第 165 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 4，表示第二个时间戳比第一个时间戳晚四个小时。建议使用第二个示例 (DATEDIFF)。

```
SELECT HOURS( '1999-07-13 06:07:12',
   '1999-07-13 10:07:12' );

SELECT DATEDIFF( hour,
   '1999-07-13 06:07:12',
   '1999-07-13 10:07:12' );
```

以下语句返回值 17517342。

```
SELECT HOURS( '1998-07-13 06:07:12' );
```

以下语句返回日期时间 13.05.99 02:05:070.000。建议使用第二个示例 (DATEADD)。

```
SELECT HOURS(
   CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );

SELECT DATEADD( hour, 5, '1999-05-12 21:05:07' );
```

HTML_DECODE 函数 [Miscellaneous]

对出现在 HTML 文字字符串中的特殊字符实体进行解码。

语法

HTML_DECODE(*string* **)**

参数

- **string** HTML 文档所使用的任意文字字符串。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

此函数在进行以下一组替代后返回字符串参数：

字符	替代
&quot;	"
&#39;	'
&amp;	&
&lt;	<
&gt;	>
&#x*hexadecimal-number*;	Unicode 代码点，指定为十六进制数。例如，' 返回一个撇号。
&#*decimal-number*;	Unicode 代码点，指定为十进制数。例如，™ 返回商标符号。

指定 Unicode 代码点时，如果可以将值转换为数据库字符集中的字符，则将值转换为字符。否则返回未解释的值。

SQL Anywhere 支持 HTML 4.01 规范中指定的所有字符实体引用。请参见 http://www.w3.org/TR/html4/。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “HTML_ENCODE 函数 [Miscellaneous]”一节第 210 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

HTML_ENCODE 函数 [Miscellaneous]

对要插入到 HTML 文档中的字符串中的特殊字符进行编码。

语法

HTML_ENCODE(*string* **)**

参数

- **string** 要在 HTML 文档中使用的任意字符串。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

此函数在进行以下一组替代后返回字符串参数：

字符	替代
"	&quot;
'	&#39;
&	&amp;
<	&lt;
>	&gt;
小于 0x20 的字符 *nn*	&#x*nn*;

此函数支持 NCHAR 输入和/或输出。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “HTML_DECODE 函数 [Miscellaneous]”一节第 209 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

HTTP_DECODE 函数 [HTTP]

对在 HTTP 中使用的字符串中的特殊字符进行解码。

语法

HTTP_DECODE(*string* **)**

参数

- **string** 要在 HTTP 请求中使用的任意字符串。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

此函数在将所有 %*nn* 形式的字符串序列替换为编码为 *nn* 的字符后返回字符串参数，其中 *nn* 为一个十六进制值。此外，所有加号 (+) 都替换为空格。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “HTTP_ENCODE 函数 [HTTP]”一节第 212 页
- “HTTP_HEADER 函数 [HTTP]”一节第 214 页
- “HTTP_VARIABLE 函数 [HTTP]”一节第 216 页
- “NEXT_HTTP_HEADER 函数 [HTTP]”一节第 245 页
- “NEXT_HTTP_VARIABLE 函数 [HTTP]”一节第 246 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

HTTP_ENCODE 函数 [HTTP]

对在 HTTP 中使用的字符串中的特殊字符进行编码。

语法

HTTP_ENCODE(*string* **)**

参数

- **string** 要在 HTTP 请求中使用的任意字符串。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

此函数在进行以下一组替代后返回字符串参数。此外，所有十六进制编码小于 20 或大于 7E 的字符都将替换为 %*nn*，其中 *nn* 为字符编码。

字符	替代
空格	%20
"	%22
#	%23

字符	替代
%	%25
&	%26
,	%2C
;	%3B
<	%3C
>	%3E
[	%5B
\	%5C
]	%5D
`	%60
{	%7B
|	%7C
}	%7D
小于 0x20 且大于 0x7f 的字符编码 *nn*。	%*nn*

此函数支持 NCHAR 输入和/或输出。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “HTTP_DECODE 函数 [HTTP]”一节第 211 页
- “HTTP_HEADER 函数 [HTTP]”一节第 214 页
- “HTTP_VARIABLE 函数 [HTTP]”一节第 216 页
- “NEXT_HTTP_HEADER 函数 [HTTP]”一节第 245 页
- “NEXT_HTTP_VARIABLE 函数 [HTTP]”一节第 246 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

HTTP_BODY 函数 [HTTP]

以二进制形式返回 HTTP 请求的主体。例如，在 POST 请求中，这是原始 POST 数据。

语法

HTTP_BODY( )

参数

None

返回值

LONG VARCHAR

HTTP 请求以二进制形式返回；不执行字符集的转换。

注释

如果请求主体不存在，或者不是从 web 服务调用函数，则将返回 NULL 值。

另请参见

- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_http_php_page 系统过程”一节第 838 页
- “sa_http_php_page_interpreted 系统过程”一节第 838 页
- “sa_http_header_info 系统过程”一节第 837 页
- “sa_set_http_header 系统过程”一节第 894 页
- “sa_set_http_option 系统过程”一节第 895 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

有关 HTTP 请求和 web 服务的详细信息，请参见“SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》。

HTTP_HEADER 函数 [HTTP]

返回 HTTP 标头的值。

语法

HTTP_HEADER(*header-field-name* **)**

参数

- **header-field-name** HTTP 标头字段名称。

返回值

LONG VARCHAR

注释

此函数返回指定的 HTTP 标头字段的值，如果不是从 HTTP 服务进行调用，则返回 NULL。当通过 Web 服务处理 HTTP 请求时，将使用该函数。

如果给定 *header-field-name* 的标头不存在，则返回值为 NULL。如果不是从 Web 服务调用该函数，返回值也是 NULL。

以下是可能会在处理 HTTP Web 服务请求时值得关注的一些标头。有关这些标头的详细信息，请访问 http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html.org/Protocols/rfc2616/rfc2616-sec14.html。

- **Cookie** cookie 值（如果有），由客户端存储，并与请求的 URI 关联。
- **Referer** 包含指向所请求 URI 链接的页面的 URL。
- **Host** 提交请求的主机的名称或 IP。
- **User-Agent** 客户端应用程序的名称。
- **Accept-Encoding** 客户端应用程序可接受的响应的编码列表。

处理 HTTP Web 服务请求时总是会定义这些特殊标头：

- **@HttpMethod** 返回要处理的请求的类型。可接受的值包括 HEAD、GET 或 POST。
- **@HttpURI** 请求的完整 URI，即 HTTP 请求中指定的 URI。
- **@HttpVersion** 请求的 HTTP 版本（例如，1.0 或 1.1）。
- **@HttpQueryString** 返回所请求 URI 的查询部分（如果存在）。

可以通过这些特殊标头对客户端请求的第一行（也称为请求行）进行访问。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_set_http_header 系统过程”一节第 894 页
- “处理 HTTP 标头”一节 《SQL Anywhere 服务器 - 编程》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “HTTP_DECODE 函数 [HTTP]”一节第 211 页
- “HTTP_ENCODE 函数 [HTTP]”一节第 212 页
- “HTTP_HEADER 函数 [HTTP]”一节第 214 页
- “HTTP_VARIABLE 函数 [HTTP]”一节第 216 页
- “NEXT_HTTP_HEADER 函数 [HTTP]”一节第 245 页
- “NEXT_HTTP_VARIABLE 函数 [HTTP]”一节第 246 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在由 HTTP Web 服务所调用的存储过程内，使用以下示例会获取 Cookie 标头的值：

```
SET cookie_value = HTTP_HEADER( 'Cookie' );
```

在由 HTTP Web 服务所调用的存储过程内，使用以下示例会返回第一个 HTTP 标头的值。

```
DECLARE header_name LONG VARCHAR;
DECLARE header_value LONG VARCHAR;
SET header_name = NEXT_HTTP_HEADER( NULL );
SET header_value = HTTP_HEADER( header_name );
```

HTTP_VARIABLE 函数 [HTTP]

返回一个 HTTP 变量的值。

语法

HTTP_VARIABLE(*var-name* [[, *instance*] , *http-header-field*] **)**

参数

- **var-name** HTTP 变量的名称。
- **instance** 当有多个变量具有相同的名称时，则为字段实例的实例编号；当要获取第一个变量时，则为 NULL。当选择列表允许进行多个选择时会有用处。
- **http-header-field** 在有多个部分的请求中，与 *var-name* 中指定的命名字段相关联的标头字段名。

返回值

此函数返回指定的 HTTP 变量的值。在 Web 服务内处理 HTTP 请求时，将使用该函数。

LONG VARCHAR

注释

如果给定 *var-name* 的标头不存在，则返回值为 NULL。

当 Web 服务请求为 POST 且变量数据作为多部分/形式数据进行发送时，HTTP 服务器会分别接收每个变量的 HTTP 标头。如果指定了 *http-header-field* 参数，HTTP_VARIABLE 函数会从 POST 请求返回特定变量的关联多部分/形式数据标头值。

所有输入数据都会经历客户端（例如，浏览器）字符集与数据库字符集之间的字符集转换。不过，如果为 *http-header-field* 指定了 @BINARY，则返回变量输入值而不进行字符集转换。从客户端接收二进制数据（如图像数据）时，这可能会有用处。

如果不是从 Web 服务进行调用，此函数会返回 NULL。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “处理变量”一节 《SQL Anywhere 服务器 - 编程》
- “HTTP_DECODE 函数 [HTTP]”一节第 211 页
- “HTTP_ENCODE 函数 [HTTP]”一节第 212 页
- “HTTP_HEADER 函数 [HTTP]”一节第 214 页
- “NEXT_HTTP_HEADER 函数 [HTTP]”一节第 245 页
- “NEXT_HTTP_VARIABLE 函数 [HTTP]”一节第 246 页
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在由 HTTP Web 服务所调用的存储过程内，使用以下语句会请求图像变量的 Content-Disposition 和 Content-Type 标头：

```
SET v_name = HTTP_VARIABLE( 'image', NULL, 'Content-Disposition' );
SET v_type = HTTP_VARIABLE( 'image', NULL, 'Content-Type' );
```

在由 HTTP Web 服务所调用的存储过程内，使用以下语句会请求图像变量在其当前字符集中的值，即不进行字符集转换的值：

```
SET v_image = HTTP_VARIABLE( 'image', NULL, '@BINARY' );
```

IDENTITY 函数 [Miscellaneous]

为查询中的每个连续行生成整数值（起始整数为 1）。它的实现方式与 NUMBER 函数完全相同。

语法

IDENTITY(*expression* **)**

参数

- **expression** 表达式。将会对该表达式进行分析，但在执行此函数时会忽略该表达式。

返回值

INT

注释

有关 IDENTITY 函数使用方法的说明，请参见“NUMBER 函数 [Miscellaneous]”一节第 249 页。

另请参见

- “NUMBER 函数 [Miscellaneous]”一节第 249 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下语句返回按顺序排列的职员列表。

```
SELECT IDENTITY( 10 ), Surname FROM Employees;
```

IFNULL 函数 [Miscellaneous]

如果第一个表达式为 NULL 值，则返回第二个表达式的值。如果第一个表达式不为 NULL，则返回第三个表达式的值。如果第一个表达式不为 NULL，并且没有第三个表达式，则返回 NULL。

语法

IFNULL(*expression-1*, *expression-2* [, *expression-3*] **)**

参数

- **expression-1** 要计算的表达式。它的值决定返回 *expression-2* 还是 *expression-3*。
- **expression-2** *expression-1* 为 NULL 时的返回值。
- **expression-3** *expression-1* 不为 NULL 时的返回值。

返回值

返回的数据类型取决于 *expression-2* 和 *expression-3* 的数据类型。

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下语句返回值 -66。

```
SELECT IFNULL( NULL, -66 );
```


以下语句返回 NULL，因为第一个表达式不为 NULL，并且没有第三个表达式。

```
SELECT IFNULL( -66, -66 );
```

INDEX_ESTIMATE 函数 [Miscellaneous]

以百分比的形式返回索引中的选择性估计，该百分比由查询优化程序根据指定的参数计算得出。

语法

INDEX_ESTIMATE(*column-name* [**,** *value* [**,** *relation-string*]] **)**

参数

- **column-name** 在估计中使用的列。
- **value** 要与列进行比较的值。缺省值为 NULL。
- **relation-string** 用于比较的比较运算符（用单引号括起来）。此参数可接受的值为：'=' , '>' , '<' , '>=' , '<=' , '<>' , '!=' , '!<' '=、'>、'<、'>=、'<=、'<>、'!=、'!<' 和 '!>'。缺省值为 '='。

返回值

REAL

注释

如果 *value* 为 NULL，则关系字符串 '=' 和 '!=' 将分别被解释为 IS NULL 和 IS NOT NULL 条件。

另请参见

- “ESTIMATE 函数 [Miscellaneous]”一节第 187 页
- “ESTIMATE_SOURCE 函数 [杂类]”一节第 188 页
- “EXPERIENCE_ESTIMATE 函数 [Miscellaneous]”一节第 194 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回估计大于 200 的 EmployeeID 值的百分比。

```
SELECT INDEX_ESTIMATE( EmployeeID, 200, '>' )
FROM Employees;
```

INSERTSTR 函数 [String]

将一个字符串插入到另一个字符串中的指定位置。

语法

INSERTSTR(*integer-expression*, *string-expression-1*, *string-expression-2* **)**

参数

- **integer-expression** 要在其后插入字符串的位置。使用零可在起始处插入字符串。
- **string-expression-1** 要在其中插入其它字符串的字符串。
- **string-expression-2** 要插入的字符串。

返回值

LONG VARCHAR

注释

此函数支持 NCHAR 输入和/或输出。

另请参见

- “STUFF 函数 [String]”一节第 305 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 backoffice。

```
SELECT INSERTSTR( 0, 'office ', 'back' );
```

INTTOHEX 函数 [Data type conversion]

返回包含整数的等效十六进制值的字符串。

语法

INTTOHEX(*integer-expression* **)**

参数

- **integer-expression** 要转换成十六进制的整数。

返回值

VARCHAR

注释

CAST、CONVERT、HEXTOINT、和 INTTOHEX 函数可用于十六进制值与其它值之间的相互转换。有关详细信息，请参见“转换至/自十六进制值”一节第 10 页。

另请参见

- “HEXTOINT 函数 [Data type conversion]”一节第 206 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下语句返回值 0000009c。

```
SELECT INTTOHEX( 156 );
```

ISDATE 函数 [Data type conversion]

测试字符串参数是否可以转换为日期。

语法

ISDATE(*string* **)**

参数

- **string** 要分析的字符串（该字符串将被分析以确定是否代表了一个有效的时间）。

返回值

INT

注释

如果可以进行转换，则此函数返回 1；否则返回 0。如果参数为 NULL，则返回 0。

此函数支持 NCHAR 输入和/或输出。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例从外部文件导入数据，导出包含无效值的行，并将其余的行复制到永久表中。

```
CREATE GLOBAL TEMPORARY TABLE MyData(
      person VARCHAR(100),
      birth_date VARCHAR(30),
      height_in_cms VARCHAR(10)
   ) ON COMMIT PRESERVE ROWS;
   LOAD TABLE MyData FROM 'exported.dat';
   UNLOAD
      SELECT * FROM MyData
      WHERE ISDATE( birth_date ) = 0
  OR ISNUMERIC( height_in_cms ) = 0
   TO 'badrows.dat';
   INSERT INTO PermData
      SELECT person, birth_date, height_in_cms
      FROM MyData
      WHERE ISDATE( birth_date ) = 1
  AND ISNUMERIC( height_in_cms ) = 1;
   COMMIT;
   DROP TABLE MyData;
```

ISNULL 函数 [Miscellaneous]

返回列表中的第一个非 NULL 表达式。此函数与 COALESCE 函数完全相同。

语法

ISNULL(*expression*, *expression* [, ...] **)**

参数

- **expression** 要测试是否为 NULL 的表达式。

  必须至少给此函数传递两个表达式，并且所有表达式必须可以进行比较。

返回值

此函数的返回类型取决于指定的表达式。即，数据库服务器计算此函数时，首先搜索可用于比较所有表达式的数据类型。如果找到通用类型，则数据库服务器比较表达式，然后以用于比较的类型返回结果。如果数据库服务器找不到通用比较类型，则返回错误。

另请参见

- “COALESCE 函数 [Miscellaneous]”一节第 145 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 -66。

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 );
```

ISNUMERIC 函数 [Miscellaneous]

确定字符串参数是否为有效数。

语法

ISNUMERIC(*string* **)**

参数

- **string** 要进行分析以确定是否代表有效数的字符串。

返回值

INT

注释

如果输入字符串经过计算确定为有效整数或浮点数，则 ISNUMERIC 返回 1；否则返回 0。如果字符串仅包含空白或为 NULL，此函数也返回 0。

以下值也会使 ISNUMERIC 函数返回 0：

- 将字母 d 或 D 用作指数分隔符的值。例如，1d2。
- 诸如 NAN、0x12、INF 和 INFINITY 等特殊值。
- NULL（例如 SELECT ISNUMERIC(NULL);）。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例从外部文件导入数据，导出包含无效值的行，并将其余行复制到永久表中。在本示例中，ISNUMERIC 语句将校验 height_in_cms 值中的值是否为数字类型。

```
CREATE GLOBAL TEMPORARY TABLE MyData(
      person VARCHAR(100),
      birth_date VARCHAR(30),
      height_in_cms VARCHAR(10)
   ) ON COMMIT PRESERVE ROWS;
   LOAD TABLE MyData FROM 'exported.dat';
   UNLOAD
      SELECT *
      FROM MyData
      WHERE ISDATE( birth_date ) = 0
  OR ISNUMERIC( height_in_cms ) = 0
   TO 'badrows.dat';
   INSERT INTO PermData
      SELECT person, birth_date, height_in_cms
      FROM MyData
      WHERE ISDATE( birth_date ) = 1
  AND ISNUMERIC( height_in_cms ) = 1;
   COMMIT;
   DROP TABLE MyData;
```

LAST_VALUE 函数 [Aggregate]

从窗口最后一行返回值。

语法

LAST_VALUE(*expression*[{ **RESPECT** | **IGNORE** } **NULLS**])
OVER (*window-spec* **)**

window-spec：请参见下面的“注释”部分

参数

- **expression** 要计算的表达式。例如，列名。

返回值

参数的数据类型。

注释

LAST_VALUE 函数使您不用自连接就能（依照某种排序）选择表中的最后一个值。如果您希望使用最后一个值作为计算的基准，此函数很有价值。

LAST_VALUE 函数在完成 ORDER BY 后从分区提取最后一个记录。然后针对最后一个记录计算 *expression* 并返回结果。

如果指定 IGNORE NULL，则返回 *expression* 的最后一个非 NULL 值。如果指定 RESPECT NULLS（缺省值），则会返回最后一个值，无论其是否为 NULL。

LAST_VALUE 函数与其它大多数集合函数的不同之处在于：该函数只能与窗口说明一起使用。

可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “窗口集合函数”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “FIRST_VALUE 函数 [Aggregate]”一节第 197 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例返回每个雇员的工资以及在同一部门中工资最高的雇员的姓名：

```
SELECT GivenName + ' ' + Surname AS employee_name,
       Salary, DepartmentID,
       LAST_VALUE( employee_name ) OVER Salary_Window AS highest_paid
FROM Employees
WINDOW Salary_Window AS ( PARTITION BY DepartmentID ORDER BY Salary
                          RANGE BETWEEN UNBOUNDED PRECEDING
                          AND UNBOUNDED FOLLOWING );
```

employee_name	Salary	DepartmentID	highest_paid
Michael Lynch	24903	500	Jose Martinez
Joseph Barker	27290	500	Jose Martinez
Sheila Romero	27500	500	Jose Martinez
Felicia Kuo	28200	500	Jose Martinez
Jeannette Bertrand	29800	500	Jose Martinez
Jane Braun	34300	500	Jose Martinez

employee_name	Salary	DepartmentID	highest_paid
Anthony Rebeiro	34576	500	Jose Martinez
Charles Crowley	41700	500	Jose Martinez
Jose Martinez	55500.8	500	Jose Martinez
Doug Charlton	28300	400	Scott Evans
Elizabeth Lambert	29384	400	Scott Evans
Joyce Butterfield	34011	400	Scott Evans
Robert Nielsen	34889	400	Scott Evans
Alex Ahmed	34992	400	Scott Evans
Ruth Wetherby	35745	400	Scott Evans
...	...	...	...

Jose Martinez 在部门 500 中工资最高，Scott Evans 在部门 400 中工资最高。

LCASE 函数 [String]

将字符串中的所有字符转换成小写形式。此函数与 LOWER 函数完全相同。

语法

LCASE(*string-expression* **)**

参数

- **string-expression** 要转换成小写形式的字符串。

返回值

CHAR

NCHAR

LONG VARCHAR

VARCHAR

NVARCHAR

注释

LCASE 函数类似于 LOWER 函数。

另请参见

- “LOWER 函数 [String]”一节第 233 页
- “UCASE 函数 [String]”一节第 317 页
- “UPPER 函数 [String]”一节第 320 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 chocolate。

```
SELECT LCASE( 'ChoCOlatE' );
```

LEFT 函数 [String]

从字符串开始处返回多个字符。

语法

LEFT(*string-expression*, *integer-expression* **)**

参数

- **string-expression** 字符串。
- **integer-expression** 要返回的字符数。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

如果字符串包含多字节字符，并且使用了适当的归类，则返回的字节数可能大于指定的字符数。

可以指定比参数字符串表达式中的值大的 *integer-expression*。这种情况下将返回整个值。

此函数支持 NCHAR 输入和/或输出。如果输入字符串使用字符长度语义，就会在可能的情况下根据字符长度语义对返回值进行说明。

另请参见

- “RIGHT 函数 [String]”一节第 282 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 Customers 表中每个 Surname 值的前 5 个字符。

```
SELECT LEFT( Surname, 5) FROM Customers;
```

LENGTH 函数 [String]

返回指定字符串中的字符数。

语法

LENGTH(*string-expression* **)**

参数

- **string-expression** 字符串。

返回值

INT

注释

此函数用于确定字符串的长度。例如，为 *string-expression* 指定列名以确定该列中各值的长度。

如果字符串包含多字节字符，并且使用了适当的归类，则 LENGTH 返回字符数而不是字节数。如果字符串数据类型为 BINARY，则 LENGTH 函数的行为将与 BYTE_LENGTH 函数的行为相同。

> **注意**
> 对于 CHAR、VARCHAR 和 LONG VARCHAR 以及 NCHAR 数据类型，LENGTH 函数和 CHAR_LENGTH 函数可以互换使用。不过，对于 BINARY 和位数组数据类型，必须使用 LENGTH 函数。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “BYTE_LENGTH 函数 [String]”一节第 140 页
- “国际语言和字符集” 《SQL Anywhere 服务器 - 数据库管理》
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 9。

```
SELECT LENGTH( 'chocolate' );
```

LESSER 函数 [Miscellaneous]

返回两个参数值中的较小参数值。

语法

LESSER(*expression-1*, *expression-2* **)**

参数

- **expression-1** 要比较的第一个参数值。
- **expression-2** 要比较的第二个参数值。

返回值

此函数的返回类型取决于指定的表达式。即，数据库服务器计算此函数时，首先搜索可用于比较所有表达式的数据类型。如果找到通用类型，则数据库服务器比较表达式，然后以用于比较的类型返回结果。如果数据库服务器找不到通用比较类型，则返回错误。

注释

如果两个参数相等，则返回第一个值。

另请参见

- “GREATER 函数 [Miscellaneous]”一节第 204 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 5。

```
SELECT LESSER( 10, 5 ) FROM dummy;
```

LIST 函数 [Aggregate]

返回以逗号分隔的值列表。

语法

LIST(
[**DISTINCT**] *string-expression*
[, *delimiter-string*]
[**ORDER BY** *order-by-expression* [**ASC** | **DESC**], ...] **)**

参数

- **string-expression** 字符串表达式，通常是列名。对于列中的每一行，其值都会添加到一个逗号分隔的列表中。如果指定了 DISTINCT，则只添加唯一的值。

- **delimiter-string** 列表项的分隔字符串。缺省设置是逗号。如果设为 NULL 值或空字符串，则没有分隔符。*delimiter-string* 必须是常量。
- **order-by-expression** 对该函数返回的项进行排序。此参数之前没有逗号，以便在未提供 *delimiter-string* 的情况下使用。

  order-by-expression 不能是整数值。但可以是包含整数的变量。此外，同一查询块中的多个 LIST 函数不允许使用不同的 *order-by-expression* 参数。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

NULL 值不会添加到列表中。LIST (X) 返回一串组中每一行 X 的所有非 NULL 值（带分隔符）。如果组中没有任何一行具有明确的 X 值，则 LIST(X) 返回空字符串。

LIST 函数不能用作窗口函数，但可用作窗口函数的输入。

此函数支持 NCHAR 输入和/或输出。

标准和兼容性

- **SQL/2003** 服务商扩充。

另请参见

- “sa_split_list 系统过程”一节第 900 页

示例

以下语句返回值 487 Kennedy Court, 547 School Street。

```
SELECT LIST( Street ) FROM Employees
WHERE GivenName = 'Thomas';
```

以下语句列出雇员 ID。结果集中的每一行都包含某一部门雇员 ID 的逗号分隔列表。

```
SELECT LIST( EmployeeID )
FROM Employees
GROUP BY DepartmentID;
```

LIST(EmployeeID)
102,105,160,243,247,249,266,278,...
129,195,299,467,641,667,690,856,...
148,390,586,757,879,1293,1336,...
184,207,318,409,591,888,992,1062,...
191,703,750,868,921,1013,1570,...

以下语句对职员 ID 按职员的姓进行排序：

```
SELECT LIST( EmployeeID ORDER BY Surname ) AS "Sorted IDs"
FROM Employees
GROUP BY DepartmentID;
```

Sorted IDs '1751,591,1062,1191,992,888,318,184,1576,207,1684,1643,1607,1740,409,1507'

Sorted IDs
1013,191,750,921,868,1658,...
1751,591,1062,1191,992,888,318,...
1336,879,586,390,757,148,1483,...
1039,129,1142,195,667,1162,902,...
160,105,1250,247,266,249,445,...

以下语句返回以分号分隔的列表。请注意 ORDER BY 子句的位置和列表分隔符：

```
SELECT LIST( EmployeeID, ';' ORDER BY Surname ) AS "Sorted IDs"
FROM Employees
GROUP BY DepartmentID;
```

Sorted IDs
1013;191;750;921;868;1658;703;...
1751;591;1062;1191;992;888;318;...
1336;879;586;390;757;148;1483;...
1039;129;1142;195;667;1162;902; ...
160;105;1250;247;266;249;445;...

务必要将以下语句与前一个语句加以区分，以下语句返回以逗号分隔并按复合排序关键字 `( Surname, ';' )` 排序的雇员 ID 列表：

```
SELECT LIST( EmployeeID ORDER BY Surname, ';' ) AS "Sorted IDs"
FROM Employees
GROUP BY DepartmentID;
```

LOCATE 函数 [String]

返回一个字符串在另一个字符串中的位置。

语法

LOCATE(*string-expression-1*, *string-expression-2* [, *integer-expression*] **)**

参数

- **string-expression-1** 被搜索的字符串。
- **string-expression-2** 要搜索的字符串。此字符串的长度不应超过 255 个字节。
- **integer-expression** 字符串中开始进行搜索的字符位置。第一个字符在位置 1。如果起始偏移是负值，则定位函数返回最后一个匹配字符串偏移而非第一个。负的偏移指示从搜索中排除字符串尾的多长一部分。排除的字节数计算公式为 (-1 * 偏移) -1。

返回值

INT

注释

如果指定了 *integer-expression*，则从字符串中的该偏移处开始搜索。

第一个字符串可以是长字符串（长于 255 个字节），但第二个字符串的长度不能超过 255 个字节。如果第二个参数是长字符串，此函数返回 NULL 值。如果未找到字符串，则返回 0。搜索零长度字符串将返回 1。如果有某个参数为 NULL，则结果为 NULL。

如果使用多字节字符，并具有适当的归类，则开始位置和返回值可能不同于字节的位置。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “字符串函数”一节第 125 页
- “CHARINDEX 函数 [String]”一节第 144 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 8。

```
SELECT LOCATE(
   'office party this week - rsvp as soon as possible',
   'party',
   2 );
```

以下语句

```
BEGIN
   DECLARE STR LONG VARCHAR;
   DECLARE POS INT;
   SET str = 'c:\test\functions\locate.sql';
   SET pos = LOCATE( str, '\', -1 );
   select str, pos,
      SUBSTR( str, 1, pos -1 ) AS path,
      SUBSTR( str, pos +1 ) AS filename;
END;
```

返回以下输出：

str	pos	path	filename
c:\test\functions\locate.sql	18	c:\test\functions	locate.sql

LOG 函数 [Numeric]

返回一个数字的自然对数。

语法

LOG(*numeric-expression* **)**

参数

- **numeric-expression** 数字。

返回值

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。如果参数为 NULL，则结果为 NULL。

注释

参数是返回任何内置数字数据类型值的表达式。

另请参见

- “LOG10 函数 [Numeric]”一节第 232 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 50 的自然对数。

```
SELECT LOG( 50 );
```

LOG10 函数 [Numeric]

返回一个数字的以 10 为底的对数。

语法

LOG10(*numeric-expression* **)**

参数

- **numeric-expression** 数字。

返回值

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果参数为 NULL，则结果为 NULL。

注释

参数是返回任何内置数字数据类型值的表达式。

另请参见

- “LOG 函数 [Numeric]”一节第 232 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 50 的以 10 为底的对数。

```
SELECT LOG10( 50 );
```

LOWER 函数 [String]

将字符串中的所有字符转换成小写形式。此函数与 LCASE 函数完全相同。

语法

LOWER(*string-expression* **)**

参数

- **string-expression** 要转换的字符串。

返回值

CHAR

NCHAR

LONG VARCHAR

VARCHAR

NVARCHAR

注释

LCASE 函数与 LOWER 函数完全相同。

另请参见

- “LCASE 函数 [String]”一节第 225 页
- “UCASE 函数 [String]”一节第 317 页
- “UPPER 函数 [String]”一节第 320 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

以下语句返回值 chocolate。

```
SELECT LOWER( 'chOCOLate' );
```

LTRIM 函数 [String]

删除字符串中的前导和尾随空白。

语法

LTRIM(*string-expression* **)**

参数

- **string-expression** 要剪裁的字符串。

返回值

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

注释

结果的实际长度为表达式的长度减去删除的字符数。如果删除了所有字符，则结果为空字符串。

如果参数可以为空值，则结果也可以为空值。

如果参数为空值，则结果为空值。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “RTRIM 函数 [String]”一节第 286 页
- “TRIM 函数 [String]”一节第 315 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

  由 SQL/2003 标准定义的 TRIM 说明（LEADING 和 TRAILING）分别由 SQL Anywhere LTRIM 和 RTRIM 函数提供。

示例

以下语句返回值 Test Message，其中的所有前导空白都已删除。

```
SELECT LTRIM( '     Test Message' );
```

MAX 函数 [Aggregate]

返回在每一组行中找到的最大 *expression* 值。

语法 1

MAX(*expression* | **DISTINCT** *expression* **)**

语法 2

MAX(*expression* **) OVER (** *window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明 on below

参数

- **expression** 要计算最大值的表达式。这通常是列名。
- **DISTINCT expression** 返回的值与 MAX(*expression*) 相同，将其列出是为了保持完整性。

返回值

与参数相同的数据类型。

注释

忽略 *expression* 为 NULL 的行。对于不包含任何行的组返回 NULL。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “MIN 函数 [Aggregate]”一节第 236 页

标准和兼容性

- **SQL/2003** 核心特性。语法 2 为特性 T611。

示例

以下语句返回值 138948.000，表示 Employees 表中的最高薪水。

```
SELECT MAX( Salary )
FROM Employees;
```

MIN 函数 [Aggregate]

返回在每一组行中找到的最小表达式值。

语法 1

MIN(*expression* | **DISTINCT** *expression* **)**

语法 2

MIN(*expression* **) OVER (** *window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **expression** 要计算最小值的表达式。这通常是列名。
- **DISTINCT expression** 返回的值与 MIN(*expression*) 相同，将其列出是为了保持完整性。

返回值

与参数相同的数据类型。

注释

忽略 *expression* 为 NULL 的行。对于不包含任何行的组返回 NULL。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “MAX 函数 [Aggregate]”一节第 235 页

标准和兼容性

- **SQL/2003** 核心特性。语法 2 为特性 T611。

示例

以下语句返回值 24903.000，表示 Employees 表中的最低薪水。

```
SELECT MIN( Salary )
FROM Employees;
```

MINUTE 函数 [Date and time]

返回日期时间值的分钟部分。

语法

MINUTE(*datetime-expression* **)**

参数

- **datetime-expression** 日期时间值。

返回值

SMALLINT

注释

返回的值是 0 到 59 之间的某个数字，对应于日期时间的分钟。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 22。

```
SELECT MINUTE( '1998-07-13 12:22:34' );
```

MINUTES 函数 [Date and time]

所指定的信息会更改此函数的行为：

- 如果提供一个日期，此函数返回自 0000-02-29 以来的分钟数。

  注意
  0000-02-29 并不表示实际日期，它是日期算法使用的日期。

- 如果提供两个时间戳，此函数会返回它们之间的整分钟数。替代方法为使用 DATEDIFF 函数。
- 如果提供一个日期和一个整数，此函数会为指定时间戳添加整数分钟数。替代方法为使用 DATEADD 函数。

语法 1：integer

MINUTES([*datetime-expression*,] *datetime-expression* **)**

语法 2：timestamp

MINUTES(*datetime-expression*, *integer-expression* **)**

参数

- **datetime-expression** 日期和时间。
- **integer-expression** 要添加到 *datetime-expression* 中的分钟数。如果 *integer-expression* 是负数，则从日期时间值中减去相应的秒数。如果提供整数表达式，则必须将 *datetime-expression* 显式地转换为 DATETIME 数据类型。

返回值

INT

TIMESTAMP

注释

由于此函数返回整数，因此当使用语法 1 并且时间戳大于或等于 4083-03-23 02:08:00 时可能会发生溢出。

另请参见

- “CAST 函数 [Data type conversion]”一节第 141 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 240，表示第二个时间戳比第一个时间戳晚 240 分钟。建议使用第二个示例 (DATEDIFF)。

```
SELECT MINUTES( '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' );

SELECT DATEDIFF( minute,
     '1999-07-13 06:07:12',
     '1999-07-13 10:07:12' );
```

以下语句返回值 1051040527。

```
SELECT MINUTES( '1998-07-13 06:07:12' );
```

以下语句返回时间戳 12.05.99 21:10:070.000。建议使用第二个示例 (DATEADD)。

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5);

SELECT DATEADD( minute, 5, '1999-05-12 21:05:07' );
```


MOD 函数 [Numeric]

返回一个整数除以另一个整数之后产生的余数。

语法

MOD(*dividend*, *divisor* **)**

参数

- **dividend** 被除数，即除法的分子。
- **divisor** 除数，即除法的分母。

返回值

SMALLINT

INT

NUMERIC

注释

被除数为负数时结果为负数或零。除数的符号没有影响。

另请参见

- “REMAINDER 函数 [Numeric]”一节第 276 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性。

示例

以下语句返回值 2。

```
SELECT MOD( 5, 3 );
```

MONTH 函数 [Date and time]

返回给定日期的月份。

语法

MONTH(*date-expression* **)**

参数

- **date-expression** 日期时间值。

返回值

SMALLINT

注释

返回的值是 1 到 12 之间的某个数字，对应于日期时间的月份。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 7。

```
SELECT MONTH( '1998-07-13' );
```

MONTHNAME 函数 [Date and time]

返回日期中的月份名称。

语法

MONTHNAME(*date-expression* **)**

参数

- **date-expression** 日期时间值。

返回值

VARCHAR

注释

MONTHNAME 函数返回字符串，即使结果是数字类型（如表示二月的 2）。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 September。

```
SELECT MONTHNAME( '1998-09-05' );
```

MONTHS 函数 [Date and time]

所指定的信息会更改此函数的行为：

- 如果提供一个日期，此函数返回自 0000-02 以来的月份数。

> **注意**
> 0000-02 并不表示实际日期，它是日期算法使用的日期。

- 如果提供两个时间戳，此函数会返回它们之间的整月份数。替代方法为使用 DATEDIFF 函数。
- 如果提供一个日期和一个整数，此函数会为指定时间戳添加整数分钟数。替代方法为使用 DATEADD 函数。

语法 1：integer

MONTHS([*datetime-expression*,] *datetime-expression* **)**

语法 2：timestamp

MONTHS(*datetime-expression*, *integer-expression* **)**

参数

- **datetime-expression** 日期和时间。
- **integer-expression** 要添加到 *datetime-expression* 中的月份数。如果 *integer-expression* 是负数，则从日期时间值中减去相应的月数。如果提供 *integer-expression*，则必须将 *datetime-expression* 显式地转换为日期时间数据类型。

  有关转换数据类型的信息，请参见“CAST 函数 [Data type conversion]”一节第 141 页。

返回值

INT

TIMESTAMP

注释

MONTHS 的值是根据两个日期之间每个月的 1 号的数目计算出的。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 2，表示第二个日期比第一个日期晚两个月。建议使用第二个示例 (DATEDIFF)。

```
SELECT MONTHS( '1999-07-13 06:07:12',
   '1999-09-13 10:07:12' );

SELECT DATEDIFF( month,
   '1999-07-13 06:07:12',
   '1999-09-13 10:07:12' );
```

以下语句返回值 23981。

```
SELECT MONTHS( '1998-07-13 06:07:12' );
```

以下语句返回时间戳 1999-10-12 21:05:070.000。建议使用第二个示例 (DATEADD)。

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5);

SELECT DATEADD( month, 5, '1999-05-12 21:05:07' );
```

NCHAR 函数 [String]

返回一个 NCHAR 字符串，其中包含一个在参数中指定了 Unicode 代码点的字符；如果该值不是有效的代码点值，则返回 NULL。

语法

NCHAR(*integer* **)**

参数

- **integer** 要转换为相应 Unicode 代码点的数字。

返回值

NVARCHAR

另请参见

- “CONNECTION_EXTENDED_PROPERTY 函数 [String]”一节第 149 页
- “TO_NCHAR 函数 [String]”一节第 312 页
- “TO_CHAR 函数 [String]”一节第 311 页
- “UNICODE 函数 [String]”一节第 318 页
- “UNISTR 函数 [String]”一节第 319 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例返回阿拉伯字母 ALEF，它是 Unicode 代码点 U+627：

```
SELECT NCHAR( 1575 );
```

NEWID 函数 [Miscellaneous]

生成 UUID（通用唯一标识符）值。UUID 与 GUID（全局唯一标识符）相同。

语法

NEWID( )

参数

没有任何与 NEWID 函数关联的参数。

返回值

UNIQUEIDENTIFIER

注释

NEWID 函数可以在列的 DEFAULT 子句中使用。

UUID 可用于唯一地标识表中的行。在一台计算机上生成的值与在另一台计算机上生成的值不匹配，因此，在同步和复制环境中可将这些值作为关键字来使用。

UUID 包含连字符，以便与其它 RDBMS 兼容。

NEWID 函数是非确定型函数，因此对其进行连续调用可能会返回不同的值。查询优化程序不会对 NEWID 函数的结果进行高速缓存。

有关非确定型函数的详细信息，请参见“函数高速缓存”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “NEWID 缺省值”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “STRTOUUID 函数 [String]”一节第 304 页
- “UUIDTOSTR 函数 [String]”一节第 322 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句创建一个包含两列、名为 mytab 的表。列 pk 的数据类型是唯一标识符，它指派 NEWID 函数作为缺省值。列 c1 的数据类型是整型。

```
CREATE TABLE mytab(
   pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
   c1 INT );
```

以下语句返回字符串形式的唯一标识符：

```
SELECT NEWID();
```

例如，返回的值可能是 96603324-6FF6-49DE-BF7D-F44C1C7E6856。

NEXT_CONNECTION 函数 [System]

返回下一个连接的标识号。

语法

NEXT_CONNECTION([*connection-id*] [, *database-id*] **)**

返回值

INT

参数

- **connection-id** 一个整数，通常是从上一个 NEXT_CONNECTION 调用返回。如果 *connection-id* 为 NULL，则 NEXT_CONNECTION 返回最近的连接 ID。
- **database-id** 表示当前服务器上其中一个数据库的整数。如果不提供 *database-id*，将使用当前数据库。如果提供 NULL，则 NEXT_CONNECTION 返回下一个连接，不考虑数据库。

注释

NEXT_CONNECTION 可用于枚举到数据库的连接。连接 ID 通常以单调递增顺序创建。此函数以相反顺序返回下一连接 ID。

要获取最近连接的连接 ID 值，请输入 NULL 作为 *connection-id*。要获取随后的连接，请输入上一返回值。不再有依循该顺序的连接时，此函数返回 NULL。

如果想要断开特定时间之前创建的所有连接，NEXT_CONNECTION 会有用处。不过，因为 NEXT_CONNECTION 是以相反顺序返回连接 ID，所以不会返回在启动函数后建立的连接。如果想要确保所有连接都断开，请避免在运行 NEXT_CONNECTION 前创建新连接。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回当前数据库上第一个连接的标识符。该标识符是一个类似 10 的整数值。

```
SELECT NEXT_CONNECTION( NULL );
```

以下语句返回类似 5 的值。

```
SELECT NEXT_CONNECTION( 10 );
```

以下调用从当前数据库的指定 *connection-id* 开始，以相反顺序返回下一连接 ID。

```
SELECT NEXT_CONNECTION( connection-id );
```

以下调用从指定 *connection-id* 开始（不考虑数据库），以相反顺序返回下一连接 ID。

```
SELECT NEXT_CONNECTION( connection-id, NULL );
```

以下调用从指定数据库的指定 *connection-id* 开始，以相反顺序返回下一连接 ID。

```
SELECT NEXT_CONNECTION( connection-id, database-id );
```

以下调用返回第一个（最早的）连接（不考虑数据库）。

```
SELECT NEXT_CONNECTION( NULL, NULL );
```

以下调用返回指定数据库上的第一个（最早的）连接。

```
SELECT NEXT_CONNECTION( NULL, database-id );
```

NEXT_DATABASE 函数 [System]

返回数据库的标识号。

语法

NEXT_DATABASE({ **NULL** | *database-id* } **)**

参数

- **database-id** 一个整数，指定数据库的 ID 号。

返回值

INT

注释

NEXT_DATABASE 函数用于枚举数据库服务器上运行的数据库。若要获取第一个数据库，请传递 NULL；若要获得每个后续数据库，请传递前一个返回值。当没有其它的数据库时，此函数返回 NULL。数据库 ID 号不是按特定顺序返回，但通过数据库 ID 可以得知建立到服务器的连接的顺序。将为第一个连接到服务器的数据库指派值 0，随后连接到服务器的数据库将以值 1 为增量为其指派数据库 ID。

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下语句返回值 0，这是第一个数据库值。

```
SELECT NEXT_DATABASE( NULL );
```

以下语句返回 NULL，表示服务器上不再有数据库。

```
SELECT NEXT_DATABASE( 0 );
```

NEXT_HTTP_HEADER 函数 [HTTP]

获得下一个 HTTP 标头名称。

语法

NEXT_HTTP_HEADER(*header-name* **)**

参数

header-name 上一个标头的名称。如果 header-name 为 NULL，则此函数返回第一个 HTTP 标头的名称。

返回值

LONG VARCHAR

注释

此函数对请求内包含的 HTTP 标头进行迭代并返回下一个 HTTP 标头名。如果用 NULL 调用该函数，该函数会返回第一个标头的名称。通过向该函数传递上一个标头的名称来检索后续的标头。使用上一标头的名称调用此函数或不是从 Web 服务进行调用时，此函数返回 NULL。

重复调用该函数将会返回所有的头字段且仅返回一次，但不一定与其出现在 HTTP 请求中的顺序相同。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “处理 HTTP 标头”一节 《SQL Anywhere 服务器 - 编程》
- “HTTP_DECODE 函数 [HTTP]”一节第 211 页
- “HTTP_ENCODE 函数 [HTTP]”一节第 212 页
- “HTTP_HEADER 函数 [HTTP]”一节第 214 页
- “HTTP_VARIABLE 函数 [HTTP]”一节第 216 页
- “NEXT_HTTP_VARIABLE 函数 [HTTP]”一节第 246 页
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页
- “处理 HTTP 标头”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在由 HTTP Web 服务所调用的存储过程内，使用以下示例会返回第一个 HTTP 标头的名称。

```
BEGIN
DECLARE header_name LONG VARCHAR;
SET header_name  = NULL;
SET header_name = NEXT_HTTP_HEADER( header_name )
END;
```

NEXT_HTTP_VARIABLE 函数 [HTTP]

获得下一个 HTTP 变量的名称。

语法

NEXT_HTTP_VARIABLE(*var-name* **)**

参数

- **var-name** 上一个变量的名称。如果 *var-name* 为 NULL，则此函数返回第一个 HTTP 变量的名称。

返回值

LONG VARCHAR

注释

此函数对请求内包含的 HTTP 变量进行迭代。当使用 NULL 调用时，该函数返回第一个变量的名称。使用上一个变量名称调用该函数则可检索到后续的变量名称。使用最后一个变量的名称调用此函数或不是从 Web 服务进行调用时，此函数返回 NULL。

重复调用此函数时将会返回所有变量且仅返回一次，但返回的顺序不一定是它们在 HTTP 请求中出现的顺序。如果 URL PATH 分别设置为 ON 或 ELEMENTS，则包括变量 url 或 url1、url2、……、url10。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “处理变量”一节 《SQL Anywhere 服务器 - 编程》
- “HTTP_DECODE 函数 [HTTP]”一节第 211 页
- “HTTP_ENCODE 函数 [HTTP]”一节第 212 页
- “HTTP_HEADER 函数 [HTTP]”一节第 214 页
- “HTTP_VARIABLE 函数 [HTTP]”一节第 216 页
- “NEXT_HTTP_HEADER 函数 [HTTP]”一节第 245 页
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在由 HTTP Web 服务所调用的存储过程内，使用以下示例会返回第一个 HTTP 变量的名称。

```
BEGIN
DECLARE variable_name LONG VARCHAR;
SET variable_name = NULL;
SET variable_name = NEXT_HTTP_VARIABLE( variable_name )
END;
```

NEXT_SOAP_HEADER 函数 [SOAP]

返回 SOAP 请求标头中的下一个标头键。

语法

NEXT_SOAP_HEADER(*header-key* **)**

参数

header-key 给定标头条目顶级 XML 元素的 XML 本地名。

返回值

LONG VARCHAR

注释

如果为 *header-key* 指定 NULL，则此函数返回在 SOAP 标头中找到的第一个标头条目的标头键。

如果使用上一 *header-key* 进行调用，则此函数返回 NULL。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “处理 SOAP 标头”一节 《SQL Anywhere 服务器 - 编程》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “SOAP_HEADER 函数 [SOAP]”一节第 292 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在由 HTTP Web 服务所调用的存储过程内，使用以下语句会返回在 SOAP 标头中找到的第一个标头关键字。

```
SET header_key = NEXT_SOAP_HEADER( NULL );
```

NOW 函数 [Date and time]

返回当前的年、月、日、小时、分钟、秒和秒的小数部分。其准确性受系统时钟准确性的限制。

语法

NOW(*)

返回值

TIMESTAMP

注释

NOW 函数返回的信息与 GETDATE 函数返回的信息及 CURRENT TIMESTAMP 特殊值相同。

另请参见

- “GETDATE 函数 [Date and time]”一节第 202 页
- “CURRENT TIMESTAMP 特殊值”一节第 57 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回当前日期和时间。

```
SELECT NOW( * );
```


NULLIF 函数 [Miscellaneous]

通过比较表达式提供缩写的 CASE 表达式。

语法

NULLIF(*expression-1*, *expression-2* **)**

参数

- **expression-1** 要比较的表达式。
- **expression-2** 要比较的表达式。

返回值

第一个参数的数据类型。

注释

NULLIF 比较两个表达式的值。

如果第一个表达式的值与第二个表达式的值相等，NULLIF 返回 NULL。

如果第一个表达式的值不等于第二个表达式的值，或者第二个表达式为 NULL，则 NULLIF 返回第一个表达式。

NULLIF 函数提供了编写某些 CASE 表达式的简便方法。

另请参见

- “CASE 表达式”一节第 18 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

以下语句返回值 a：

```
SELECT NULLIF( 'a', 'b' );
```

下面的语句返回 NULL。

```
SELECT NULLIF( 'a', 'a' );
```

NUMBER 函数 [Miscellaneous]

为查询结果中的每个连续行生成从 1 开始的编号。NUMBER 函数主要供在选择列表中使用。

由于受到 NUMBER 函数的限制（下面的注释部分中有说明），请改为使用“ROW_NUMBER 函数 [Miscellaneous]”一节第 284 页。ROW_NUMBER 函数提供了相同的功能，却没有 NUMBER 函数的限制。

语法

NUMBER(*)

返回值

INT

注释

可以在选择列表中使用 NUMBER(*)，以为结果集中的行提供顺序编号。NUMBER(*) 返回每个结果行的 ANSI 行号值。这意味着 NUMBER 函数返回的值可能是正数或负数，具体取决于应用程序在结果集中滚动的方式。对于不敏感游标，NUMBER(*) 的值总是正数，因为整个结果集是在 OPEN 状态下实现的。

此外，行号可能会受某些游标类型更改的影响。对于不敏感游标和滚动游标，该值是固定的。如果存在并发更新，则该值对于动态游标和敏感游标可能会发生变化。

在以下位置使用 NUMBER 函数会生成语法错误：DELETE 语句、WHERE 子句、HAVING 子句、ORDER BY 子句、子查询、涉及集合的查询、任何约束、GROUP BY 子句、DISTINCT 子句、查询表达式（UNION、EXCEPT、INTERSECT）或派生表。

NUMBER(*) 可在视图中使用（受上述限制的约束），但在查询或外部视图中，对应于涉及 NUMBER(*) 的表达式的视图列最多只能被引用一次，而且视图在左外连接或完全外连接中不能以提供 NULL 的表的形式出现。

在嵌入式 SQL 中，当您使用的游标引用包含 NUMBER(*) 函数的查询时，应小心谨慎。具体地讲，当使用相对于游标结尾处的位置（偏移为负的绝对位置）来定位数据库游标时，此函数返回负数。

在 UPDATE 语句的 SET 子句中，可以在赋值符号的右侧使用 NUMBER。例如，`SET x = NUMBER(*)`。

从 SELECT 语句中使用 INSERT（请参见“INSERT 语句”一节第 616 页）时，还可以使用 NUMBER 函数生成主键，尽管生成顺序主键的首选机制是使用 AUTOINCREMENT 子句。

有关 AUTOINCREMENT 子句的信息，请参见“CREATE TABLE 语句”一节第 497 页。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回按顺序排列的部门列表。

```
SELECT NUMBER( * ), DepartmentName
FROM Departments
WHERE DepartmentID > 5
ORDER BY DepartmentName;
```


SQL 函数 (P-Z)

列出了每个函数，并在它旁边指出函数类型（数字、字符等等）。

有关指向给定类型所有函数的链接，请参见“函数类型”一节第 118 页。

另请参见

- “SQL 函数 (A-D)”一节第 129 页
- “SQL 函数 (E-O)”一节第 185 页

PATINDEX 函数 [String]

返回一个整数，该整数表示模式第一次在字符串中出现的起始位置。

语法

PATINDEX('%*pattern***%'**, *string-expression* **)**

参数

- **pattern**　要搜索的模式。如果省略前导百分号通配符，则模式出现在字符串起始处时，PATINDEX 函数返回一 (1)，否则返回零。

  模式与 LIKE 比较运算使用相同的通配符。这些通配符如下：

通配符	匹配项
_（下划线）	任意一个字符
%（百分号）	包含零个或多个字符的任意字符串
[]	指定范围或集合中的任何单个字符
[^]	不在指定范围或集合中的任何单个字符

- **string-expression**　要在其中搜索模式的字符串。

返回值

INT

注释

PATINDEX 函数返回模式第一次出现的开始位置。如果未找到模式，则返回零 (0)。

另请参见

- “LIKE 搜索条件”一节第 38 页
- “LOCATE 函数 [String]”一节第 230 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 2。

```
SELECT PATINDEX( '%hoco%', 'chocolate' );
```

以下语句返回值 11。

```
SELECT PATINDEX( '%4_5_', '0a1A 2a3A 4a5A' );
```

以下语句返回 14，即字符串表达式中第一个非字母数字字符。注意，如果数据库区分大小写，则可使用模式 '%[^a-z0-9]%' 代替 '%[^a-zA-Z0-9]%'。

```
SELECT PATINDEX( '%[^a-zA-Z0-9]%', 'SQLAnywhere11 has many new features' );
```

要获得字符串中的第一个字母数字单词，可以通过类似下面的方式：

```
SELECT LEFT( @string, PATINDEX( '%[^a-zA-Z0-9]%', @string ) );
```

PERCENT_RANK 函数 [Ranking]

对于任意的行 X（由函数的参数和 ORDER BY 说明来定义），PERCENT_RANK 函数计算其排位的方法是：用行 X 减 1 除以组中的行数。

语法

PERCENT_RANK( ) OVER (*window-spec* **)**

window-spec：请参见下面的“注释”部分

返回值

PERCENT_RANK 函数返回介于 0 和 1 之间的 DOUBLE 值。

注释

可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。作为窗口函数使用时，必须指定 ORDER BY 子句，还可以指定 PARTITION BY 子句，但不能指定 ROWS 或 RANGE 子句。请参见“WINDOW 子句”一节 第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “CUME_DIST 函数 [Ranking]”一节第 162 页
- “DENSE_RANK 函数 [Ranking]”一节第 181 页
- “RANK 函数 [Ranking]”一节第 261 页

标准和兼容性

- **SQL/2003** SQL/OLAP 特性 T612。

示例

以下示例返回按性别显示 New York 雇员薪水排位的结果集。这些结果按降序列出排位并按性别分区。

```
SELECT DepartmentID, Surname, Salary, Sex,
PERCENT_RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) "Rank"
FROM Employees
WHERE State IN ('NY');
```

DepartmentID	Surname	Salary	Sex	Rank
200	Martel	55700.000	M	0
100	Guevara	42998.000	M	0.333333333
100	Soo	39075.000	M	0.666666667
400	Ahmed	34992.000	M	1
300	Davidson	57090.000	F	0
400	Blaikie	54900.000	F	0.333333333
100	Whitney	45700.000	F	0.666666667
400	Wetherby	35745.000	F	1

PI 函数 [Numeric]

返回数字值 PI。

语法

PI(*)

返回值

DOUBLE

标准和兼容性

- **SQL/2003** 服务商扩充。

注释

此函数返回一个 DOUBLE 值。

示例

以下语句返回值 3.141592653……

```
SELECT PI( * );
```

PLAN 函数 [Miscellaneous]

以字符串形式返回 SQL 语句的详细计划优化策略。

语法

PLAN(*string-expression*, [*cursor-type*], [*update-status*] **)**

参数

- **string-expression** SQL 语句，通常是 SELECT 语句，但也可以是 UPDATE 或 DELETE 语句。
- **cursor-type** 一个字符串。*cursor-type* 可以是 asensitive（缺省类型）、insensitive、sensitive 或 keyset-driven。
- **update-status** 字符串参数，它采用下列值之一来指示优化程序如何处理给定的游标：

值	说明
READ-ONLY	游标是只读的。
READ-WRITE（缺省值）	可以读取或写入游标。
FOR UPDATE	可以读取或写入游标。这与 READ-WRITE 完全相同。

返回值

LONG VARCHAR

另请参见

- “EXPLANATION 函数 [Miscellaneous]”一节第 195 页
- “GRAPHICAL_PLAN 函数 [Miscellaneous]”一节第 202 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句将 SELECT 语句作为字符串参数传递，并返回查询的执行计划。

```
SELECT PLAN(
   'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

此信息有助于决定要添加的索引或数据库的构建方法，以便获得更好的性能。

以下语句返回一个字符串，其中包含 SELECT * FROM Departments WHERE DepartmentID > 100; 查询中 INSENSITIVE 游标的文本计划。

```
SELECT PLAN(
   'SELECT * FROM Departments WHERE DepartmentID > 100',
   'insensitive',
   'read-only' );
```

POWER 函数 [Numeric]

以一个数字为底数另一个数字为指数计算乘方值。

语法

POWER(*numeric-expression-1*, *numeric-expression-2* **)**

参数

- **numeric-expression-1** 底数。
- **numeric-expression-2** 指数。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果任何参数为 NULL，则结果为 NULL 值。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 64。

```
SELECT POWER( 2, 6 );
```

PROPERTY 函数 [System]

以字符串形式返回指定的数据库服务器属性的值。

语法

PROPERTY({ *property-id* | *property-name* } **)**

参数

- **property-id** 一个表示数据库服务器属性的属性号的整数。通过 PROPERTY_NUMBER 函数可以确定此数字。*property-id* 通常是在一组属性中循环时使用。

- **property-name** 提供数据库属性的名称的字符串。

返回值

VARCHAR

注释

每个属性都有编号和名称，但编号会因发行版本的不同而有变化，因此不应将其用作给定属性的可靠标识符。

另请参见

- “数据库服务器属性”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回当前数据库服务器的名称：

```
SELECT PROPERTY( 'Name' );
```

PROPERTY_DESCRIPTION 函数 [System]

返回有关属性的说明。

语法

PROPERTY_DESCRIPTION({ *property-id* | *property-name* } **)**

参数

- **property-id** 一个表示数据库属性的属性号的整数。通过 PROPERTY_NUMBER 函数可以确定此数字。*property-id* 通常是在一组属性中循环时使用。
- **property-name** 提供数据库属性的名称的字符串。

返回值

VARCHAR

注释

每个属性都有编号和名称，但编号会因发行版本的不同而有变化，因此不应将其用作给定属性的可靠标识符。

另请参见

- “连接属性、数据库属性和数据库服务器属性” 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回索引插入的说明性编号。

```
SELECT PROPERTY_DESCRIPTION( 'IndAdd' );
```

PROPERTY_NAME 函数 [System]

通过提供指定连接级别的属性 ID，返回该属性的名称。

语法

PROPERTY_NAME(*property-id* [, *property-scope*] **)**

property-scope:
NULL
| **'server'**
| **'database'**
| **'db'**
| **'connection'**
| **'conn'**

参数

- **property-id** 数据库属性的属性 ID。
- **property-scope** 属性的范围，或者 NULL。

返回值

VARCHAR

另请参见

- “连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “数据库服务器属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回与属性 ID 102 关联的服务器级别属性。

```
SELECT PROPERTY_NAME( 102, 'server' );
```

PROPERTY_NUMBER 函数 [System]

返回具有所提供属性名称的属性的属性号。

语法

PROPERTY_NUMBER(*property-name* **)**

参数

- **property-name** 属性名称。

返回值

INT

注释

每个属性都有编号和名称，但编号会因发行版本的不同而有变化，因此不应将其用作给定属性的可靠标识符。如果属性编号或属性名都可以使用，使用属性名更为可取。请始终使用 PROPERTY_NUMBER 函数来确保对于正在使用的服务器，属性编号是最新的。

另请参见

- “连接属性、数据库属性和数据库服务器属性” 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回一个整数值。实际值因发行版本而异。

```
SELECT PROPERTY_NUMBER( 'PAGESIZE' );
```

QUARTER 函数 [Date and time]

从提供的日期表达式中返回一个表示某季度的数字。

语法

QUARTER(*date-expression* **)**

参数

- **date-expression** 日期。

返回值

INT

注释

有如下这些季度：

Quarter	期限（含起止日期）
1	1 月 1 日到 3 月 31 日
2	4 月 1 日到 6 月 30 日
3	7 月 1 日到 9 月 30 日
4	10 月 1 日到 12 月 31 日

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 2。

```
SELECT QUARTER( '1987/05/02' );
```

RADIANS 函数 [Numeric]

将数字由角度转换成弧度。

语法

RADIANS(*numeric-expression* **)**

参数

- **numeric-expression** 数字（以度为单位）。此角度将转换成弧度。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回近似值 0.5236。

```
SELECT RADIANS( 30 );
```

RAND 函数 [Numeric]

返回 0 到 1 之间的一个随机数（带可选的种子值）。

语法

RAND([*integer-expression*] **)**

参数

- **integer-expression** 用于创建随机数的可选种子。此参数用于创建可重复的随机数序列。

返回值

DOUBLE

注释

RAND 函数是一个倍增线性同余随机数发生器。请参见 Park and Miller (1988) 出版的《CACM 31(10)》1192-1201 页及 Press et al.(1992) 出版的《Numerical Recipes in C》（第 2 版，第 7 章，第 279 页）。279). 调用 RAND 函数的结果是生成一个伪随机数 *n*，其中 $0 < n < 1$（0.0 或 1.0 都不能作为结果）。

建立到服务器的连接时，随机数生成器会将初始值作为种子。将唯一地为每个连接提供种子，从而使每个连接看到的随机序列都与其它连接所看到的不同。还可以指定种子值 (*integer-expression*) 作为参数。通常，只应在通过对 RAND 函数进行连续调用来请求随机数序列之前执行一次此操作。如果多次初始化种子值，将会重新启动序列。如果指定同一种子值，将会生成同一序列。值相近的种子值会生成相似的初始序列，序列中的发散度会进一步外扩。

千万不要将从某个种子值生成的序列与从另一个种子值生成的序列合并，以尝试获得统计随机结果。换言之，不要在随机值的序列生成期间重设种子值。

RAND 函数被视为非确定型函数。查询优化程序不会对 RAND 函数的结果进行高速缓存。

有关非确定型函数的详细信息，请参见“函数高速缓存”一节 《SQL Anywhere 服务器 - SQL 的用法》。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句产生十一个随机结果。对 RAND 函数的每个后续调用（不指定种子）都会继续生成不同的结果：

```
SELECT RAND( 1 );
SELECT RAND( ), RAND( ), RAND( ), RAND( ), RAND( );
SELECT RAND( ), RAND( ), RAND( ), RAND( ), RAND( );
```

以下示例会产生两组具有相同序列的结果，因为种子值指定了两次：

```
SELECT RAND( 1 ), RAND( ), RAND( ), RAND( ), RAND( );
SELECT RAND( 1 ), RAND( ), RAND( ), RAND( ), RAND( );
```


以下示例产生五个值相近的结果，这些结果在分布上并不是随机的。由于这个原因，不建议使用相似的种子值多次调用 RAND 函数：

```
SELECT RAND( 1 ), RAND( 2 ), RAND( 3 ), RAND( 4 ), RAND( 5 );
```

以下示例会产生五个完全相同的结果，应该加以避免：

```
SELECT RAND( 1 ), RAND( 1 ), RAND( 1 ), RAND( 1 ), RAND( 1 );
```

RANK 函数 [Ranking]

计算一个值在一组值中的排位。如果出现并列的情况，RANK 函数会在排名序列中留出空位。

语法

RANK( ) OVER (*window-spec* **)**

window-spec：请参见下面的“注释”部分

返回值

INTEGER

注释

可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。作为窗口函数使用时，必须指定 ORDER BY 子句，还可以指定 PARTITION BY 子句，但不能指定 ROWS 或 RANGE 子句。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “CUME_DIST 函数 [Ranking]”一节第 162 页
- “DENSE_RANK 函数 [Ranking]”一节第 181 页
- “ROW_NUMBER 函数 [Miscellaneous]”一节第 284 页
- “PERCENT_RANK 函数 [Ranking]”一节第 252 页

标准和兼容性

- **SQL/2003** SQL/OLAP 特性 T612。

示例

以下示例提供了 Utah 和 New York 州雇员薪水的降序排位。注意到第 7 名职员与第 8 名具有相同的薪水因而并列第 7 位。接下来的职员的排位为第 9 位而在排位序列中留下了空位（没有第 8 位排位）。

```
SELECT Surname, Salary, State,
RANK() OVER (ORDER BY Salary DESC) "Rank"
FROM Employees WHERE State IN ('NY','UT');
```

Surname	Salary	State	Rank
Shishov	72995.000	UT	1
Wang	68400.000	UT	2
Cobb	62000.000	UT	3
Morris	61300.000	UT	4
Davidson	57090.000	NY	5
Martel	55700.000	NY	6
Blaikie	54900.000	NY	7
Diaz	54900.000	NY	7
Driscoll	48023.690	UT	9
Hildebrand	45829.000	UT	10
Whitney	45700.000	NY	11
...	...	...	...
Lynch	24903.000	UT	19

READ_CLIENT_FILE 函数 [String]

由客户端计算机的指定文件中读取数据。

语法

READ_CLIENT_FILE(*client-filename-expression* **)**

参数

- **client-filename-expression** 表示客户端计算机上文件名的 CHAR 值。将相对于客户端应用程序的当前工作目录来解析此路径。

返回值

LONG BINARY

注释

READ_CLIENT_FILE 函数所返回的值表示指定客户端文件的内容。只要允许使用 BINARY 表达式，您就可以在语法中使用该函数。

因为数据以二进制字符串形式返回，所以如果数据使用了其它字符集，或者经过压缩或加密，您也可能会需要对其进行字符集转换、解压或解密。

在 READ_CLIENT_FILE 求值期间，数据库服务器会发出从客户端传输指定文件的请求。客户端接收到传输请求后，即会获取客户端文件的共享锁，并且在数据库服务器请求客户端中止该请求之前会一直持有该锁。

文件的读取由客户端软件库执行，数据的传输则使用命令序列通信协议来完成。

权限

读取客户端计算机上的文件时：

- 需要 READCLIENTFILE 权限。请参见“READCLIENTFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
- 需要具有对从中读取的目录的读取权限。
- 必须启用 allow_read_client_file 数据库选项。请参见“allow_read_client_file 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- 必须启用 read_client_file 受保护的功能。请参见“-sf 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

标准和兼容性

- **SQL/2003** 服务商扩充。

另请参见

- “访问客户端计算机上的数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “READCLIENTFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》
- “DECOMPRESS 函数 [String]”一节第 177 页
- “DECRYPT 函数 [String]”一节第 179 页
- “CSCONVERT 函数 [String]”一节第 161 页

REGEXP_SUBSTR 函数 [String]

使用正则表达式从字符串中抽取子串。

语法

REGEXP_SUBSTR(*expression*,
regular-expression
[, *start-offset* [, *occurrence-number* [, *escape-expression*]]])

参数

- **expression** 被搜索的字符串。
- **regular-expression** 尝试匹配的模式。有关正则表达式语法的详细信息，请参见“正则表达式概述”一节第 20 页。

- **start-offset** 开始搜索时相对于 *expression* 的偏移。*start-offset* 以正整数表示，表示从字符串的左端数的字符数。缺省值为 1（字符串的起点）。
- **occurrence-number** 当 *expression* 中有多个匹配项时，指定一个整数来表示要定位在第几个出现的匹配项。例如，3 表示查找第三个出现的匹配项。缺省值为 1。
- **escape-expression** *regular-expression* 所使用的转义字符。缺省为反斜线字符 (\)。

返回值

LONG VARCHAR

注释

如果未找到 *regular-expression*，则 REGEXP_SUBSTR 返回 NULL。

与 REGEXP 搜索条件类似，REGEXP_SUBSTR 函数使用代码点进行匹配和范围评估。这意味着数据库区分大小写并不影响结果。有关 REGEXP_SUBSTR 如何执行匹配和集评估的详细信息，请参见“LIKE、REGEXP 和 SIMILAR TO：字符比较上的差异”一节第 37 页。

当针对只包含一个子字符类的字符类进行匹配时，请将外层方括号和子字符类的方括号一同包括进去（例如，`REGEXP_SUBSTR (expression, '[[:digit:]]')`）。有关子字符类匹配的详细信息，请参见“正则表达式：特殊子字符类”一节第 23 页。

另请参见

- “正则表达式语法”一节第 20 页
- “REGEXP 搜索条件”一节第 43 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例将 Employees.Street 列中的值分为 street number 和 street name 两部分：

```
SELECT REGEXP_SUBSTR( Street, '^\S+' ) as street_num,
 REGEXP_SUBSTR( Street, '(?<=^\S+\s+).*$' ) AS street_name
    FROM Employees;
```

street_num	street_name
9	East Washington Street
7	Pleasant Street
539	Pond Street
1244	Great Plain Avenue
...	...

要确定当前连接的 IP 地址是否在 IP 地址的范围之内（在本例中，为 10.25.101.xxx 或 10.25.102.xxx），可以执行以下语句：

```
IF REGEXP_SUBSTR( CONNECTION_PROPERTY( 'NodeAddress' ), '\\d+\\.\\d+\\.\\d
+' )
   IN ( '10.25.101' , '10.25.102' ) THEN
      MESSAGE 'In range' TO CLIENT;
ELSE
      MESSAGE 'Out of range' TO CLIENT;
END IF;
```

REGR_AVGX 函数 [Aggregate]

计算回归线的独立变量的平均值。

语法 1

REGR_AVGX(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_AVGX(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。在一次处理数据期间同时计算该函数。排除 NULL 值后，将进行以下计算，其中的 *x* 表示 *independent-expression*：

```
AVG( x )
```

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “AVG 函数 [Aggregate]”一节第 134 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_INTERCEPT 函数 [Aggregate]”一节第 268 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_SLOPE 函数 [Aggregate]”一节第 271 页
- “REGR_SXX 函数 [Aggregate]”一节第 272 页
- “REGR_SXY 函数 [Aggregate]”一节第 274 页
- “REGR_SYY 函数 [Aggregate]”一节第 275 页
- “REGR_AVGY 函数 [Aggregate]”一节第 266 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例计算非独立变量（雇员年龄）的平均值。

```
SELECT REGR_AVGX( Salary, ( 2008 - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_AVGY 函数 [Aggregate]

计算回归线的非独立变量的平均值。

语法 1

REGR_AVGY(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_AVGY(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。在一次处理数据期间同时计算该函数。排除 NULL 值后，将进行以下计算，其中的 *y* 表示 *dependent-expression*：

```
AVG( y )
```

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_INTERCEPT 函数 [Aggregate]”一节第 268 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_SLOPE 函数 [Aggregate]”一节第 271 页
- “REGR_SXX 函数 [Aggregate]”一节第 272 页
- “REGR_SXY 函数 [Aggregate]”一节第 274 页
- “REGR_SYY 函数 [Aggregate]”一节第 275 页
- “REGR_AVGX 函数 [Aggregate]”一节第 265 页
- “AVG 函数 [Aggregate]”一节第 134 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例计算独立变量（雇员薪水）的平均值。

```
SELECT REGR_AVGY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_COUNT 函数 [Aggregate]

返回一个整数，表示用来拟合回归线的非 NULL 数对的数量。

语法 1

REGR_COUNT(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_COUNT(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

INTEGER

注释

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “REGR_INTERCEPT 函数 [Aggregate]”一节第 268 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_SLOPE 函数 [Aggregate]”一节第 271 页
- “REGR_SXX 函数 [Aggregate]”一节第 272 页
- “REGR_SXY 函数 [Aggregate]”一节第 274 页
- “REGR_SYY 函数 [Aggregate]”一节第 275 页
- “REGR_AVGY 函数 [Aggregate]”一节第 266 页
- “REGR_AVGX 函数 [Aggregate]”一节第 265 页
- “COUNT 函数 [Aggregate]”一节第 157 页
- “AVG 函数 [Aggregate]”一节第 134 页
- “SUM 函数 [Aggregate]”一节第 307 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例返回用于拟合回归线的非 NULL 值对的数量。

```
SELECT REGR_COUNT( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_INTERCEPT 函数 [Aggregate]

计算可最佳拟合非独立与独立变量的线性回归线的 y 截距。

语法 1

REGR_INTERCEPT(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_INTERCEPT(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。在一次处理数据期间同时计算该函数。排除 NULL 值后，将进行以下计算，其中的 *y* 表示 *dependent-expression*，*x* 表示 *independent-expression*：

```
AVG( y ) - REGR_SLOPE( y, x ) * AVG( x )
```

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_SLOPE 函数 [Aggregate]”一节第 271 页
- “REGR_SXX 函数 [Aggregate]”一节第 272 页
- “REGR_SXY 函数 [Aggregate]”一节第 274 页
- “REGR_SYY 函数 [Aggregate]”一节第 275 页
- “REGR_AVGY 函数 [Aggregate]”一节第 266 页
- “REGR_AVGX 函数 [Aggregate]”一节第 265 页
- “REGR_SLOPE 函数 [Aggregate]”一节第 271 页
- “AVG 函数 [Aggregate]”一节第 134 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例返回线性回归线的 y 截距。

```
SELECT REGR_INTERCEPT( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_R2 函数 [Aggregate]

计算回归线的确定系数（也称为 *R 平方*或*适配度*统计）。

语法 1

REGR_R2(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_R2(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_INTERCEPT 函数 [Aggregate]”一节第 268 页
- “REGR_SLOPE 函数 [Aggregate]”一节第 271 页
- “REGR_SXX 函数 [Aggregate]”一节第 272 页
- “REGR_SXY 函数 [Aggregate]”一节第 274 页
- “REGR_SYY 函数 [Aggregate]”一节第 275 页
- “REGR_AVGX 函数 [Aggregate]”一节第 265 页
- “REGR_AVGY 函数 [Aggregate]”一节第 266 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例返回回归线的确定系数。

```
SELECT REGR_R2( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_SLOPE 函数 [Aggregate]

计算拟合到非 NULL 数对的线性回归线的斜率。

语法 1

REGR_SLOPE(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_SLOPE(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。在一次处理数据期间同时计算该函数。排

除 NULL 值后，将进行以下计算，其中的 *y* 表示 *dependent-expression*，*x* 表示 *independent-expression*：

```
COVAR_POP( y, x ) / VAR_POP( x )
```

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_INTERCEPT 函数 [Aggregate]”一节第 268 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_SXX 函数 [Aggregate]”一节第 272 页
- “REGR_SXY 函数 [Aggregate]”一节第 274 页
- “REGR_SYY 函数 [Aggregate]”一节第 275 页
- “REGR_AVGX 函数 [Aggregate]”一节第 265 页
- “REGR_AVGY 函数 [Aggregate]”一节第 266 页
- “COVAR_POP 函数 [Aggregate]”一节第 159 页
- “VAR_POP 函数 [Aggregate]”一节第 323 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例返回值 935.3429749445614。

```
SELECT REGR_SLOPE( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_SXX 函数 [Aggregate]

返回线性回归模型中使用的独立表达式的平方和。REGR_SXX 函数可用于计算回归模型的统计有效性。

语法 1

REGR_SXX(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_SXX(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。在一次处理数据期间同时计算该函数。排除 NULL 值后，将进行以下计算，其中的 *y* 表示 *dependent-expression*，*x* 表示 *independent-expression*：

```
REGR_COUNT( y, x ) * VAR_POP( x )
```

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_INTERCEPT 函数 [Aggregate]”一节第 268 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_AVGX 函数 [Aggregate]”一节第 265 页
- “REGR_AVGY 函数 [Aggregate]”一节第 266 页
- “REGR_SXY 函数 [Aggregate]”一节第 274 页
- “REGR_SYY 函数 [Aggregate]”一节第 275 页
- “VAR_POP 函数 [Aggregate]”一节第 323 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例返回值 5916.4800000000105。

```
SELECT REGR_SXX( Salary, ( YEAR( NOW() ) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_SXY 函数 [Aggregate]

返回非独立变量与独立变量的积和。REGR_SXY 函数可用于计算回归模型的统计有效性。

语法 1

REGR_SXY(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_SXY(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。如果将该函数应用于某一空集，则它会返回 NULL。

该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。在一次处理数据期间同时计算该函数。排除 NULL 值后，将进行以下计算，其中的 *y* 表示 *dependent-expression*，*x* 表示 *independent-expression*：

```
REGR_COUNT( y, x ) * COVAR_POP( y, x )
```

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_INTERCEPT 函数 [Aggregate]”一节第 268 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_SLOPE 函数 [Aggregate]”一节第 271 页
- “REGR_AVGX 函数 [Aggregate]”一节第 265 页
- “REGR_AVGY 函数 [Aggregate]”一节第 266 页
- “REGR_SXX 函数 [Aggregate]”一节第 272 页
- “REGR_SYY 函数 [Aggregate]”一节第 275 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例返回非独立变量与独立变量的积和。

```
SELECT REGR_SXY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REGR_SYY 函数 [Aggregate]

返回可以计算回归模型统计有效性的值。

语法 1

REGR_SYY(*dependent-expression* , *independent-expression* **)**

语法 2

REGR_SYY(*dependent-expression* , *independent-expression* **)**
OVER (*window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **dependent-expression** 受独立变量影响的变量。
- **independent-expression** 影响结果的变量。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。如果将该函数应用于某一空集，则它会返回 NULL。

该函数应用于排除所有 *dependent-expression* 或 *independent-expression* 之一为 NULL 的对后剩下的 *dependent-expression* 和 *independent-expression* 对的集合。在一次处理数据期间同时计算该函数。排

除 NULL 值后，将进行以下计算，其中的 *y* 表示 *dependent-expression*，*x* 表示 *independent-expression*：

```
REGR_COUNT( y, x ) * VAR_POP( y )
```

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_INTERCEPT 函数 [Aggregate]”一节第 268 页
- “REGR_COUNT 函数 [Aggregate]”一节第 267 页
- “REGR_AVGX 函数 [Aggregate]”一节第 265 页
- “REGR_AVGY 函数 [Aggregate]”一节第 266 页
- “REGR_SLOPE 函数 [Aggregate]”一节第 271 页
- “REGR_SXX 函数 [Aggregate]”一节第 272 页
- “REGR_SXY 函数 [Aggregate]”一节第 274 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下示例返回值 26、708、672,843.3002。

```
SELECT REGR_SYY( Salary, ( YEAR( NOW( )) - YEAR( BirthDate ) ) )
FROM Employees;
```

REMAINDER 函数 [Numeric]

返回一个整数除以另一个整数之后产生的余数。

语法

REMAINDER(*dividend*, *divisor* **)**

参数

- **dividend** 被除数，即除法的分子。
- **divisor** 除数，即除法的分母。

返回值

INTEGER

NUMERIC

注释

可以尝试使用 MOD 函数作为替代。

另请参见

- “MOD 函数 [Numeric]”一节第 239 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 2。

```
SELECT REMAINDER( 5, 3 );
```

REPEAT 函数 [String]

以指定的次数连接字符串。

语法

REPEAT(*string-expression*, *integer-expression* **)**

参数

- **string-expression** 要重复的字符串。
- **integer-expression** 字符串的重复次数。如果 *integer-expression* 是负数，则返回空字符串。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

如果结果字符串的实际长度超过返回类型的最大值，会产生错误。结果将截断为所允许的最大字符串大小。

可以尝试使用 REPLICATE 函数作为替代。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “REPLICATE 函数 [String]”一节第 279 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句返回值 repeatrepeatrepeat。

```
SELECT REPEAT( 'repeat', 3 );
```

REPLACE 函数 [String]

使用另一个字符串替换某个字符串，并返回新的结果。

语法

REPLACE(*original-string*, *search-string*, *replace-string* **)**

参数

如果有某个参数为 NULL，此函数返回 NULL。

- **original-string** 被搜索的字符串。可为任意长度。
- **search-string** 要进行搜索并以 *replace-string* 进行替换的字符串。此字符串的长度不应超过 255 个字节。如果 *search-string* 是空字符串，则按原样返回原始字符串。
- **replace-string** 替代字符串，用于替换 *search-string*。可为任意长度。如果 *replacement-string* 是空字符串，则会删除出现的所有 *search-string*。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

此函数替换所有出现的字符串。

对于区分大小写的数据库，比较也区分大小写。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “SUBSTRING 函数 [String]”一节第 305 页
- “CHARINDEX 函数 [String]”一节第 144 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 xx.def.xx.ghi。

```
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' );
```

以下语句生成包含 ALTER PROCEDURE 语句的结果集，这些语句在执行时会修复那些引用已重命名的表的存储过程。（表名必须唯一才会有用处。）

```
SELECT REPLACE(
   REPLACE( proc_defn, 'OldTableName', 'NewTableName' ),
   'CREATE PROCEDURE',
   'ALTER PROCEDURE')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%';
```

REPLICATE 函数 [String]

以指定的次数连接字符串。

语法

REPLICATE(*string-expression*, *integer-expression* **)**

参数

- **string-expression** 要重复的字符串。
- **integer-expression** 字符串的重复次数。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

如果结果字符串的实际长度超过返回类型的最大值，会产生错误。结果将截断为所允许的最大字符串大小。

可以尝试使用 REPEAT 函数作为替代。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “REPEAT 函数 [String]”一节第 277 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句返回值 repeatrepeatrepeat。

```
SELECT REPLICATE( 'repeat', 3 );
```

REVERSE 函数 [String]

返回字符表达式的相反值。

语法

REVERSE(*string-expression* **)**

参数

- **string-expression** 要求相反值的字符串。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

此函数支持 NCHAR 输入和/或输出。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 cba。

```
SELECT REVERSE( 'abc' );
```

REWRITE 函数 [Miscellaneous]

返回重写的 SELECT、UPDATE 或 DELETE 语句。

语法

REWRITE(*select-statement* **[, 'ANSI'])**

参数

- **select-statement** 要应用重写优化以便生成该函数结果的 SQL 语句。

返回值

LONG VARCHAR

注释

使用 REWRITE 函数时可以不带 ANSI 参数，这样可以帮助理解优化程序是如何为给定查询生成访问计划的。具体地讲，可以了解 SQL Anywhere 如何重写语句的 WHERE、ON 和 HAVING 子句中的条件，然后确定是否存在可用于缩短请求执行时间的适用索引。

REWRITE 返回的语句可能与原始语句的语义不匹配。这是因为一些重写优化引入了不能直接转换成 SQL 的内部机制。例如，服务器使用行标识符执行重复项消除就不能转换成 SQL。

REWRITE 函数重写的查询不是用来执行的。它是一种工具，用于通过显示重写阶段后传递给优化程序的内容来分析性能问题。

有些重写优化在 REWRITE 的输出中并未反映出来。它们包括 LIKE 优化、最小或最大函数的优化、上限/下限消除以及判定包容。

如果指定了 ANSI，则 REWRITE 返回语句的 ANSI 等效项。在这种情况下，将仅应用以下重写优化：

- Transact-SQL 外连接被重写为 ANSI SQL 外连接。
- 消除重复的相关名。
- KEY 和 NATURAL 连接被重写为 ANSI SQL 连接。

另请参见

- “语义查询转换”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “extended_join_syntax 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “Transact-SQL 外连接（*= 或 =*）”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “键连接”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “自然连接”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “连接中的重复相关名（星形连接）”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在以下示例中，对查询执行了两个重写优化。第一个优化解除了对 Employees 与 SalesOrders 表之间连接进行的子查询的嵌套。第二个优化通过消除 Employees 与 SalesOrders 之间的主键 - 外键连接简化了查询。此重写优化的一部分用连接谓语 s.SalesRepresentative IS NOT NULL 替换谓语 e.EmployeeID=s.SalesRepresentative。

```
SELECT REWRITE( 'SELECT s.ID, s.OrderDate
    FROM SalesOrders s
    WHERE EXISTS ( SELECT *
       FROM Employees e
          WHERE e.EmployeeID = s.SalesRepresentative)' ) FROM dummy;
```

查询返回包含重写后查询的单列结果集：

```
'SELECT s.ID, s.OrderDate FROM SalesOrders s WHERE s.SalesRepresentative IS
NOT NULL'
```

下一个 REWRITE 示例使用 ANSI 参数。

```
SELECT REWRITE( 'SELECT DISTINCT s.ID, s.OrderDate, e.GivenName, e.EmployeeID
    FROM SalesOrders s, Employees e
          WHERE e.EmployeeID *= s.SalesRepresentative', 'ANSI' ) FROM dummy;
```

结果返回语句相应的 ANSI 形式。这种情况下，Transact-SQL 外连接转换为 ANSI 外连接。此查询返回一个单列结果集（该结果集已被分成几行以提高可读性）：

```
'SELECT DISTINCT s.ID, s.OrderDate, e.GivenName, e.EmployeeID
 FROM Employees as e
 LEFT OUTER JOIN  SalesOrders as s
 ON e.EmployeeID = s.SalesRepresentative';
```

RIGHT 函数 [String]

返回字符串中最右边的字符。

语法

RIGHT(*string-expression*, *integer-expression* **)**

参数

- **string-expression** 从左边截断的字符串。
- **integer-expression** 要返回的从字符串结尾处开始计数的字符数。

返回值

LONG VARCHAR

LONG NVARCHAR

注释

如果字符串包含多字节字符，并且使用了适当的归类，则返回的字节数可能大于指定的字符数。

指定的 *integer-expression* 可以比列中的值大。这种情况下将返回整个值。

此函数支持 NCHAR 输入和/或输出。如果输入字符串使用字符长度语义，就会在可能的情况下根据字符长度语义对返回值进行说明。

另请参见

- “LEFT 函数 [String]”一节第 226 页
- “国际语言和字符集” 《SQL Anywhere 服务器 - 数据库管理》
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 Customers 表中每个 Surname 值的最后 5 个字符。

```
SELECT RIGHT( Surname, 5) FROM Customers;
```


ROUND 函数 [Numeric]

将 *numeric-expression* 舍入到小数点后的指定 *integer-expression* 位数。

语法

ROUND(*numeric-expression*, *integer-expression* **)**

参数

- **numeric-expression** 要舍入的数字（传递给函数）。
- **integer-expression** 正整数指定小数点右边舍入到的有效位数。负数表达式指定小数点左边舍入到的有效位数。

返回值

NUMERIC

注释

此函数的结果为数字或双精度值。当结果为数字值并且整数 *integer-expression* 是负值时，精度将加一。

另请参见

- “TRUNCNUM 函数 [Numeric]”一节第 316 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 123.200。

```
SELECT ROUND( 123.234, 1 );
```

ROWID 函数 [Miscellaneous]

返回一个无符号 64 位值，它唯一地标识表内的一行。

语法

ROWID(*correlation-name* **)**

参数

- **correlation-name** 查询中使用的表的相关名。相关名应引用基表、临时表、全局临时表或代理表（仅在基础代理服务器支持类似函数时允许）。ROWID 函数的参数不应引用视图、派生表、公用表表达式或过程。

返回值

UNSIGNED BIGINT

注释

返回对应于给定相关名的表中的行标识符。

此函数返回的值在进行不同查询时不一定会保持不变，因为在数据库上执行的各种操作可能会导致表的行标识符发生变化。具体地讲，REORGANIZE TABLE 语句可能会导致行标识符发生变化。此外，还可能在删除行后重用了行标识符。因此，用户应避免在普通情况下使用 ROWID 函数，而应改用按主键值检索。建议仅在进行诊断时使用 ROWID。

尽管此函数的结果为 UNSIGNED BIGINT 类型，但对此值进行的大部分算术运算的结果并无特殊意义。例如，请不要期望将行标识符加一即可得到下一行的行标识符。另外，只有等式和 IN 谓语涉及 ROWID 的使用时，才可以对它们进行优化。必要时，涉及 ROWID 的谓语（如 `ROWID( T ) = literal`）可能会转换为 64 位 UNSIGNED INTEGER 值。如果转换无法执行，会发生数据异常。如果 *literal* 的值是无效的行标识符，则比较谓语的计算结果为 FALSE。

ROWID 函数不能在表或列中的 CHECK 约束内使用，也不能在计算列的 COMPUTE 表达式中使用。

另请参见

- “ROW_NUMBER 函数 [Miscellaneous]”一节第 284 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 Employee 中 id 等于 105 的行的行标识符：

```
SELECT ROWID( Employees ) FROM Employees WHERE Employees.EmployeeID = 105;
```

以下语句返回 Employees 表中行上锁的列表以及这些行的内容：

```
SELECT *
  FROM sa_locks() S JOIN Employees WITH( NOLOCK )
     ON ROWID( Employees ) = S.row_identifier
  WHERE S.table_name = 'Employees';
```

ROW_NUMBER 函数 [Miscellaneous]

为每一行指派一个唯一的编号。使用此函数替代 NUMBER 函数。

语法

ROW_NUMBER( ) OVER (*window-spec* **)**

window-spec：请参见下面的“注释”部分

返回值

INTEGER

注释

可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。作为窗口函数使用时，必须指定 ORDER BY 子句，还可以指定 PARTITION BY 子句，但不能指定 ROWS 或 RANGE 子句。请参见“WINDOW 子句”一节 第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “NUMBER 函数 [Miscellaneous]”一节第 249 页
- “RANK 函数 [Ranking]”一节第 261 页
- “ROWID 函数 [Miscellaneous]”一节第 283 页

标准和兼容性

- **SQL/2003** SQL/OLAP 特性 T612。

示例

以下示例返回一个结果集，它提供 New York 和 Utah 州每个雇员的唯一行号。由于该查询按 Salary 降序排序，因此第一个行号将指派给数据集中薪水最高的雇员。尽管有两个雇员的薪水完全相同，但不会解决这种并列情况，因为给这两个雇员指派了唯一的行号。

```
SELECT Surname, Salary, State,
ROW_NUMBER() OVER (ORDER BY Salary DESC) "Rank"
FROM Employees WHERE State IN ('NY','UT');
```

Surname	Salary	State	Rank
Shishov	72995.000	UT	1
Wang	68400.000	UT	2
Cobb	62000.000	UT	3
Morris	61300.000	UT	4
Davidson	57090.000	NY	5
Martel	55700.000	NY	6
Blaikie	54900.000	NY	7
Diaz	54900.000	NY	8
Driscoll	48023.690	UT	9
Hildebrand	45829.000	UT	10
...	...	...	...

Surname	Salary	State	Rank
Lynch	24903.000	UT	19

RTRIM 函数 [String]

删除字符串中的前导和尾随空白。

语法

RTRIM(*string-expression* **)**

参数

- **string-expression** 要剪裁的字符串。

返回值

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

注释

结果的实际长度为表达式的长度减去删除的字符数。如果删除了所有字符，则结果为空字符串。

如果参数为空值，则结果为空值。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “TRIM 函数 [String]”一节第 315 页
- “LTRIM 函数 [String]”一节第 234 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

  由 SQL/2003 标准定义的 TRIM 说明（LEADING 和 TRAILING）分别由 SQL Anywhere LTRIM 和 RTRIM 函数提供。

示例

以下语句返回字符串 Test Message，其中的所有末尾空格都已删除。

```
SELECT RTRIM( 'Test Message     ' );
```


SECOND 函数 [Date and time]

返回给定日期的秒。

语法

SECOND(*datetime-expression* **)**

参数

- **datetime-expression** 日期时间值。

返回值

SMALLINT

注释

返回 0 到 59 之间的一个数字，该数字对应于给定日期时间值的秒部分。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 25。

```
SELECT SECOND( '1998-07-13 21:21:25' );
```

SECONDS 函数 [Date and time]

此函数的行为会视所提供内容的不同而变化：

- 如果提供一个日期，此函数返回自 0000-02-29 以来的秒数。

  注意
  0000-02-29 并不表示实际日期，它是日期算法使用的日期。

- 如果提供两个时间戳，此函数返回它们之间的整秒数。替代方法为使用 DATEDIFF 函数。
- 如果提供一个日期和一个整数，此函数会为指定时间戳添加整数秒数。替代方法为使用 DATEADD 函数。

语法 1：integer

SECONDS([*datetime-expression*,] *datetime-expression* **)**

语法 2：timestamp

SECONDS(*datetime-expression*, *integer-expression* **)**

参数

- **datetime-expression** 日期和时间。
- **integer-expression** 要添加到 *datetime-expression* 中的秒数。如果 *integer-expression* 是负数，则从日期时间值中减去相应的秒数。如果提供整数表达式，则必须将 *datetime-expression* 显式地转换为日期时间数据类型。

返回值

INTEGER

TIMESTAMP

另请参见

- “CAST 函数 [Data type conversion]”一节第 141 页
- “DATEADD 函数 [Date and time]”一节第 165 页
- “DATEDIFF 函数 [Date and time]”一节第 166 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 14400，表示第二个时间戳比第一个时间戳晚 14400 秒。

```
SELECT SECONDS( '1999-07-13 06:07:12',
   '1999-07-13 10:07:12' );

SELECT DATEDIFF( second,
   '1999-07-13 06:07:12',
   '1999-07-13 10:07:12' );
```

以下语句返回值 63062431632。

```
SELECT SECONDS( '1998-07-13 06:07:12' );
```

以下语句返回日期时间 1999-05-12 21:05:120.000。

```
SELECT SECONDS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5);

SELECT DATEADD( second, 5, '1999-05-12 21:05:07' );
```

SET_BIT 函数 [Bit array]

设置位数组中特定位的值。

语法

SET_BIT([*bit-expression*,]*bit-position* [, *value*]**)**

参数

- **bit-expression** 在其中更改位的位数组。
- **bit-position** 要设置的位的位置。它必须是无符号整数。
- **value** 要将位设置成的值。

返回值

LONG VARBIT

注释

bit-expression 的缺省值是长度为 *bit-position* 的位数组，其中包含所有设置为 0 (FALSE) 的位。

value 的缺省值是 1 (TRUE)。

如果任何参数为 NULL，则结果为 NULL。

数组中的位置从左侧开始计数（从 1 开始）。

另请参见

- “GET_BIT 函数 [Bit array]”一节第 200 页
- “SET_BITS 函数 [Aggregate]”一节第 289 页
- “INTEGER 数据类型”一节第 88 页
- “位运算符”一节第 14 页
- “sa_get_bits 系统过程”一节第 827 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 00100011：

```
SELECT SET_BIT( '00110011', 4 , 0);
```

以下语句返回值 00111011：

```
SELECT SET_BIT( '00110011', 5 , 1);
```

以下语句返回值 00111011：

```
SELECT SET_BIT( '00110011', 5 );
```

以下语句返回值 00001：

```
SELECT SET_BIT( 5 );
```

SET_BITS 函数 [Aggregate]

创建一个位数组，其中与某组行的值相对应的特定位设置为 1 (TRUE)。

语法

SET_BITS(*expression* **)**

参数

- **expression** 用于确定哪些位设置为 1 的表达式。其值通常为列名。

返回值

LONG VARBIT

注释

将忽略那些指定的值为 NULL 的行。

如果没有任何行，则返回 NULL。

结果的长度是设置为 1 的最大位置。

SET_BITS 函数等同于以下语句，但执行速度更快：

```
SELECT BIT_OR( SET_BIT( expression ) )
FROM table;
```

另请参见

- “位运算符”一节第 14 页
- “GET_BIT 函数 [Bit array]”一节第 200 页
- “SET_BIT 函数 [Bit array]”一节第 288 页
- “sa_get_bits 系统过程”一节第 827 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回第 2、5、10 位设置为 1（或 0100100001）的位数组：

```
CREATE TABLE t( r INTEGER );
INSERT INTO t values( 2 );
INSERT INTO t values( 5 );
INSERT INTO t values(10 );
SELECT SET_BITS( r ) FROM t;
```

SIGN 函数 [Numeric]

返回一个数字的符号。

语法

SIGN(*numeric-expression* **)**

参数

- **numeric-expression** 要返回其符号的数字。

返回值

SMALL INT

注释

对于负数，SIGN 函数返回 -1。

对于零，SIGN 函数返回 0。

对于正数，SIGN 函数返回 1。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 -1

```
SELECT SIGN( -550 );
```

SIMILAR 函数 [String]

返回一个数字，以表示两个字符串之间的相似性。

语法

SIMILAR(*string-expression-1*, *string-expression-2* **)**

参数

- **string-expression-1** 要比较的第一个字符串。
- **string-expression-2** 要比较的第二个字符串。

返回值

SMALL INT

注释

此函数返回 0 和 100 之间的一个整数，表示两个字符串之间的相似性。结果可以被解释成两个字符串之间匹配字符的百分比。值 100 表示两个字符串完全相同。

此函数可用于更正名称（如客户）列表。某些客户可能用稍微不同的名称多次添加到列表中。将表与自身连接，并生成一个报告，其中列出所有大于 90% 但小于 100% 的相似性。

为 SIMILAR 函数执行的计算要比仅计算匹配的字符数更为复杂。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 75，表示两个值的相似性为 75%。

```
SELECT SIMILAR( 'toast', 'coast' );
```

SIN 函数 [Numeric]

返回一个数字的正弦值。

语法

SIN(*numeric-expression* **)**

参数

- **numeric-expression** 角度（以弧度为单位）。

返回值

DOUBLE

注释

SIN 函数返回参数的正弦，参数是以弧度表示的角度。SIN 和 ASIN 函数互为逆运算。

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。

另请参见

- “ASIN 函数 [Numeric]”一节第 131 页
- “COS 函数 [Numeric]”一节第 156 页
- “COT 函数 [Numeric]”一节第 156 页
- “TAN 函数 [Numeric]”一节第 309 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 0.52 的 SIN 值。

```
SELECT SIN( 0.52 );
```

SOAP_HEADER 函数 [SOAP]

返回 SOAP 标头条目，或 SOAP 请求的标头条目的特性值。

语法

SOAP_HEADER(*header-key* [*index*, *header-attribute*] **)**

参数

- **header-key** 此 VARCHAR 参数为给定 SOAP 标头条目指定顶级 XML 元素的 XML 本地名。
- **index** 此可选 INTEGER 参数用于区分具有完全相同名称的 SOAP 标头字段。当多个标头条目具有 localname 相同的顶级 XML 元素时，就会发生这种情况。通常，此类元素具有唯一的命名空间。
- **header-attribute** 此可选 VARCHAR 参数可以是标头条目元素内的任何属性节点，其中包括：
  - **@namespace** 用于访问给定标头条目的命名空间的特殊 SQL Anywhere 特性。
  - **mustUnderstand** 一个 SOAP 1.1 标头条目特性，表示供接收方处理的标头条目是强制性的还是非强制性的。
  - **encodingStyle** 一个表示编码样式的 SOAP 1.1 标头条目。可以访问此特性，但 SQL Anywhere 并不在内部使用它。
  - **actor** 一个 SOAP 1.1 标头条目特性，它通过指定接收方的 URL 来表示计划的标头条目接收方。

返回值

LONG VARCHAR

注释

此函数可以与单个参数 *header-key* 一起使用以返回标头条目。标头条目是包含在 SOAP 标头内的元素及其所有子元素的 XML 字符串表示。

通过指定可选 *index* 和 *header-attribute* 参数，此函数还可以用于抽取标头条目特性。

此函数返回指定 SOAP 标头字段的值，如果不是从 SOAP 服务进行调用，则返回 NULL。当通过 Web 服务处理 SOAP 请求时，将使用该函数。

如果给定 *header-key* 的标头不存在，则返回值为 NULL。

另请参见

- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “处理 SOAP 标头”一节 《SQL Anywhere 服务器 - 编程》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “NEXT_SOAP_HEADER 函数 [SOAP]”一节第 247 页
- “sa_set_soap_header 系统过程”一节第 897 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

标准和兼容性

- **SQL/2003** 服务商扩充。

SORTKEY 函数 [String]

生成排序关键字值。也就是说，可用于根据备选归类规则对字符串进行排序的值。

语法

SORTKEY(*string-expression*
[, { *collation-id*
| *collation-name*[**(***collation-tailoring-string***)**] }]
)

参数

- **string-expression** 字符串表达式必须包含使用数据库的字符集编码的字符。

  如果 *string-expression* 是空字符串，则 SORTKEY 函数将返回长度为零的二进制值。如果 *string-expression* 为 NULL，则 SORTKEY 函数返回 NULL 值。空字符串的排序顺序值不同于数据库列中的 NULL 字符串。

  SORTKEY 函数可以处理的字符串的最大长度是 254 个字节。任何多出的部分都将被忽略。

- **collation-name** 一个字符串或字符变量，用于指定要使用的排序顺序的名称。还可以指定别名 char_collation 或等效的 db_collation，以生成数据库正在使用的 CHAR 归类所使用的 sortkey。类似地，可以指定别名 nchar_collation 以生成数据库正在使用的 NCHAR 归类所使用的 sortkey。

- **collation-id** 一个变量、整数常量或字符串，用于指定要使用的排序顺序的 ID 号。如果不指定 *collation-id*，则缺省值为缺省的 Unicode 多语言。有关有效归类的列表，请参见“支持的归类和替代归类”一节 《SQL Anywhere 服务器 - 数据库管理》和“建议的字符集和归类”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **collation-tailoring-string** 您还可以指定归类定制选项 (*collation-tailoring-string*)，以便能够对字符的排序和比较进行更多的控制。这些选项采用 "关键字=值" 对的形式，包括在圆括号内，位于归类名后。例如，`'UCA(locale=es;case=LowerFirst;accent=respect)'`。指定这些选项的语法与为 CREATE DATABASE 语句的 COLLATION 子句定义的语法完全相同。请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

  注意
  指定 UCA 归类时，支持所有归类定制选项。对于所有其它归类，仅支持区分大小写定制。

返回值

BINARY

注释

SORTKEY 函数生成的值可用于根据预定义的排序顺序行为对结果进行排序。这样，您就可以使用数据库归类无法提供的字符排序顺序行为。返回值是一个二进制值，包含有关通过 SORTKEY 函数保留的输入字符串的已编码排序顺序信息。例如，可以将 SORTKEY 函数返回的值存储在包含源字符串的列中。当您想按所需顺序检索字符数据时，SELECT 语句只需要加入作用于包含 SORTKEY 函数运行结果的列的 ORDER BY 子句即可。

SORTKEY 函数保证它为一组给定的排序顺序标准返回的值适用于对 VARBINARY 数据类型执行的二进制比较。

生成查询的 sortkey 开销可能很大。作为频繁请求的 sortkey 的一种替代，可以考虑创建一个计算列来保存 sortkey 值，然后在查询的 ORDER BY 子句中引用该列。

SORTKEY 函数的输入最多可以为每个输入字符生成 6 个字节的排序顺序信息。SORTKEY 函数的输出是 VARBINARY 类型，最大长度为 1024 个字节。

如果在生成排序关键字期间为归类指定 UCA，缺省情况下归类定制是区分重音和大小写的。例如，如果 UCA 指定自身，应用的缺省定制等效于 'UCA(case=UpperFirst;accent=Respect;punct=Primary)'。

如果在 SORTKEY 的第二个参数中提供不同的定制，这些设置替换缺省设置。例如，以下两个语句是等效的：

```
SELECT SORTKEY( 'abc', 'UCA(accent=Ignore)' );
SELECT SORTKEY( 'abc', 'UCA(case=UpperFirst;accent=Ignore;punct=Primary)' );
```

如果指定非 UCA 归类，缺省情况下，归类定制也是区分重音和大小写的。但是，对于非 UCA 归类，只能使用归类定制来替换区分大小写特性。例如：

```
SELECT SORTKEY( 'abc', '1252LATIN1(case=Respect)' );
```

如果创建数据库时未指定定制选项（例如，dbinit -c -zn uca mydb.db），即使为 SORTKEY 函数指定了数据库归类名称，以下两个子句也可能生成不同的排序顺序：

```
ORDER BY string-expression
ORDER BY SORTKEY( string-expression, database-collation-name )
```

这是因为数据库创建所使用的缺省定制设置与 SORTKEY 函数所使用的缺省定制设置不同。要使 SORTKEY 获得与数据库归类相同的行为，可以为 *collation-tailoring-string* 提供与数据库归类的设置匹配的定制语法，也可以为 *collation-name* 指定 db_collation。例如：

```
SORTKEY( expression, 'db_collation' )
```

注意

排序关键字值取决于创建数据库文件时所用的 SQL Anywhere 版本。例如，由 10.0.0 之前版本的 SQL Anywhere 创建的排序关键字值与 SQL Anywhere 版本 10 及 SQL Anywhere 版本 11 所创建的值不同。如果将某版本 SQL Anywhere 中创建的排序关键字值用在其它版本的 SQL Anywhere 所创建的数据库中，这种情况可能会对您的应用程序带来问题。

建议您在使用卸载和重装的方式升级 SQL Anywhere 软件时，重新生成存储在数据库中的排序关键字值。

另请参见

- “sort_collation 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “COMPARE 函数 [String]”一节第 146 页
- “国际语言和字符集” 《SQL Anywhere 服务器 - 数据库管理》
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句查询 Employees 表并返回所有雇员的 FirstName 和 Surname，按 Surname 列的 sortkey 值排序，使用 dict 归类（拉丁文-1、英语、法语、德语字典）。

```
SELECT Surname, GivenName FROM Employees ORDER BY SORTKEY( Surname, 'dict' );
```

以下示例使用 UCA 归类和定制选项来返回 abc 的 sortkey 值。

```
SELECT SORTKEY( 'abc', 'UCA(locale=es;case=LowerFirst;accent=respect)' );
```

SOUNDEX 函数 [String]

返回表示字符串声音的数字。

语法

SOUNDEX(*string-expression* **)**

参数

- **string-expression** 要计算的字符串。

返回值

SMALLINT

注释

字符串的 SOUNDEX 函数值基于第一个字母和随后三个除 H、Y 和 W 之外的辅音。*string-expression* 中的元音将被忽略，除非它们是字符串的第一个字母。重复的字母按一个字母计数。例如，词语 apples 基于字母 A、P、L 和 S。

SOUNDEX 函数会忽略多字节字符。

尽管并不完美，但对于那些发音相似及以相同字母开头的单词，SOUNDEX 函数返回的数字通常是相同的。

SOUNDEX 函数对英语单词的效果最好。对其它语言单词的效果较差。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回两个完全相同的数字 3827，表示每个名称的声音。

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' );
```

SPACE 函数 [String]

返回指定数目的空格。

语法

SPACE(*integer-expression* **)**

参数

- **integer-expression** 要返回的空格数。

返回值

LONG VARCHAR

注释

如果 *integer-expression* 为负，则返回空字符串。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回包含 10 个空格的字符串。

```
SELECT SPACE( 10 );
```

SQLDIALECT 函数 [Miscellaneous]

返回 Watcom-SQL 或 Transact-SQL，以表示语句的 SQL 方言。

语法

SQLDIALECT(*sql-statement-string* **)**

参数

- **sql-statement-string** 函数用来确定其方言的 SQL 语句。

返回值

LONG VARCHAR

另请参见

- “TRANSACTSQL 函数 [Miscellaneous]”一节第 314 页
- “WATCOMSQL 函数 [Miscellaneous]”一节第 326 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回字符串 Transact-SQL。

```
SELECT
   SQLDIALECT( 'SELECT EmployeeName = Surname FROM Employees' )
FROM dummy;
```

SQLFLAGGER 函数 [Miscellaneous]

返回给定 SQL 语句是否符合指定标准。

语法

SQLFLAGGER(*sql-standard-string*, *sql-statement-string* **)**

参数

- **sql-standard-string** 测试遵从性所使用的标准级别。可能的值与 sql_flagger_error_level 数据库选项的值相同：
  - **SQL:2003/Core** 测试是否符合核心 SQL/2003 语法。
  - **SQL:2003/Package** 测试是否符合完整的 SQL/2003 语法。
  - **SQL:1999/Core** 测试是否符合核心 SQL/1999 语法。
  - **SQL:1999/Package** 测试是否符合完整的 SQL/1999 语法。
  - **SQL:1992/Entry** 测试是否符合入门级 SQL/1992 语法。
  - **SQL:1992/Intermediate** 测试是否符合中级 SQL/1992 语法。
  - **SQL:1992/Full** 测试是否符合完整的 SQL/1992 语法。
  - **UltraLite** 测试是否符合 UltraLite。
- **sql-statement-string** 要检查符合性的 SQL 语句。

返回值

LONG VARCHAR

另请参见

- “sql_flagger_error_level 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL 预处理器”一节 《SQL Anywhere 服务器 - 编程》
- “sa_ansi_standard_packages 系统过程”一节第 796 页
- “使用 SQL Flagger 测试 SQL 遵从性”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句显示发现不允许的扩充时返回的消息的示例：


```
SELECT SQLFLAGGER(
   'SQL:2003/Package', 'SELECT top 1 dummy_col FROM sys.dummy ORDER BY
dummy_col' );
```

此语句返回消息 '0AW03 Disallowed language extension detected in syntax near 'top' on line 1'。

以下语句将返回 '00000'，因为它不包含不允许的扩展：

```
SELECT SQLFLAGGER( 'SQL:2003/Package', 'SELECT dummy_col FROM sys.dummy' );
```

SQRT 函数 [Numeric]

返回一个数字的平方根。

语法

SQRT(*numeric-expression* **)**

参数

- **numeric-expression** 要计算其平方根的数字。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 3。

```
SELECT SQRT( 9 );
```

STDDEV 函数 [Aggregate]

STDDEV_SAMP 的别名。请参见“STDDEV_SAMP 函数 [Aggregate]”一节第 301 页。

STDDEV_POP 函数 [Aggregate]

计算由数字表达式组成的总体的标准差，类型为 DOUBLE。

语法 1

STDDEV_POP(*numeric-expression* **)**

语法 2

STDDEV_POP(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **numeric-expression** 一个表达式，其基于总体的标准差是以一组行为基础进行计算。此表达式通常是列名。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

根据下面的公式计算基于总体的标准差：

$s = [(1/N) * SUM(x_I - mean(x))^2]^{½}$

此标准差不包括 *numeric-expression* 是 NULL 的行。对于不包含任何行的组，它返回 NULL。

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下语句列出不同时间段每个订单的产品数目的平均值和方差：

```
SELECT YEAR( ShipDate ) AS Year,
    QUARTER( ShipDate ) AS Quarter,
    AVG( Quantity ) AS Average,
    STDDEV_POP( quantity ) AS Variance
FROM SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.2794...

Year	Quarter	Average	Variance
2000	2	27.050847	15.0270...
...	...	...	...

STDDEV_SAMP 函数 [Aggregate]

计算由数字表达式组成的样本的标准差，类型为 DOUBLE。

语法 1

STDDEV_SAMP(*numeric-expression* **)**

语法 2

STDDEV_SAMP(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **numeric-expression** 一个表达式，其基于样本的标准差以一组行为基础进行计算。此表达式通常是列名。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，并以双精度浮点执行计算。

根据下面的公式计算标准差，其中假定正态分布：

$s = [(1/(N - 1)) * SUM(x_I - mean(x))^2]^{½}$

此标准差不包括 *numeric-expression* 是 NULL 的行。对于包含 0 或 1 行的组，它返回 NULL。

有关执行的统计计算的详细信息，请参见“集合函数的数学公式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T621)。

示例

以下语句列出不同时间段每个订单的产品数目的平均值和方差：

```
SELECT YEAR( ShipDate ) AS Year,
    QUARTER( ShipDate ) AS Quarter,
    AVG( Quantity ) AS Average,
    STDDEV_SAMP( quantity ) AS Variance
FROM SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	14.3218...
2000	2	27.050847	15.0696...
...	...	...	...

STR 函数 [String]

返回数字的等效字符串。

语法

STR(*numeric-expression* [, *length* [, *decimal*]] **)**

参数

- **numeric-expression** 任何 -1E126 与 1E127 之间的近似数字（浮点、实数或双精度）表达式。
- **length** 要返回的字符数（包括小数点、小数点右侧和左侧的所有位以及空白）。缺省值为 10。
- **decimal** 要返回的小数位数。缺省值为 0。

返回值

VARCHAR

注释

如果数字的整数部分超过了指定的长度，则结果是具有指定长度但只包含星号的字符串。例如，以下语句返回 ***。

```
SELECT STR( 1234.56, 3 );
```

注意
所支持的最大长度为 128。任何不在 1 到 128 范围内的长度都会生成 NULL 结果。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回的字符串共有十个字符，前面是六个空格，后面是 1235。

```
SELECT STR( 1234.56 );
```

以下语句返回结果 1234.6。

```
SELECT STR( 1234.56, 6, 1 );
```

STRING 函数 [String]

将一个或多个字符串连接为一个大字符串。

语法

STRING(*string-expression* **[, ...])**

参数

- **string-expression** 要计算的字符串。

  如果只提供一个参数，则会将它转换为单个表达式。如果提供多个参数，则将它们连接成单个字符串。

返回值

LONG VARCHAR

LONG NVARCHAR

LONG BINARY

注释

数字或日期参数在连接前将会转换为字符串。STRING 函数还可用于通过将任何单个表达式作为唯一的参数进行提供来将该表达式转换为字符串。

如果所有参数都为 NULL，则 STRING 返回 NULL。如果任何参数为非 NULL，则将所有 NULL 参数都视为空字符串。

另请参见

- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 *testing123*。

```
SELECT STRING( 'testing', NULL, 123 );
```

STRTOUUID 函数 [String]

将字符串值转换为唯一标识符（UUID 或 GUID）值。

较新的数据库中不需要
在使用 9.0.2 以前的版本创建的数据库中，UNIQUEIDENTIFIER 数据类型定义为用户定义的数据类型，需要使用 STRTOUUID 和 UUIDTOSTR 函数在 UUID 值的二进制与字符串表示之间进行转换。

在使用 9.0.2 或更高版本创建的数据库中，UNIQUEIDENTIFIER 数据类型已更改为本地数据类型，SQL Anywhere 会在需要的时候进行转换。在这些版本中不需要使用 STRTOUUID 和 UUIDTOSTR 函数。

有关详细信息，请参见“UNIQUEIDENTIFIER 数据类型”一节第 102 页。

语法

STRTOUUID(*string-expression* **)**

参数

- **string-expression** *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* 格式的字符串。

返回值

UNIQUEIDENTIFIER

注释

将 *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*（其中 *x* 是十六进制数字）格式的字符串转换为一个唯一的标识符值。

如果该字符串不是有效的 UUID 字符串，将返回转换错误，除非 conversion_error 选项设置为 OFF（在此情况下返回 NULL）。

此函数用于在数据库中插入 UUID 值。

此函数支持 NCHAR 输入和/或输出。

另请参见

- “UUIDTOSTR 函数 [String]”一节第 322 页
- “NEWID 函数 [Miscellaneous]”一节第 242 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

STUFF 函数 [String]

从一个字符串中删除多个字符，并用另一个字符串替换它们。

语法

STUFF(*string-expression-1*, *start*, *length*, *string-expression-2* **)**

参数

- **string-expression-1** 要用 STUFF 函数修改的字符串。
- **start** 开始删除字符的字符位置。字符串中第一个字符的位置是 1。
- **length** 要删除的字符数。
- **string-expression-2** 要插入的字符串。要使用 STUFF 函数删除字符串的一部分，请使用 NULL 作为替换字符串。

返回值

LONG NVARCHAR

注释

此函数支持 NCHAR 输入和/或输出。

另请参见

- “INSERTSTR 函数 [String]”一节第 219 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 chocolate pie。

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' );
```

SUBSTRING 函数 [String]

返回字符串的子串。

语法

{ **SUBSTRING** | **SUBSTR** } **(** *string-expression*, *start*
[, *length*] **)**

参数

- **string-expression** 从中返回子串的字符串。

- **start** 要返回的子串的开始位置（以字符为单位）。
- **length** 要返回的子串的长度（以字符为单位）。如果指定了 *length*，便将子串限制为不能超过该长度。

返回值

LONG BINARY

LONG VARCHAR

LONG NVARCHAR

注释

此函数的行为取决于 ansi_substring 数据库选项的设置。当将 ansi_substring 选项设置为 On（缺省值）时，SUBSTRING 函数的行为对应于 ANSI/ISO SQL/2003 行为。其行为如下所示：

ansi_substring 选项设置	起始值	长度值
On	字符串中的第一个字符处于位置 1。对负或零起始偏移的处理方法是：将其当作在字符串左侧填充了非字符。	正的 *length* 指定子串在开始位置右侧第 *length* 个字符处结束。 负的 *length* 会返回错误。
Off	字符串中的第一个字符处于位置 1。负起始位置指定到字符串结尾（而非开头）的字符数。 如果 *start* 为零且 length 为非负数，则使用 1 作为 start 的值。如果 start 为零且 *length* 为负数，则使用 -1 作为 start 的值。	正的 *length* 指定子串在开始位置右侧第 *length* 个字符处结束。 负 *length* 表示至多返回开始位置左侧（包括开始位置）的 *length* 个字符。

如果 *string-expression* 是二进制数据类型，则 SUBSTRING 函数的行为与 BYTE_SUBSTR 相同。

要获取字符串尾端的字符，应使用 RIGHT 函数。

此函数支持 NCHAR 输入和/或输出。如果输入字符串使用字符长度语义，就会在可能的情况下根据字符长度语义对返回值进行说明。

另请参见

- “BYTE_SUBSTR 函数 [String]”一节第 140 页
- “LEFT 函数 [String]”一节第 226 页
- “RIGHT 函数 [String]”一节第 282 页
- “CHARINDEX 函数 [String]”一节第 144 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

下表显示了 SUBSTRING 函数返回的值。

示例	结果
SUBSTRING('front yard', 1, 4)	fron
SUBSTRING('back yard', 6, 4)	yard
SUBSTR('abcdefgh', 0, -2)	如果 ansi_substring 为 On，返回错误
SUBSTR('abcdefgh', -2, 2)	如果 ansi_substring 为 On，返回空字符串

SUM 函数 [Aggregate]

返回每一组行的指定表达式总数。

语法 1

SUM(*expression* | **DISTINCT** *expression* **)**

语法 2

SUM(*expression* **) OVER (** *window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **expression**　要求和的对象。这通常是列名。
- **DISTINCT expression**　计算输入中 *expression* 的唯一值的总和。

返回值

INTEGER

DOUBLE

NUMERIC

注释

不包括指定表达式为 NULL 的行。

对于不包含任何行的组返回 NULL。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “COUNT 函数 [Aggregate]”一节第 157 页
- “AVG 函数 [Aggregate]”一节第 134 页

标准和兼容性

- **SQL/2003** 核心特性。语法 2 为特性 T611。

示例

以下语句返回值 3749146.740。

```
SELECT SUM( Salary )
FROM Employees;
```

SUSER_ID 函数 [System]

返回指定用户名的数字用户 ID。

语法

SUSER_ID([*user-name*] **)**

参数

- **user-name** 您在搜索的用户 ID 的用户名。

返回值

INT

注释

如果不指定 *user-name*，则返回当前用户的 ID。

另请参见

- “SUSER_NAME 函数 [System]”一节第 308 页
- “USER_ID 函数 [System]”一节第 321 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下语句返回 GROUPO 用户的 ID。

```
SELECT SUSER_ID( 'GROUPO' );
```

SUSER_NAME 函数 [System]

返回指定用户 ID 的用户名。

语法

SUSER_NAME([*user-id*] **)**

参数

- **user-id** 您在搜索的用户的用户 ID。

返回值

LONG VARCHAR

注释

如果不指定 *user-id*，则返回当前用户的用户名。

另请参见

- “SUSER_ID 函数 [System]”一节第 308 页
- “USER_NAME 函数 [System]”一节第 321 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下语句返回 ID 为 101 的用户的用户名。

```
SELECT SUSER_NAME( 101 );
```

TAN 函数 [Numeric]

返回一个数字的正切值。

语法

TAN(*numeric-expression* **)**

参数

- **numeric-expression** 角度（以弧度为单位）。

返回值

DOUBLE

注释

ATAN 和 TAN 函数互为逆运算。

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。

另请参见

- “COS 函数 [Numeric]”一节第 156 页
- “SIN 函数 [Numeric]”一节第 292 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 0.52 的正切值。

```
SELECT TAN( 0.52 );
```

TEXTPTR 函数 [Text and image]

返回指向指定文本列的第一页的 16 字节二进制指针。

语法

TEXTPTR(*column-name* **)**

参数

- **column-name** 文本列的名称。

返回值

BINARY

注释

包括此函数是为了与 Transact-SQL 兼容。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

使用 TEXTPTR 定位与作者的 blurbs 表中的 au_id 486-29-1786 关联的文本列 copy。

文本指针放置在局部变量 @val 中，并作为 readtext 命令的参数提供，此命令返回从第二个字节处（偏移 1）开始的 5 个字节。

```
DECLARE @val VARBINARY(16)
SELECT @val = TEXTPTR(copy)
FROM blurbs
WHERE au_id = "486-29-1786"
READTEXT blurbs.copy @val 1 5;
```


TO_CHAR 函数 [String]

将字符数据从任何支持的字符集转换为数据库的 CHAR 字符集。

语法

TO_CHAR(*string-expression* **[,** *source-charset-name* **])**

参数

- **string-expression** 要转换的字符串。
- **source-charset-name** 字符串的字符集。

返回值

LONG VARCHAR

注释

如果指定了 *source-charset-name*，则此函数等效于：

```
CAST( CSCONVERT( CAST( string-expression AS BINARY ),
 'db_charset', source-charset-name )
   AS CHAR );
```

有关 db_charset 的详细信息，请参见“CSCONVERT 函数 [String]”一节第 161 页。

如果未指定 *source-charset-name*，则此函数等效于：

```
CAST( string-expression AS CHAR );
```

另请参见

- “建议的字符集和归类”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CONNECTION_EXTENDED_PROPERTY 函数 [String]”一节第 149 页
- “CSCONVERT 函数 [String]”一节第 161 页
- “NCHAR 函数 [String]”一节第 242 页
- “TO_NCHAR 函数 [String]”一节第 312 页
- “UNICODE 函数 [String]”一节第 318 页
- “UNISTR 函数 [String]”一节第 319 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

如果有包含采用 cp850 字符集的数据的 BINARY 值，则以下语句会将数据转换为 CHAR 字符集和数据类型：

```
SELECT TO_CHAR( 'cp850_data', 'cp850' );
```

TO_NCHAR 函数 [String]

将字符数据从任何支持的字符集转换为 NCHAR 字符集。

语法

TO_NCHAR(*string-expression* **[,** *source-charset-name* **])**

参数

- **string-expression** 要转换的字符串
- **source-charset-name** 字符串的字符集。

返回值

LONG NVARCHAR

注释

如果指定了 *source-charset-name*，则此函数等效于：

```
CAST( CSCONVERT( CAST( string-expression AS BINARY ),
 'nchar_charset', source-charset-name )
 AS NCHAR );
```

有关 nchar_charset 的详细信息，请参见“CSCONVERT 函数 [String]”一节第 161 页。

如果未提供 *source-charset-name*，则此函数等效于：

```
CAST( string-expression AS NCHAR );
```

另请参见

- “建议的字符集和归类”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CONNECTION_EXTENDED_PROPERTY 函数 [String]”一节第 149 页
- “CSCONVERT 函数 [String]”一节第 161 页
- “NCHAR 函数 [String]”一节第 242 页
- “TO_CHAR 函数 [String]”一节第 311 页
- “UNICODE 函数 [String]”一节第 318 页
- “UNISTR 函数 [String]”一节第 319 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

如果有包含采用 cp850 字符集的数据的 BINARY 值，则以下示例会将数据转换为 NCHAR 字符集和数据类型：

```
SELECT TO_NCHAR( 'cp850_data', 'cp850' );
```


TODAY 函数 [Date and time]

返回当前日期。

语法

TODAY(*)

返回值

DATE

注释

使用此语法代替以前的 CURRENT DATE 函数。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句根据系统时钟返回当前日期。

```
SELECT TODAY( * );
SELECT CURRENT DATE;
```

TRACEBACK 函数 [Miscellaneous]

返回字符串，其中包含在最近的异常（错误）发生时执行的过程和触发器的跟踪信息。

语法

TRACEBACK(*)

返回值

LONG VARCHAR

注释

这对于调试过程和触发器很有用。

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

使用 traceback 函数的方法是，在过程执行中发生错误后输入以下语句：

```
SELECT TRACEBACK( * );
```

TRACED_PLAN 函数 [Miscellaneous]

Sybase Central 将此函数用于为使用跟踪数据的查询生成图形式计划。

语法

TRACED_PLAN(*logging_session_id*, *query_id* **)**

参数

- **logging_session_id** 与 *query_id* 结合使用时，此 INTEGER 参数标识 sa_diagnostic_query 表中要为其生成计划的行。
- **query_id** 与 *logging_session_id* 结合使用时，此 INTEGER 参数标识 sa_diagnostic_query 表中要为其生成计划的行。

返回值

LONG VARCHAR

注释

此函数供 Sybase Central 使用。

另请参见

- “sa_diagnostic_query 表”一节第 774 页

标准和兼容性

- **SQL/2003** 服务商扩充。

TRANSACTSQL 函数 [Miscellaneous]

接受 Watcom-SQL 语句并用 Transact-SQL 方言重写。

语法

TRANSACTSQL(*sql-statement-string* **)**

参数

- **sql-statement-string** 要以 Transact-SQL 重写的 SQL 语句。

返回值

LONG VARCHAR

另请参见

- “SQLDIALECT 函数 [Miscellaneous]”一节第 297 页
- “WATCOMSQL 函数 [Miscellaneous]”一节第 326 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回字符串 'SELECT EmployeeName=empl_name FROM Employees'。

```
SELECT TRANSACTSQL( 'SELECT empl_name as EmployeeName FROM Employees' ) FROM
dummy;
```

TRIM 函数 [String]

删除字符串中的前导和尾随空白。

语法

TRIM(*string-expression* **)**

参数

- **string-expression** 要剪裁的字符串。

返回值

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

注释

此函数支持 NCHAR 输入和/或输出。

另请参见

- “LTRIM 函数 [String]”一节第 234 页
- “RTRIM 函数 [String]”一节第 286 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** TRIM 函数是 SQL/2003 的一个核心特性。

  SQL Anywhere 不支持附加参数 *trim specification* 和 *trim character*（SQL/2003 中定义了这些参数）。TRIM 的 SQL Anywhere 实现对应于 BOTH 的 TRIM 说明。

  针对其它由 SQL/2003 标准定义的 TRIM 说明（LEADING 和 TRAILING），SQL Anywhere 分别提供了 LTRIM 和 RTRIM 函数。

示例

以下语句返回没有前导和尾随空白的值 chocolate。

```
SELECT TRIM( '   chocolate   ' );
```

TRUNCNUM 函数 [Numeric]

在小数点后的指定位数截断数字。

语法

{ **TRUNCNUM** | **"TRUNCATE"** }(*numeric-expression*, *integer-expression*)

参数

- **numeric-expression** 要截断的数字。
- **integer-expression** 正整数指定小数点右边舍入到的有效位数。负数表达式指定小数点左边舍入到的有效位数。

返回值

NUMERIC

注释

截断数字时应使用 TRUNCNUM 函数而不是 TRUNCATE 函数。

不建议使用 TRUNCATE 语句，因为词语 truncate 是关键字，因此需要将 quoted_identifier 选项设置为 OFF，或在词语 TRUNCATE 两侧加引号。

另请参见

- “ROUND 函数 [Numeric]”一节第 283 页
- “quoted_identifier 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 600。

```
SELECT TRUNCNUM( 655, -2 );
```

以下语句返回值 655.340。

```
SELECT TRUNCNUM( 655.348, 2 );
```

TSEQUAL 函数 [System]（不再支持）

比较两个时间戳的值并返回它们是否相同。

语法

TSEQUAL (*timestamp1*, *timestamp2* **)**

参数

- **timestamp1** 时间戳表达式。
- **timestamp2** 时间戳表达式。

返回值

BIT

注释

TSEQUAL 函数只能用于 WHERE 子句中，并且大部分情况下作为 UPDATE 语句的一部分来使用。

如果 *timestamp1* 与 *timestamp2* 相等，则说明某行在读取后曾被更改。如果该行被更改了，则其时间戳会被修改过并且 TSEQUAL 函数会返回 FALSE。如果 TSEQUAL 函数返回 FALSE，应用程序确定没有更新过的行并假定该行被其它用户修改过。更新的行将重新读取。

可以使用 TSEQUAL 函数确定某行在读取后是否被更改过。

另请参见

- “时间戳列的数据类型”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “TIMESTAMP 特殊值”一节第 61 页
- “特殊的 Transact-SQL 时间戳列和数据类型”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

假设创建一个 TIMESTAMP 列 Products.LastUpdated 来存储该行上次更新的时间戳。以下的 UPDATE 语句使用 TSEQUAL 函数来确定是否更新该行。如果 LastUpdated 的值为 '2010/12/25 11:08:34.173226'，将更新该行。

```
UPDATE Products
SET Color = 'Yellow'
WHERE ID = '300'
AND TSEQUAL( LastUpdated, '2010/12/25 11:08:34.173226' );
```

UCASE 函数 [String]

将字符串中的所有字符转换成大写形式。此函数与 UPPER 函数相同。

语法

UCASE(*string-expression* **)**

参数

- **string-expression** 要转换成大写形式的字符串。

返回值

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

另请参见

- “UPPER 函数 [String]”一节第 320 页
- “LCASE 函数 [String]”一节第 225 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句返回值 CHOCOLATE。

```
SELECT UCASE( 'ChocoLate' );
```

UNICODE 函数 [String]

返回一个整数，它包含字符串中第一个字符的 Unicode 代码点；如果第一个字符不是有效的编码，则返回 NULL。

语法

UNICODE(*nchar-string-expression* **)**

参数

- **nchar-string-expression** 第一个字符将转换为整数的 NCHAR 字符串。

返回值

INT

另请参见

- “CONNECTION_EXTENDED_PROPERTY 函数 [String]”一节第 149 页
- “NCHAR 函数 [String]”一节第 242 页
- “TO_CHAR 函数 [String]”一节第 311 页
- “TO_NCHAR 函数 [String]”一节第 312 页
- “UNISTR 函数 [String]”一节第 319 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例返回整数 65536：


```
SELECT UNICODE(UNISTR( '\u010000data' ));
```

UNISTR 函数 [String]

将包含字符和 Unicode 转义序列的字符串转换为 NCHAR 字符串。

语法

UNISTR(*string-expression* **)**

参数

- **string-expression** 要转换的字符串。

返回值

NVARCHAR

LONG NVARCHAR

注释

UNISTR 函数允许使用 Unicode 字符，这些字符无法以 SQL 语句使用的 CHAR 字符集进行表示。例如，在英语环境中，UNISTR 函数可用于加入中文字符。

UNISTR 函数提供了与 N" 常量类似的功能，不过，UNISTR 函数允许使用 Unicode 字符以及 CHAR 字符集的字符，而 N" 常量仅允许 CHAR 字符集的字符。

string-expression 包含字符和 Unicode 转义序列。Unicode 转义序列的形式为 \uXXXX 或 \uXXXXXX，其中 X 是十六进制数字。UNISTR 函数会将每个字符及每个 Unicode 转义序列转换为对应的 Unicode 字符。

如果使用了 6 位数 Unicode 转义序列，则其值不得超过 10FFFF，即最大的 Unicode 代码点。\u234567 这样的序列不是 6 位数 Unicode 转义序列。它是后跟字符 6 和 7 的 4 位数序列 \u2345。

如果两个相邻的 Unicode 转义序列形成一个 UTF-16 代理对，则会在输出中将它们合并成一个 Unicode 字符。

另请参见

- “CONNECTION_EXTENDED_PROPERTY 函数 [String]”一节第 149 页
- “NCHAR 函数 [String]”一节第 242 页
- “TO_CHAR 函数 [String]”一节第 311 页
- “TO_NCHAR 函数 [String]”一节第 312 页
- “UNICODE 函数 [String]”一节第 318 页
- “字符串”一节第 9 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例返回字符串 Hello。


```
SELECT UNISTR( 'Hel\u006c\u006F' );
```

以下示例将 UTF-16 代理对 D800-DF02 合并到 Unicode 代码点 10302 中。

```
SELECT UNISTR( '\uD800\uDF02' );
```

该示例等同于前一个示例。

```
SELECT UNISTR( '\u010302' );
```

UPPER 函数 [String]

将字符串中的所有字符转换成大写形式。此函数与 UCASE 函数完全相同。

语法

UPPER(*string-expression* **)**

参数

- **string-expression** 要转换成大写形式的字符串。

返回值

VARCHAR

NVARCHAR

LONG VARCHAR

LONG NVARCHAR

注释

UCASE 函数类似于 UPPER 函数。

另请参见

- “UCASE 函数 [String]”一节第 317 页
- “LCASE 函数 [String]”一节第 225 页
- “LOWER 函数 [String]”一节第 233 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句返回值 CHOCOLATE。

```
SELECT UPPER( 'ChocoLate' );
```


USER_ID 函数 [System]

返回指定用户名的数字用户 ID。

语法

USER_ID([*user-name*] **)**

参数

- **user-name** 您在搜索的用户 ID 的用户名。

返回值

INT

注释

如果不指定 *user-name*，则返回当前用户的 ID。

另请参见

- “USER_NAME 函数 [System]”一节第 321 页
- “SUSER_ID 函数 [System]”一节第 308 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下语句返回 GROUPO 用户 ID。

```
SELECT USER_ID( 'GROUPO' );
```

USER_NAME 函数 [System]

返回指定用户 ID 的用户名。

语法

USER_NAME([*user-id*] **)**

参数

- **user-id** 您在搜索的用户的用户 ID。

返回值

LONG VARCHAR

注释

如果不指定 *user-id*，则返回当前用户的用户名。

另请参见

- “USER_ID 函数 [System]”一节第 321 页
- “SUSER_NAME 函数 [System]”一节第 308 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回 ID 为 101 的用户的用户名。

```
SELECT USER_NAME( 101 );
```

UUIDTOSTR 函数 [String]

将唯一标识符值（UUID，又称 GUID）转换为字符串值。

较新的数据库中不需要
在使用 9.0.2 以前的版本创建的数据库中，UNIQUEIDENTIFIER 数据类型定义为用户定义的数据类型，需要使用 STRTOUUID 和 UUIDTOSTR 函数在 UUID 值的二进制与字符串表示之间进行转换。

在使用 9.0.2 或更高版本创建的数据库中，UNIQUEIDENTIFIER 数据类型已更改为本地数据类型，SQL Anywhere 会在需要的时候进行转换。在这些版本中不需要使用 STRTOUUID 和 UUIDTOSTR 函数。

有关详细信息，请参见“UNIQUEIDENTIFIER 数据类型”一节第 102 页。

语法

UUIDTOSTR(*uuid-expression* **)**

参数

- **uuid-expression** 唯一标识符值。

返回值

VARCHAR

注释

将唯一标识符转换为 *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx*（其中 x 是十六进制数字）格式的字符串值。如果此二进制值不是有效的唯一标识符，则返回 NULL。

如果您想查看 UUID 值，则可使用此函数。

另请参见

- “NEWID 函数 [Miscellaneous]”一节第 242 页
- “STRTOUUID 函数 [String]”一节第 304 页
- “字符串函数”一节第 125 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句创建一个包含两列的 mytab 表。列 pk 的数据类型是唯一标识符，列 c1 的数据类型是整数。随后该语句在列 c1 中插入两行（值分别为 1 和 2）。

```
CREATE TABLE mytab(
    pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
    c1 INT );
INSERT INTO mytab( c1 ) values ( 1 );
INSERT INTO mytab( c1 ) values ( 2 );
```

执行以下 SELECT 语句后，将返回新创建的表中的所有数据。

```
SELECT * FROM mytab;
```

您将看到一个包含两列、两行的表。为列 pk 显示的值将是二进制值。

若要将唯一标识符值转换为可读的格式，请执行以下的命令：

```
SELECT UUIDTOSTR(pk), c1 FROM mytab;
```

对于使用 9.0.2 或更高版本创建的数据库，不需要 UUIDTOSTR 函数。

VAR_POP 函数 [Aggregate]

计算由数字表达式组成的总体的统计方差，类型为 DOUBLE。

语法 1

VAR_POP(*numeric-expression* **)**

语法 2

VAR_POP(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **numeric-expression** 一个表达式，其基于总体的方差以一组行为基础进行计算。此表达式通常是列名。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。

根据以下公式计算 *numeric-expression* (x) 基于总体的方差 (s^2)：

$s^2 = (1/N) * SUM(x_I - mean(x))^2$

此方差不包括 *numeric-expression* 是 NULL 的行。对于不包含任何行的组，它返回 NULL。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “集合函数”一节第 118 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性 (T611)。

示例

以下语句列出不同时间段每个订单的产品数目的平均值和方差：

```
SELECT YEAR( ShipDate ) AS Year,
    QUARTER( ShipDate ) AS Quarter,
    AVG( Quantity ) AS Average,
    VAR_POP( quantity ) AS Variance
FROM SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	203.9021...
2000	2	27.050847	225.8109...
...	...	...	...

VAR_SAMP 函数 [Aggregate]

计算由数字表达式组成的样本的统计方差，类型为 DOUBLE。

语法 1

VAR_SAMP(*numeric-expression* **)**

语法 2

VAR_SAMP(*numeric-expression* **) OVER (** *window-spec* **)**

window-spec：请参见下面“注释”部分对“语法 2”的说明

参数

- **numeric-expression** 一个表达式，其基于样本的方差以一组行为基础进行计算。此表达式通常是列名。

返回值

DOUBLE

注释

此函数将其参数转换为 DOUBLE，以双精度浮点执行计算，然后返回 DOUBLE 值作为结果。

根据以下公式计算 *numeric-expression* (x) 的方差 (s^2)，该公式假定数据呈正态分布：

$s^2 = (1/(N - 1)) * SUM(x_I - mean(x))^2$

此方差不包括 *numeric-expression* 是 NULL 的行。对于包含 0 或 1 行的组，它返回 NULL。

语法 2 表示作为 SELECT 语句中的窗口函数的用法。因此，可以在函数语法中指定 *window-spec* 的元素（内置），也可以将这些元素与 SELECT 语句中的 WINDOW 子句一并加以指定。请参见“WINDOW 子句”一节第 749 页中提供的 *window-spec* 定义。

有关在 SELECT 语句中使用窗口函数的详细信息（包括工作示例），请参见“窗口函数”一节《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “集合函数”一节第 118 页
- “VARIANCE 函数 [Aggregate]”一节第 326 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL 基础特性。VARIANCE 语法是一个服务商扩充。

示例

以下语句列出不同时间段每个订单的产品数目的平均值和方差：

```
SELECT YEAR( ShipDate ) AS Year,
    QUARTER( ShipDate ) AS Quarter,
    AVG( Quantity ) AS Average,
    VAR_SAMP( quantity ) AS Variance
FROM SalesOrderItems
GROUP BY Year, Quarter
ORDER BY Year, Quarter;
```

Year	Quarter	Average	Variance
2000	1	25.775148	205.1158...
2000	2	27.050847	227.0939...
...	...	...	...

VARIANCE 函数 [Aggregate]

VAR_SAMP 的别名。请参见“VAR_SAMP 函数 [Aggregate]”一节第 324 页。

VAREXISTS 函数 [Miscellaneous]

如果已使用给定名创建或声明了用户定义变量，则返回 1。如果未创建这样的变量，则返回 0。

语法

VAREXISTS(*variable-name-string* **)**

参数

- **variable-name-string** 要测试的变量名（字符串形式）。

返回值

INT

另请参见

- “CREATE VARIABLE 语句”一节第 517 页
- “DECLARE 语句”一节第 523 页
- “IF 语句”一节第 607 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下 IF 语句创建一个名为 start_time 的变量（如果尚未创建或声明该变量）。之后便可安全地使用该变量。

```
IF VAREXISTS( 'start_time' ) = 0 THEN
    CREATE VARIABLE start_time TIMESTAMP;
END IF;
SET start_time = current timestamp;
```

WATCOMSQL 函数 [Miscellaneous]

接受 Transact-SQL 语句并以 Watcom-SQL 方言重写该语句。将现有 Adaptive Server Enterprise 存储过程转换成 Watcom SQL 语法时，此函数会有用处。

语法

WATCOMSQL(*sql-statement-string* **)**

参数

- **sql-statement-string** 函数用来确定其方言的 SQL 语句。

返回值

LONG VARCHAR

另请参见

- “SQLDIALECT 函数 [Miscellaneous]”一节第 297 页
- “TRANSACTSQL 函数 [Miscellaneous]”一节第 314 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回字符串 'SELECT empl_name AS EmployeeName FROM Employees'。

```
SELECT WATCOMSQL( 'SELECT EmployeeName=empl_name FROM Employees' ) FROM dummy;
```

WEEKS 函数 [Date and time]

返回两个日期之间的周数。

语法 1

WEEKS([*datetime-expression*,] *datetime-expression* **)**

语法 2

WEEKS(*datetime-expression*, *integer-expression* **)**

参数

- **datetime-expression** 日期和时间。
- **integer-expression** 要添加到 *datetime-expression* 中的周数。如果 *integer-expression* 是负数，则从日期时间值中减去相应的周数。如果提供 *integer-expression*，则必须将 *datetime-expression* 显式地转换为 DATETIME。

返回值

语法 1 返回 INTEGER。

语法 2 返回 TIMESTAMP。

注释

如果给定一个单独的日期（语法 1），WEEKS 函数将返回自 0000-02-29 后的周数。

如果给定两个日期（语法 1），WEEKS 函数将返回两者之间的周数。WEEKS 函数与 DATEDIFF 函数类似，但它们用于计算两个不同日期之间周数的方法不同，而且可以返回不同的结果。WEEKS 的返回值的确定方式是，将两个日期间的天数除以七，然后向下舍入；然而，DATEDIFF 使用的是周边界数。这会导致返回的值不同。例如，如果第一个日期为星期五，第二个日期为下一个星期一，则 WEEKS 函数会返回差值 0，DATEDIFF 函数会返回差值 1。当两种方法相对彼此而言均无优势时，则在 WEEKS 和 DATEDIFF 之间进行选择时应考虑到差值。

有关 DATEDIFF 函数的详细信息，请参见“DATEDIFF 函数 [Date and time]”一节第 166 页。

如果给定一个日期和一个整数（语法 2），WEEKS 函数会将该整周数与指定日期相加。此函数与 DATEADD 函数类似。

有关 DATEADD 函数的详细信息，请参见“DATEADD 函数 [Date and time]”一节第 165 页。

另请参见

有关转换数据类型的信息，请参见“CAST 函数 [Data type conversion]”一节第 141 页。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 8，表示 2008-09-13 10:07:12 比 2008-07-13 06:07:12 晚八周。

```
SELECT WEEKS( '2008-07-13 06:07:12',
   '2008-09-13 10:07:12' );
```

以下语句返回值 104792，表示该日期比 0000-02-29 晚 104792 周。

```
SELECT WEEKS( '2008-07-13 06:07:12' );
```

以下语句返回时间戳 2008-06-16 21:05:07.0，表示该日期和时间比 2008-05-12 21:05:07 晚五周。

```
SELECT WEEKS( CAST( '2008-05-12 21:05:07'
AS TIMESTAMP ), 5);
```

WRITE_CLIENT_FILE 函数 [String]

在客户端计算机上创建并写入一个文件。

语法

WRITE_CLIENT_FILE(*filename*, *blob-expression* [, **'A'**] **)**

参数

- **filename** 客户端计算机的文件名。将在客户端计算机上相对于客户端应用程序的当前工作目录来解析此文件名称。
- **blob-expression** 写入到客户端计算机上 *filename* 中的二进制字符串。
- **A** 缺省情况下，如果文件已经存在，则将覆盖它。如果要将数附加到现有数据之后，则指定 'A'。如果文件尚不存在，并且指定了 'A'，文件仍将创建。

返回值

INT

注释

数据库服务器将 *filename* 从数据库字符集转换为客户端字符集。然后在客户端计算机上，*filename* 将转换为操作系统字符集。

因为数据是二进制字符串，如果想要数据采用特定字符集，或者要将其压缩或加密，则必须在将其发送至 WRITE_CLIENT_FILE 函数前对其进行这些操作。

文件的读取由客户端软件库执行，数据的传输则使用命令序列通信协议来完成。

权限

当写入到客户端计算机的文件中时：

- 需要 WRITECLIENTFILE 权限。请参见“WRITECLIENTFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
- 客户端应用程序必须拥有所要写入计算机的写权限。
- 必须启用 allow_write_client_file 数据库选项。请参见“allow_write_client_file 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- 必须启用 write_client_file 受保护的功能。请参见“-sf 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

另请参见

- “访问客户端计算机上的数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “WRITECLIENTFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》
- “UNLOAD 语句”一节第 731 页
- “CSCONVERT 函数 [String]”一节第 161 页
- “DECOMPRESS 函数 [String]”一节第 177 页
- “DECRYPT 函数 [String]”一节第 179 页

标准和兼容性

- **SQL/2003** 服务商扩充。

XMLAGG 函数 [Aggregate]

从 XML 值的集合生成一个 XML 元素林。

语法

XMLAGG(*value-expression* [**ORDER BY** *order-by-expression*] **)**

参数

- **value-expression** XML 值。始终将元素内容转义，除非数据类型为 XML。*order-by-expression* 会对该函数返回的元素排序。
- **order-by-expression** 用以对 XML 元素排序的表达式，根据该表达式的值对 XML 元素排序。

返回值

XML

注释

结果中任何为 NULL 的值都将被忽略。如果所有的输入都为 NULL，或输入 0 行数据，则结果为 NULL。如果您要求格式正确的 XML 文档，则必须确保编写查询以使生成的 XML 只包含一个根元素。

当执行一个包含 XMLAGG 的查询时，BINARY、LONG BINARY、IMAGE 和 VARBINARY 列中的数据将被自动以 base64 编码格式返回。

有关使用带有 ORDER BY 子句的 XMLAGG 函数的查询示例，请参见“使用 XMLAGG 函数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “使用 XMLAGG 函数”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- SQL/XML 标准草案的一部分。

示例

以下示例生成了显示每个客户所发出的订单的 XML 文档。

```
SELECT XMLELEMENT( NAME "order",
                   XMLATTRIBUTES( ID AS order_id ),
                      ( SELECT XMLAGG(
                          XMLELEMENT(
                            NAME "Products",
                            XMLATTRIBUTES( ProductID, Quantity AS
"quantity_shipped" ) ) )
                           FROM SalesOrderItems soi
                           WHERE soi.ID = so.ID
                          )
                     ) AS products_ordered
FROM SalesOrders so
ORDER BY so.ID;
```

XMLCONCAT 函数 [String]

生成一个 XML 元素林。

语法

XMLCONCAT(*xml-value* **[,** ... **])**

参数

- **xml-value**　要连接的 XML 值。

返回值

XML

注释

生成一个 XML 元素林。在一个未被分析的 XML 文档中，元素林是指文档中的多个根节点。NULL 将被从结果中忽略。如果所有的值均为 NULL，则返回 NULL。XMLCONCAT 函数不会检查参数是否具有处理指令。如果您要求格式正确的 XML 文档，则必须确保编写查询以生成单个根元素。

始终将元素内容转义，除非数据类型为 XML。执行包含 XMLCONCAT 函数的查询时，将自动以 base64 编码格式返回 BINARY、LONG BINARY、IMAGE 和 VARBINARY 列中的数据。

另请参见

- “使用 XMLCONCAT 函数”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “XMLELEMENT 函数 [String]”一节第 331 页
- “XMLFOREST 函数 [String]”一节第 333 页
- “字符串函数”一节第 125 页

标准和兼容性

- SQL/XML 标准草案的一部分。

示例

以下查询生成每个客户的 <CustomerID>、<cust_fname> 和 <cust_lname> 元素。

```
SELECT XMLCONCAT( XMLELEMENT ( NAME CustomerID, ID ),
                  XMLELEMENT( NAME cust_fname, GivenName ),
                  XMLELEMENT( NAME cust_lname, Surname )
                ) AS "Customer Information"
FROM Customers
WHERE ID < 120;
```

XMLELEMENT 函数 [String]

在查询中生成一个 XML 元素。

语法

XMLELEMENT({ **NAME** *element-name-expression* | *string-expression* }
  [, **XMLATTRIBUTES (** *attribute-value-expression* [**AS** *attribute-name*],... **)**]
  [, *element-content-expression*,...]
)

参数

- **element-name-expression** 一个标识符。为每一行生成一个与标识符同名的 XML 元素。
- **attribute-value-expression** 元素的一个特性。此可选参数允许指定所生成的元素的属性。此参数指定属性名称和内容。如果 *attribute-value-expression* 为某一列名称，则属性名称缺省为列名称。可通过指定 *attribute-name argument* 来更改特性名。
- **element-content-expression** 元素的内容。可以是任意字符串表达式。可以指定任意数量的 *element-content-expression* 参数，这些参数将被连接到一起。例如，以下 SELECT 语句返回值 <x>abcdef</x>：

  ```
  SELECT XMLELEMENT( NAME x, 'abc', 'def' );
  ```

返回值

XML

注释

结果中将会忽略 NULL 元素值和 NULL 特性值。元素与特性名的字母大小写均依据查询来确定。

始终将元素内容转义，除非数据类型为 XML。无效的元素和属性名称也加上引号。例如，请看以下语句：

```
SELECT XMLELEMENT('H1', f_get_page_heading() );
```

如果函数 f_get_page_heading 定义为 RETURNS LONG VARCHAR 或 RETURNS VARCHAR(1000)，则结果采用 HTML 编码：

```
CREATE FUNCTION f_get_page_heading() RETURNS LONG VARCHAR
   BEGIN
      RETURN ('<B>My Heading</B>');
   END;
```

以上的 SELECT 语句返回以下结果：

```
<H1>&lt;B&gt;My Heading&lt;/B&gt;</H1>
```

如果此函数声明为 RETURNS XML，则以上 SELECT 语句返回以下结果：

```
<H1><B>My Heading</B></H1>
```

有关添加引号和 XMLELEMENT 函数的详细信息，请参见“无效的名称和 SQL/XML”一节《SQL Anywhere 服务器 - SQL 的用法》。

XMLELEMENT 函数可通过嵌套使用来创建层次。如果要返回处于文档层次中同一级别的不同元素，请使用 XMLFOREST 函数。

有关详细信息，请参见“XMLFOREST 函数 [String]”一节第 333 页。

执行包含 XMLELEMENT 函数的查询时，将自动以 base64 编码格式返回 BINARY、LONG BINARY、IMAGE 和 VARBINARY 列中的数据。

另请参见

- “使用 XMLELEMENT 函数”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “XMLCONCAT 函数 [String]”一节第 330 页
- “XMLFOREST 函数 [String]”一节第 333 页
- “字符串函数”一节第 125 页

标准和兼容性

- SQL/XML 标准草案的一部分。
- 忽略 NAME 关键字和使用字符串表达式作为第一个参数是一个服务商扩充。

示例

以下示例为结果集中的每个产品生成一个 <item_name> 元素，其中产品名为元素的内容。


```
SELECT ID, XMLELEMENT( NAME item_name, p.Name )
FROM Products p
WHERE ID > 400;
```

以下示例返回 <A HREF="http://www.ianywhere.com/" TARGET="_top">iAnywhere web site</A>:

```
SELECT XMLELEMENT(
   'A',
   XMLATTRIBUTES( 'http://www.ianywhere.com/'
      AS "HREF", '_top' AS "TARGET"),
   'iAnywhere web site'
);
```

以下示例返回 <table><tbody><tr align="center" valign="top"><td>Cell 1 info</td><td>Cell 2 info</td></tr></tbody></table>:

```
SELECT XMLELEMENT( name "table",
         XMLELEMENT( name "tbody",
           XMLELEMENT( name "tr",
             XMLATTRIBUTES('center' AS "align", 'top' AS "valign"),
             XMLELEMENT( name "td", 'Cell 1 info' ),
             XMLELEMENT( name "td", 'Cell 2 info' )
           )
         )
       );
```

以下示例返回 '<x>abcdef</x>','<custom_element>abcdef</custom_element>':

```
CREATE VARIABLE @my_element_name VARCHAR(200);
SET @my_element_name = 'custom_element';
SELECT XMLELEMENT( NAME x, 'abc', 'def' ),
   XMLELEMENT( @my_element_name,'abc', 'def' );
```

XMLFOREST 函数 [String]

生成一个 XML 元素林。

语法

XMLFOREST(*element-content-expression* [**AS** *element-name*],... **)**

参数

- **element-content-expression** 一个字符串。对每个所指定的 *element-content-expression* 参数生成一个元素。*element-content-expression* 值将成为元素的内容。例如，如果为此参数指定 Employees 表中的 EmployeeID 列，则将为此表中的每个值生成一个包含 EmployeeID 值的 <EmployeeID> 元素。

  如果要给元素指派 *element-content-expression* 以外的名称，请指定 *element-name* 参数，否则元素名在缺省情况下将是 *element-content-expression* 的名称。

返回值

XML

注释

生成一个 XML 元素林。在一个未被分析的 XML 文档中，元素林是指文档中的多个根节点。当 XMLFOREST 函数的所有参数都为 NULL 时，返回 NULL 值。当仅有某些值为 NULL 时，NULL 值将被从结果中忽略。始终将元素内容加上引号，除非数据类型为 XML。不能使用 XMLFOREST 函数来指定特性。如果要指定所生成元素的特性，请使用 XMLELEMENT 函数。

有关 XMLELEMENT 函数的详细信息，请参见“XMLELEMENT 函数 [String]”一节第 331 页。

始终将元素名称转义，除非数据类型为 XML。

如果您要求格式正确的 XML 文档，则必须确保编写查询以生成单个根元素。

当执行一个包含 XMLFOREST 的查询时，BINARY、LONG BINARY、IMAGE 和 VARBINARY 列中的数据将被自动以 base64 编码格式返回。

另请参见

- “使用 XMLFOREST 函数”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “XMLELEMENT 函数 [String]”一节第 331 页
- “XMLCONCAT 函数 [String]”一节第 330 页
- “字符串函数”一节第 125 页

标准和兼容性

- SQL/XML 标准草案的一部分。

示例

以下示例为每个职员的名和姓生成了一个 XML 元素。

```
SELECT EmployeeID,
       XMLFOREST( GivenName, Surname )
       AS "Employee Name"
FROM Employees;
```

XMLGEN 函数 [String]

基于 XQuery 构造函数生成一个 XML 值。

语法

XMLGEN(*xquery-constructor*, *content-expression* [**AS** *variable-name*],... **)**

参数

- **xquery-constructor** XQuery 构造程序。XQuery 构造函数为在 XQuery 语言中定义的一项。它给出了基于 XQuery 表达式构造 XML 元素的语法。*xquery-constructor* 参数必须为具有一个或多个变量引用的格式正确的 XML 文档。变量引用要放在大括号内，并且带有前缀 $，两侧没有空格。例如：

```
SELECT XMLGEN( '<a>{$x}</a>', 1 AS x );
```


- **content-expression** 一个变量。可以指定多个 *content-expression* 参数。此可选的 *variable-name* 参数用于对变量命名。例如，

```
SELECT XMLGEN( '<emp EmployeeID="{$EmployeeID}"><StartDate>{$x}</
StartDate></emp>',
               EmployeeID, StartDate
               AS x )
FROM Employees;
```

返回值

XML

注释

XMLGEN 函数不支持 XQuery 说明中定义的计算构造函数。

执行包含 XMLGEN 函数的查询时，将自动以 base64 编码格式返回 BINARY、LONG BINARY、IMAGE 和 VARBINARY 列中的数据。

始终将元素内容转义，除非数据类型为 XML。非法的 XML 元素和特性名也会被转义。

有关转义和 XMLGEN 函数的信息，请参见“无效的名称和 SQL/XML”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “使用 XMLGEN 函数”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “字符串函数”一节第 125 页

标准和兼容性

- SQL/XML 标准草案的一部分。

示例

以下示例会为每个雇员生成 <emp>、<Surname>、<GivenName> 和 <StartDate> 元素。

```
SELECT XMLGEN( '<emp EmployeeID="{$EmployeeID}">
                   <Surname>="{$Surname}"</Surname>
                   <GivenName>="{$GivenName}"</GivenName>
                   <StartDate>="{$StartDate}"</StartDate>
               </emp>',
               EmployeeID,
               Surname,
               GivenName,
               StartDate
             ) AS employee_list
FROM Employees;
```

YEAR 函数 [Date and time]

以时间戳值为参数，返回该时间戳所指定的年份。

语法

YEAR(*datetime-expression* **)**

参数

- **datetime-expression** 一个日期、时间或时间戳。

返回值

SMALLINT

注释

该值作为 SMALL INT 返回。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例返回值 2001。

```
SELECT YEAR( '2001-09-12' );
```

YEARS 函数 [Date and time]

给定两个日期时，此函数返回二者之间的整数年数。建议改用 DATEDIFF 函数。请参见“DATEDIFF 函数 [Date and time]”一节第 166 页。

给定一个日期时，此函数返回年份。建议改用 DATEPART 函数。请参见“DATEPART 函数 [Date and time]”一节第 169 页。

给定一个日期和一个整数时，此函数为指定日期添加整数年数。建议改用 DATEADD 函数。请参见“DATEADD 函数 [Date and time]”一节第 165 页。

语法 1

YEARS([*datetime-expression*,] *datetime-expression* **)**

语法 2

YEARS(*datetime-expression*, *integer-expression* **)**

参数

- **datetime-expression** 日期和时间。
- **integer-expression** 要添加到 *datetime-expression* 中的年数。如果 *integer-expression* 是负数，则从日期时间值中减去相应的年数。如果提供 *integer-expression*，则必须将 *datetime-expression* 显式地转换为日期时间数据类型。

  有关转换数据类型的信息，请参见“CAST 函数 [Data type conversion]”一节第 141 页。

返回值

语法 1 返回 INTEGER。语法 2 返回 TIMESTAMP。

注释

YEARS 值是从两个日期之间年中第一天的数目计算出的。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句都返回 -4。

```
SELECT YEARS( '1998-07-13 06:07:12',
              '1994-03-13 08:07:13' );

SELECT DATEDIFF( year,
   '1998-07-13 06:07:12',
   '1994-03-13 08:07:13' );
```

以下语句都返回 1998。

```
SELECT YEARS( '1998-07-13 06:07:12' )
SELECT DATEPART( year, '1998-07-13 06:07:12' );
```

以下语句返回给定日期 300 年后的日期。

```
SELECT YEARS( CAST( '1998-07-13 06:07:12' AS TIMESTAMP ), 300 )

SELECT DATEADD( year, 300, '1998-07-13 06:07:12' );
```

YMD 函数 [Date and time]

返回对应于给定年、月、日的日期值。值是 -32768 到 32767 的小整数。

语法

YMD(
integer-expression1,
integer-expression2,
integer-expression3 **)**

参数

- **integer-expression1** 年。
- **integer-expression2** 表示月份的数字。如果月份超出 1-12 的范围，则会调整年份。
- **integer-expression3** 天数。日可以是任意整数；日期可相应调整。

返回值

DATE

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句返回值 1998-06-12。

```
SELECT YMD( 1998, 06, 12 );
```

如果值超出其正常范围，则会调整日期。例如，以下语句返回值 2000-03-01。

```
SELECT YMD( 1999, 15, 1 );
```


SQL 语句

目录

使用 SQL 语句参考

本节介绍记录 SQL 语句时使用的一些约定。

SQL 语法中的常见元素

本节列出了在许多 SQL 语句的语法中出现的语言元素。

有关此处描述的元素的详细信息，请参见“标识符”一节第 8 页、“SQL 数据类型”第 75 页、“搜索条件”一节第 33 页、“表达式”一节第 16 页或“字符串”一节第 9 页。

- **column-name** 表示列名称的标识符。请参见“标识符”一节第 8 页。
- **condition** 取值为 TRUE、FALSE 或 UNKNOWN 的表达式。请参见“真值搜索条件”一节第 52 页。
- **connection-name** 表示活动连接名称的字符串。请参见“SQL Anywhere 数据库连接” 《SQL Anywhere 服务器 - 数据库管理》。
- **data-type** 存储数据类型。请参见“SQL 数据类型”第 75 页。
- **expression** 表达式。语法中表达式的常见示例是列名。请参见“表达式”一节第 16 页。
- **filename** 包含文件名的字符串。
- **hostvar** C 语言变量，声明为前面有一个冒号的主机变量。请参见“使用主机变量”一节 《SQL Anywhere 服务器 - 编程》。
- **indicator-variable** 类型为 **short int** 的另一个主机变量，紧跟在常规主机变量之后。它的前面也必须有一个冒号。指示符变量用于向数据库传入和从数据库传出 NULL 值。请参见“使用主机变量”一节 《SQL Anywhere 服务器 - 编程》。
- **materialized-view-name** 表示实例化视图名称的标识符。请参见“使用实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **数字** 任意数字序列，后跟可选的小数部分，前面有一个可选的负号。数字后面还可以有一个 E 和一个指数。例如，

```
42
-4.038
.001
3.4e10
1e-10
```

- **owner** 表示拥有数据库对象的用户 ID 的标识符。请参见“由于拥有某一对象而获得的权限”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **query-block** 查询块是简单的查询表达式，或带有 ORDER BY 子句的查询表达式。
- **query-expression** 查询表达式可以是 SELECT、UNION、INTERSECT 或 EXCEPT 块（即不包含 ORDER BY、WITH、FOR、FOR XML 或 OPTION 子句的语句），或这些块的任何组合。
- **role-name** 表示外键角色名称的标识符。概念性数据库建模中从一个角度描述某种关系的动词或短语。您可以用两个角色来描述每种关系。例如，"包含" 和 "隶属于" 便是角色。

- **savepoint-name** 表示保存点名称的标识符。请参见“事务内的保存点”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **search-condition** 取值为 TRUE、FALSE 或 UNKNOWN 的条件。请参见“搜索条件”一节第 33 页。
- **special-value** “特殊值”一节第 56 页中描述的特殊值之一。
- **statement-label** 表示循环或复合语句标签的标识符。请参见“控制语句”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **statement-list** SQL 语句的列表，每个都以分号结束。
- **string-expression** 解析为字符串的表达式。请参见“表达式”一节第 16 页。
- **table-list** 表名列表，可以包含相关名。请参见“FROM 子句”一节第 582 页和“键连接”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **table-name** 表示表名的标识符。请参见“标识符”一节第 8 页。
- **userid** 表示用户名的标识符。请参见“标识符”一节第 8 页。
- **variable-name** 表示变量名的标识符。请参见“变量”一节第 64 页。
- **window-name** 表示窗口名的标识符。用在与窗口定义相关语法中（例如 WINDOW 子句和 RANK 等窗口函数）。请参见“标识符”一节第 8 页。

语法约定

在 SQL 语法说明中，使用了以下约定：

- **关键字** 所有 SQL 关键字都以大写字母显示，就像下面示例中的 SQL 语句 ALTER TABLE：

  ALTER TABLE [*owner.*]*table-name*

- **占位符** 必须替换为相应标识符或表达式的项，以斜体形式显示，就像下面示例中的 *owner* 和 *table-name* 这两个词：

  ALTER TABLE [*owner.*]*table-name*

- **可选部分** 语句的可选部分放在方括号内。例如：

  RELEASE SAVEPOINT [*savepoint-name*]

  这些方括号表示 *savepoint-name* 是可选项。不应键入方括号。

  有些关键字也可能括在方括号内。例如，以下语法表示可以使用 COMMIT TRAN 或 COMMIT TRANSACTION 这两者中的一个：

  COMMIT TRAN[SACTION] ...

  同样，以下语法表示可以使用 COMMIT 或 COMMIT WORK 中的一个：

  COMMIT [**WORK**]

- **重复项** 可以重复的项目后面接有适当的列表分隔符和一个省略号（三个英文句点），就像下面示例中的 *column-constraint*：

  ADD *column-definition* [*column-constraint*, ...]

  在本例中，可以不指定列约束，也可以指定一个或多个列约束。如果指定了多个列约束，则必须用逗号将它们隔开。

- **选项** 如果不必选择项目列表的项目或者只能选中一个，则用竖线分隔这些项目，并将列表放在方括号内。

  [ASC | DESC]

  例如，可以从 ASC、DESC 中任选一个，或者一个也不选。不应键入方括号。

- **二选一选项** 如果必须明确选择一个选项，则将替换选项放在大括号内。

  [QUOTES { ON | OFF }]

  在本例中，如果选择了 QUOTES 选项，则必须提供 ON 或 OFF 两者之一。不应键入方括号和大括号。

语句适用性指示符

某些语句标题的后面有一个指示符（括在方括号内），用于指示语句的使用环境。这些指示符如下：

- **[ESQL]** 表示语句用于嵌入式 SQL 中。
- **[Interactive SQL]** 表示语句只能用于 Interactive SQL 中。
- **[SP]** 表示语句用于存储过程、触发器或批处理语句中。
- **[T-SQL]** 表示实现语句的目的是为了与 Adaptive Server Enterprise 兼容。在某些情况下，语句不能用于非 Transact-SQL 格式的存储过程。在其它情况下，如果不存在 Transact-SQL 兼容问题，建议使用与 SQL/2003 标准接近的替换语句。
- **[external procedures]** 表示语句用于调用外部函数和过程。
- **[MobiLink]** 表示语句仅用于 MobiLink 客户端。
- **[SQL Remote]** 表示语句只能用于 SQL Remote。
- **[web services]** 表示语句用于 Web 服务客户端。

如果使用两组括号，表示语句可用于两种环境。例如，[ESQL][SP] 表示语句既可以用于嵌入式 SQL，又可以用于存储过程。

SQL 语句 (A-D)

以下各节定义了 SQL 语句 A-D 部分的语法信息。

另请参见

- “SQL 语句 (E-O)”一节第 565 页
- “SQL 语句 (P-Z)”一节第 655 页
- “SQL 语法中的常见元素”一节第 340 页
- “语法约定”一节第 341 页
- “语句适用性指示符”一节第 342 页

ALLOCATE DESCRIPTOR 语句 [ESQL]

此语句用于为 SQL 描述符区域 (SQLDA) 分配空间。

语法

ALLOCATE DESCRIPTOR *descriptor-name*
[**WITH MAX** { *integer* | *hostvar* }]

descriptor-name : *identifier*

参数

- **WITH MAX 子句** 可用于指定描述符区域内的变量数。缺省大小为 1。在执行读取操作或任何访问描述符区域内数据的语句之前，仍必须调用 fill_sqlda 为实际的数据项分配空间。

注释

为描述符区域 (SQLDA) 分配空间。使用此语句前，必须在 C 代码中声明以下内容：

```
struct sqlda * descriptor_name
```

权限

无。

副作用

无。

另请参见

- “DEALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 522 页
- “SQL 描述符区域 (SQLDA)”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 核心特性。

示例

下面的示例程序包括 ALLOCATE DESCRIPTOR 语句用法的一个示例。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;
#include "sqldef.h"
EXEC SQL BEGIN DECLARE SECTION;
int         x;
short       type;
int         numcols;
char        string[100];
a_SQL_statement_number  stmt = 0;
EXEC SQL END DECLARE SECTION;
int main(int argc, char * argv[]){
   struct sqlda *      sqlda1;
   if( !db_init( &sqlca ) ) {
      return 1;
   }
   db_string_connect( &sqlca,
   "UID=dba;PWD=sql;DBF=d:\\DB Files\\sample.db");
   EXEC SQL ALLOCATE DESCRIPTOR sqlda1 WITH MAX 25;
   EXEC SQL PREPARE :stmt FROM
      'SELECT * FROM Employees';
   EXEC SQL DECLARE curs CURSOR FOR :stmt;
   EXEC SQL OPEN curs;
   EXEC SQL DESCRIBE :stmt into sqlda1;
   EXEC SQL GET DESCRIPTOR sqlda1 :numcols=COUNT;
   // how many columns?
   if( numcols > 25 ) {
      // reallocate if necessary
      EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
      EXEC SQL ALLOCATE DESCRIPTOR sqlda1
         WITH MAX :numcols;
      EXEC SQL DESCRIBE :stmt into sqlda1;
   }
   type = DT_STRING; // change the type to string
   EXEC SQL SET DESCRIPTOR sqlda1 VALUE 2 TYPE = :type;
   fill_sqlda( sqlda1 );
   // allocate space for the variables
   EXEC SQL FETCH ABSOLUTE 1 curs
      USING DESCRIPTOR sqlda1;
   EXEC SQL GET DESCRIPTOR sqlda1
      VALUE 2 :string = DATA;
   printf("name = %s", string );
   EXEC SQL DEALLOCATE DESCRIPTOR sqlda1;
   EXEC SQL CLOSE curs;
   EXEC SQL DROP STATEMENT :stmt;
   db_string_disconnect( &sqlca, "" );
   db_fini( &sqlca );
   return 0;
}
```

ALTER DATABASE 语句

此语句用于升级数据库、为数据库打开或关闭 jConnect 支持、校准数据库、更改事务日志和事务日志镜像文件名或强制镜像服务器接管数据库。

如果对当前正在镜像的数据库服务器执行 ALTER DATABASE UPGRADE 语句，将会返回一条错误消息。

语法 1 - 升级组件或恢复对象

ALTER DATABASE UPGRADE
[PROCEDURE ON]
[JCONNECT { ON | OFF }]

语法 2 - 执行校准

ALTER DATABASE {
CALIBRATE [SERVER]
| **CALIBRATE DBSPACE** *dbspace-name*
| **CALIBRATE DBSPACE TEMPORARY**
| **CALIBRATE GROUP READ**
| **CALIBRATE PARALLEL READ**
| **RESTORE DEFAULT CALIBRATION**
}

语法 3 - 更改事务日志和事务日志镜像名称

ALTER DATABASE *dbfile*
ALTER [TRANSACTION] LOG {
{ **ON** [*log-name*] [**MIRROR** *mirror-name*] | **OFF** }
[**KEY** *key*]

语法 4 - 更改数据库所有权

ALTER DATABASE
{ *dbname* **FORCE START**
| **SET PARTNER FAILOVER** }

参数

- **PROCEDURE 子句** 在数据库中删除并重新创建所有 dbo 和 sys 拥有的过程。
- **JCONNECT 子句** 要允许 jConnect JDBC 驱动程序访问系统目录信息，请指定 JCONNECT ON。这将会安装提供 jConnect 支持的系统对象。如果要排除 jConnect 系统对象，请指定 JCONNECT OFF。只要不访问系统信息，您就仍然可以使用 JDBC。缺省情况下，JCONNECT 为 ON。

  如果要变更在 Windows Mobile 上使用的数据库，请参见“在 Windows Mobile 上使用 jConnect”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **CALIBRATE [SERVER] 子句** 校准除临时 dbspace 外的所有 dbspace。此子句还执行由 CALIBRATE PARALLEL READ 完成的工作。
- **CALIBRATE DBSPACE 子句** 校准指定的 dbspace。
- **CALIBRATE DBSPACE TEMPORARY 子句** 校准临时 dbspace。
- **CALIBRATE GROUP READ 子句** 在临时 dbspace 上执行组读取校准。向临时 dbspace 写入大型工作表，然后使用不同的组读取大小来为文件的读取计时。如果向临时表添加空格超出了连接的限制，或高速缓存大小不足以使用最大内存大小进行校准，则校准失败并返回错误消息。

- **CALIBRATE PARALLEL READ 子句** 为所有 dbspace 文件校准设备的并行 I/O 容量。CALIBRATE [SERVER] 子句也会执行此校准。
- **RESTORE DEFAULT CALIBRATION 子句** 将磁盘传送时间 (DTT) 模型恢复为基于典型硬件和配置设置的内置缺省值。
- **ALTER [TRANSACTION] LOG 子句** 更改事务日志或事务日志镜像文件的文件名。如果未指定 MIRROR *mirror-name*，此子句将为新事务日志设置一个文件名。如果数据库当前没有使用事务日志，它会开始使用一个。如果数据库已在使用事务日志，则它会改为将新文件用作其事务日志。

  如果已指定 MIRROR *mirror-name*，此子句将为新事务日志镜像设置一个文件名。如果数据库当前没有使用事务日志镜像，则它会开始使用一个。如果数据库已在使用事务日志镜像，则它会改为将新文件用作它的事务日志镜像。

  您也可以使用此子句关闭事务或事务日志镜像。例如，ALTER DATABASE LOG OFF。
- **KEY 子句** 指定用于事务日志或事务日志镜像的加密密钥。如果在高度加密的数据库上使用 ALTER [TRANSACTION] 子句，则必须指定加密密钥。
- **dbname FORCE START 子句** 强制当前作为镜像服务器的数据库服务器接管数据库。此子句可从过程或事件内部执行，并且必须在连接到镜像服务器上的实用程序数据库时执行。请参见“强制数据库服务器成为主服务器”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **SET PARTNER FAILOVER 子句** 启动从主服务器到镜像服务器的数据库镜像故障转移。此语句必须在连接到主服务器上的数据库时执行，并且可从过程或事件内部执行。执行此语句时，到数据库的全部现有连接都将关闭，包括执行此语句的连接。如果此语句包含在过程或事件中，它后面的其它语句将不会执行。请参见“启动主服务器上的故障转移”一节 《SQL Anywhere 服务器 - 数据库管理》。

注释

语法 1 可以使用 ALTER DATABASE UPGRADE 语句取代升级实用程序来升级或更新数据库。这也适用于维护版本。运行此语句后，您应该重新启动数据库。通常情况下，数据库的各个次版本之间的改动仅限于补充的数据库选项和小的系统表以及过程更改。ALTER DATABASE UPGRADE 语句会将系统表升级到当前版本并添加任何新数据库选项。如有必要，它还会删除并重新创建所有系统过程。通过指定 PROCEDURE ON 子句可以强制重建系统过程。

还可以使用 ALTER DATABASE UPGRADE 语句将设置和系统对象恢复到最初安装时的状态。

但通过执行 ALTER DATABASE UPGRADE 语句并不能提供要求对数据库文件进行物理重组的功能。这些功能包括索引增强和数据存储方面的改进。要获得这些增强的好处，您必须卸载并重装数据库。请参见“重建数据库”一节 《SQL Anywhere 服务器 - SQL 的用法》。

小心
在升级前，应始终对该数据库文件进行备份。如果在升级现有文件时升级失败，则这些文件将无法使用。有关备份数据库的信息，请参见“备份和数据恢复” 《SQL Anywhere 服务器 - 数据库管理》。

要使用 jConnect JDBC 驱动程序访问系统目录信息，请指定 JCONNECT ON（缺省值）。如果不打算包含 jConnect 系统对象，请指定 JCONNECT OFF。设置 JCONNECT OFF 并不会从数据库中删除 jConnect 支持。此外，只要不访问系统目录信息，仍可以使用 JDBC。如果随后下载了更新版本

的 jConnect，可通过执行（或重新执行） ALTER DATABASE UPGRADE JCONNECT ON 语句升级数据库中的旧版本。请参见“在数据库中安装 jConnect 系统对象”一节 《SQL Anywhere 服务器 - 编程》。

语法 2 语法 2 用于重新校准优化程序所使用的 I/O 开销模型。这将更新磁盘传送时间 (DTT) 模型，它是开销模型使用的磁盘 I/O 的数学模型。当重新校准 I/O 开销模型时，数据库服务器不能用作其它用途。另外，计算机上的所有其它活动必须是空闲的。重新校准数据库服务器是一项资源开销巨大的操作，可能需要一些才能时间完成。建议您保留缺省设置。

当使用 CALIBRATE PARALLEL READ 子句时，对于少于 10000 页的 dbspace 文件不执行并行校准。虽然数据库服务器在校准操作期间会自动挂起其所有活动，但并行校准应在同一计算机上没有进程消耗重要资源的情况下执行。校准之后，可使用扩展数据库属性 IOParallelism 检索 dbspace 文件上所允许的最大并行 I/O 操作估计次数。请参见“DB_EXTENDED_PROPERTY 函数 [System]”一节第 173 页。

当存在大量相似的硬件安装时，要避免重复且耗时的重复校准活动，可分别使用 sa_unload_cost_model 和 sa_load_cost_model 系统过程，通过先卸载校准然后再应用（装载）到另一数据库，以此来重新使用校准。请参见“sa_unload_cost_model 系统过程”一节第 910 页和“sa_load_cost_model 系统过程”一节第 845 页。

语法 3 可以使用 ALTER DATABASE 语句更改与数据库文件相关联的事务日志和事务日志镜像的名称。这些更改与事务日志 (dblog) 实用程序所做的更改相同。您可以在连接着该实用程序数据库或另一数据库（具体是哪个数据库取决于 -gu 选项的设置）时执行此语句。如果您要更改某个加密数据库的事务日志或事务日志镜像，则必须指定一个密钥。如果数据库正使用审记，则无法停止使用事务日志。关闭审计之后，可停止使用事务日志。在过程、触发器、事件或批处理中不支持此语法。

语法 4 ALTER DATABASE ...FORCE START 必须从镜像服务器（而非主服务器）运行。如果尝试对未镜像的数据库或当前处于活动状态并由此服务器所拥有的数据库执行 ALTER DATABASE ... FORCE START 语句，则会产生错误。另外，如果主服务器仍与镜像服务器相连接，则会给出一个错误。请参见“数据库镜像简介”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

对于语法 1 和 2，必须具有 DBA 权限，并且必须是唯一的数据库连接。Windows Mobile 上不支持 ALTER DATABASE UPGRADE。

对于语法 3，必须具有事务日志所在目录的文件权限，且数据库必须未运行。

对于语法 4，必须具有 -gk 服务器选项指定的权限。

副作用

自动提交

另请参见

- “CREATE DATABASE 语句”一节第 413 页
- “升级实用程序 (dbupgrad)”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CREATE STATISTICS 语句”一节第 491 页
- “事务日志实用程序 (dblog)”一节 《SQL Anywhere 服务器 - 数据库管理》
- “DB_EXTENDED_PROPERTY 函数 [System]”一节第 173 页
- “-gu 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_unload_cost_model 系统过程”一节第 910 页
- “sa_load_cost_model 系统过程”一节第 845 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例禁用 jConnect 支持：

```
ALTER DATABASE UPGRADE JCONNECT OFF;
```

以下示例将与 *demo.db* 关联的事务日志文件名设置为 *newdemo.log*：

```
ALTER DATABASE 'demo.db'
   ALTER LOG ON 'newdemo.log';
```

ALTER DBSPACE 语句

使用此语句可为 dbspace 或事务日志预分配空间，也可以在重命名或移动 dbspace 文件时更新目录。

语法

ALTER DBSPACE { *dbspace-name* | **TRANSLOG** | **TEMPORARY** }
{ **ADD** *number* [*add-unit*]
| **RENAME** *filename* }

add-unit :
PAGES
| **KB**
| **MB**
| **GB**
| **TB**

参数

- **TRANSLOG 子句** 提供一个专用的 dbspace 名称 TRANSLOG，以便为事务日志预分配磁盘空间。如果估计事务日志增长速度很快，则预分配可以提高性能。例如，当处理许多二进制大对象 (BLOB)（如位图）时，可能需要使用此功能。
- **TEMPORARY 子句** 提供专用 dbspace 名称 TEMPORARY，用于为临时 dbspace 增加空间。在为临时 dbspace 添加空间时，所增加的空间将立即体现在相应的临时文件中。如果为数据库的临时 dbspace 预分配空间，则在执行使用大型工作表的复杂查询过程中，性能将得到提高。

- **ADD 子句** 带 ADD 子句的 ALTER DBSPACE 可为 dbspace 预分配磁盘空间。它使相应的数据库文件按指定的大小扩展，可以指定的单位包括：页、千字节 (KB)、兆字节 (MB)、千兆字节 (GB) 或千吉字节 (TB)。如果未指定单位，则使用缺省的 PAGES。创建数据库后，数据库的页面大小是固定的。

  如果未预分配空间，则当需要空间时，对于 2 KB、4 KB 和 8 KB 的页面大小，数据库文件一次大约扩展 256 KB；对于其它页面大小将扩展大约 32 个页面。预分配空间可在装载大量数据时提高性能，还可以使数据库文件在文件系统中保持更好的连续性。

  可以使用此子句向任何预定义的 dbspace（Ssystem、temporary、temp、translog 和 translogmirror）添加空间。请参见“预定义 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **RENAME 子句** 如果重命名主文件以外的数据库文件或将其移到不同的目录或设备，可以使用带 RENAME 子句的 ALTER DBSPACE，以确保数据库启动时 SQL Anywhere 能够找到新文件。*filename* 参数可以是字符串文字或变量。

  名称更改生效，如下所示：

  - 如果 dbspace 在语句执行前（即，尚未重命名实际文件）已打开，它仍是可访问的；但是存储在目录中的名称已更新。数据库停止后，必须使用 RENAME 子句重命名文件，使之与所提供的文件名匹配，否则文件名将不会与目录中的 dbspace 名匹配，而且数据库服务器无法在下次启动数据库时打开 dbspace。
  - 如果在执行语句时未打开 dbspace，数据库服务器将在更新目录后尝试打开它。如果可以打开 dbspace，就可以访问它。如果无法打开 dbspace，则不返回任何错误。

    要确定 dbspace 是否打开，请执行以下语句。如果结果为 NULL，则 dbspace 未打开。

    ```
    SELECT DB_EXTENDED_PROPERTY('FileSize','dbspace-name');
    ```

    对主 dbspace（即 system）使用带 RENAME 子句的 ALTER DBSPACE 无效。

注释

每个数据库都保存在一个或多个文件中。dbspace 是一个附加文件，它具有与每个数据库文件关联的逻辑名称，用于保存比主数据库文件自己所能容纳的数据更多的数据。ALTER DBSPACE 修改主 dbspace（也称为根文件）或附加 dbspace。数据库的 dbspace 名保存在 ISYSFILE 系统表中。主数据库文件的 dbspace 名为 system。

当启动多文件数据库时，启动行或 ODBC 数据源说明会向 SQL Anywhere 告知主数据库文件的位置。主数据库文件中保存着系统表。SQL Anywhere 在这些系统表中查找其它 dbspace 的位置，然后逐个打开它们。可通过设置 default_dbspace 选项指定在哪个 dbspace 中创建新表。

权限

必须具有 DBA 权限。必须是唯一的数据库连接。

副作用

自动提交。

另请参见

- “CREATE DBSPACE 语句”一节第 420 页
- “default_dbspace 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “使用数据库文件” 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例将 system dbspace 的大小增加 200 页：

```
ALTER DBSPACE system
ADD 200;
```

以下示例将 system dbspace 的大小增加了 400 MB：

```
ALTER DBSPACE system
ADD 400 MB;
```

以下示例更改与 system_2 dbspace 关联的文件名：

```
ALTER DBSPACE system_2
RENAME 'e:\db\dbspace2.db';
```

ALTER DOMAIN 语句

使用此语句可重命名用户定义的域或数据类型。

语法

ALTER { **DOMAIN** | **DATATYPE** } *user-type*
RENAME *new-name*

注释

执行此语句时，用户定义的域或数据类型的名称在 ISYSUSERTYPE 系统表中更新。

> **注意**
> 引用用户定义的域或数据类型的任何过程、触发器、视图或事件都必须重新创建，否则它们将继续引用旧名称。

权限

必须具有 DBA 权限或者是创建域的数据库用户。

副作用

自动提交。

另请参见

- “ISYSFILE 系统表”一节第 759 页
- “CREATE DOMAIN 语句”一节第 424 页
- “域”一节第 104 页
- “使用域”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例将 Address 域重命名为 MailingAddress：

```
ALTER DOMAIN Address RENAME MailingAddress;
```

ALTER EVENT 语句

此语句用于更改事件的定义或者与之关联的、自动执行预定义操作的处理程序，或用于更改调度操作的定义。也可以使用此语句隐藏事件处理程序的定义。

语法 1 - 变更事件

ALTER EVENT [*owner*.]*event-name*
[**AT** { **CONSOLIDATED** | **REMOTE** | **ALL** }]
[{ **DELETE TYPE**
  | **TYPE** *event-type*
  | **WHERE** { *trigger-condition* | **NULL** }
  | { **ADD** | **ALTER** | **DELETE** } **SCHEDULE** *schedule-spec* }]
[**ENABLE** | **DISABLE**]
[[**ALTER**] **HANDLER** *compound-statement* | **DELETE HANDLER**]

event-type :
BackupEnd
| **Connect**
| **ConnectFailed**
| **DatabaseStart**
| **DBDiskSpace**
| **Deadlock**
| **"Disconnect"**
| **GlobalAutoincrement**
| **GrowDB**
| **GrowLog**
| **GrowTemp**
| **LogDiskSpace**
| **RAISERROR**
| **ServerIdle**
| **TempDiskSpace**

trigger-condition :
event_condition(*condition-name* **)** { **=** | **<** | **>** | **!=** | **<=** | **>=** } *value*

schedule-spec :
[*schedule-name*]

{ **START TIME** *start-time* | **BETWEEN** *start-time* **AND** *end-time* }
[**EVERY** *period* { **HOURS** | **MINUTES** | **SECONDS** }]
[**ON** { **(** *day-of-week*, ... **)** | **(** *day-of-month*, ... **)** }]
[**START DATE** *start-date*]

event-name | *schedule-name* : *identifier*

day-of-week : *string*

value | *period* | *day-of-month* : *integer*

start-time | *end-time* : *time*

start-date : *date*

语法 2 - 隐藏事件处理程序的定义

ALTER EVENT *event-name* **SET HIDDEN**

参数

- **AT 子句** 此子句用于更改处理事件所用数据库的相关规范。
- **DELETE TYPE 子句** 此子句用于删除事件与事件类型的关联。有关事件类型的说明，请参见“了解系统事件”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **ADD | ALTER | DELETE SCHEDULE 子句** 此子句用于更改调度的定义。在任一 ALTER EVENT 语句中只能更改一个调度。
- **WHERE 子句** 此子句用于更改触发事件的触发条件。WHERE NULL 选项删除条件。有关大多数参数的说明，请参见“CREATE EVENT 语句”一节第 429 页。
- **START TIME 子句** 此子句用于指定事件的开始时间，还可用于指定事件的结束时间（可选）。*start-time* 和 *end-time* 参数为字符串（例如 `'12:34:56'`）。不允许使用变量和表达式（例如 `NOW()`）。
- **START DATE 子句** 此子句用于指定事件的开始日期。*start-date* 参数是一个字符串。不允许使用变量和表达式（例如 `TODAY()`）。
- **SET HIDDEN 子句** 此子句用于隐藏事件处理程序的定义。指定 SET HIDDEN 子句的结果是对存储在 ISYSEVENT 系统表操作列中的事件处理程序定义进行永久性模糊处理。

注释

此语句可用于更改通过 CREATE EVENT 创建的事件定义。可能的用法包括：

- 您可能需要隐藏事件处理程序的定义。
- 在开发阶段，可能需要定义和测试没有触发器条件或调度的事件处理程序，然后在事件处理程序完成后，使用 ALTER EVENT 添加执行条件。

如果需要更改某个事件，可在该事件运行时通过执行 ALTER EVENT ...DISABLE 语句禁用这些视图。要在 Sybase Central 中禁用某个事件，请右击该事件，然后清除 [已启用] 选项。禁用该事件并不会中断当前事件处理程序的执行；事件处理程序会继续执行，直到完成。事件处理程序完成后，

它不会重新启动，直到您重新启用它。您可以更改定义，然后重新启用。要确定哪些事件正在运行，请执行以下语句：

```
SELECT *
FROM dbo.sa_conn_info()
WHERE CONNECTION_PROPERTY('EventName',Number ) = 'event-name'
```

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “了解系统事件”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SYSEVENT 系统视图”一节第 946 页
- “BEGIN 语句”一节第 395 页
- “CREATE EVENT 语句”一节第 429 页

标准和兼容性

- **SQL/2003** 服务商扩充。

ALTER EXTERNAL ENVIRONMENT 语句

此语句用于指定外部环境（如 JAVA、PHP 或 Perl）的位置，或用于指定外部环境连接数据库时所使用的具有 DBA 权限的用户。

语法

ALTER EXTERNAL ENVIRONMENT *environment-name*
  [**USER** *user-name*]
  [**LOCATION** *location-string*]

environment-name :
JAVA
| **PERL**
| **PHP**
| **CLR**
| **C_ESQL32**
| **C_ESQL64**
| **C_ODBC32**
| **C_ODBC64**

参数

- **environment-name** *environment-name* 用于指定您所变更的外部环境。
- **USER 子句** USER 子句用于指定数据库中具有 DBA 权限的用户的名称。在最初启动外部环境时，必须将其连接到数据库。缺省情况下，此连接使用 DBA 用户 ID 建立，但如果数据库管理员倾向于让外部环境使用另一个具有 DBA 权限的用户 ID，则使用 *user-name* 来指示用户。

- **LOCATION 子句** LOCATION 子句用于指定数据库服务器计算机上可找到外部环境可执行文件/二进制文件的位置。它包括可执行文件名/二进制文件名。此路径可以是完全限定路径，也可以是相对路径。如果是相对路径，则可执行文件/二进制文件必须位于服务器可以找到的位置。

注释

有关如何使用外部环境的详细信息，请参见“外部环境概述”一节 《SQL Anywhere 服务器 - 编程》。

权限

必须具有 DBA 权限。

副作用

无

另请参见

- “外部环境概述”一节 《SQL Anywhere 服务器 - 编程》
- “START EXTERNAL ENVIRONMENT 语句”一节第 713 页
- “STOP EXTERNAL ENVIRONMENT 语句”一节第 720 页
- “INSTALL EXTERNAL OBJECT 语句”一节第 620 页
- “REMOVE EXTERNAL OBJECT 语句”一节第 672 页
- “SYSEXTERNENV 系统视图”一节第 947 页
- “SYSEXTERNENVOBJECT 系统视图”一节第 948 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例标识了一个具有 DBA 权限的用户，外部环境使用该用户连接到数据库服务器和 Perl 可执行文件的位置。

```
ALTER EXTERNAL ENVIRONMENT PERL
USER DBADMIN
LOCATION 'c:\\Perl64\\bin\\perl.exe';
```

ALTER FUNCTION 语句

此语句用于修改函数。必须在 ALTER FUNCTION 语句中包括整个新函数。

语法 1

ALTER FUNCTION [*owner.*]*function-name function-definition*

function-definition : CREATE FUNCTION syntax

语法 2

ALTER FUNCTION [*owner.*]*function-name*
SET HIDDEN

语法 3

ALTER FUNCTION [*owner*.]*function-name*
RECOMPILE

注释

语法 1 除第一个单词不同外，ALTER FUNCTION 语句的语法与 CREATE FUNCTION 语句的语法相同。两种语法形式的 CREATE FUNCTION 语句都可以这样更改。

系统保留函数的现有权限，因此不必重新分配。如果执行了 DROP FUNCTION 和 CREATE FUNCTION，则必须重新分配执行权限。

语法 2 使用 SET HIDDEN 对关联函数的定义进行模糊处理，使之不可读。可以卸载此函数，然后将其重装到其它数据库中。

此设置是不可逆的。如果再次需要原始源，则必须将它保存在数据库之外。

如果使用 SET HIDDEN，则使用调试程序进行调试将不会显示函数定义，也无法通过过程分析获得函数定义。

语法 3 使用 RECOMPILE 语法重新编译用户定义的 SQL 函数。当重新编译函数时，存储在目录中的定义被重新分析，其语法也被验证。重新编译没有更改函数的保留源。重新编译函数时，由 SET HIDDEN 子句模糊处理过的定义仍是模糊形式，仍不可读取。

权限

必须是函数的所有者或者具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE FUNCTION 语句（Web 服务）”一节第 445 页
- “ALTER PROCEDURE 语句”一节第 361 页
- “DROP FUNCTION 语句”一节第 546 页
- “隐藏过程、函数、触发器和视图的内容”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在此示例中，创建并更改了 MyFunction。SET HIDDEN 子句对函数定义进行模糊处理，使其不可读。

```
CREATE FUNCTION MyFunction(
   firstname CHAR(30),
   lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
   DECLARE name CHAR(61);
   SET name = firstname || ' ' || lastname;
```

```
   RETURN (name);
ALTER FUNCTION MyFunction SET HIDDEN;
END;
```

ALTER INDEX 语句

此语句用于重命名索引、主键或外键，或用于更改索引的聚簇性质。

语法

ALTER { **INDEX** *index-name*
| [**INDEX**] **FOREIGN KEY** *role-name*
| [**INDEX**] **PRIMARY KEY** }
ON [*owner*.]*object-name* { **REBUILD** | *rename-clause* | *cluster-clause* } }

object-name : *table-name* | *materialized-view-name*

rename-clause : **RENAME** { **AS** | **TO** } *new-index-name*

cluster-clause : **CLUSTERED** | **NONCLUSTERED**

参数

- **rename-clause** 为索引、主键或外键指定新名称。
- **cluster-clause** 指定索引应更改为 CLUSTERED 还是 NONCLUSTERED。一个表上只有一个索引可以是聚簇索引。
- **REBUILD 子句** 此子句用于重建索引，而不是删除然后重新创建索引。

注释

ALTER INDEX 语句执行两项任务：

- 此语句可用于重命名索引、主键或外键。
- 还可用于将索引类型从非聚簇更改为聚簇，反之亦然。

  ALTER INDEX 语句可用于更改为索引指定的聚簇类型，但不会重新组织数据。同样，每个表或实例化视图上只有一个索引可以是聚簇索引。

ALTER INDEX 不能用于更改局部临时表上的索引。如果试图执行此操作，将导致出现 [未找到索引] 错误。

当存在使用 WITH HOLD 子句打开的使用语句快照或事务快照的游标时，不能执行此语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

必须是表的所有者，或者在表或实例化视图上具有 REFERENCES 权限，或者具有 DBA 权限。

副作用

自动提交。清除 Interactive SQL 中 [结果] 窗格上的 [结果] 选项卡中的内容。关闭用于当前连接的所有游标。

另请参见

- “CREATE INDEX 语句”一节第 449 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句将 Products 表上的索引 IX_product_name 重命名为 ixProductName：

```
ALTER INDEX IX_product_name ON Products
RENAME TO ixProductName;
```

以下语句将 IX_product_name 更改为聚簇索引：

```
ALTER INDEX IX_product_name ON Products
CLUSTERED;
```

ALTER LOGIN POLICY 语句

使用此语句变更现有的登录策略。

语法

ALTER LOGIN POLICY *policy-name policy-options*

policy options :
policy-option [*policy-option ...*]

policy-option :
policy-option-name **=** *policy-option-value*

policy-option-value :
{ **UNLIMITED**
| **root**
| *legal-option-value* }

参数

- **policy-name** 登录策略的名称。指定修改根登录策略的根。
- **policy-option-name** 策略选项的名称。要查看缺省的登录策略选项名称列表及说明，请参见“CREATE LOGIN POLICY 语句”一节第 453 页的注释部分。
- **policy-option-value** 指派给登录策略选项的值。如果指定为 UNLIMITED，则未使用限制。如果指定为缺省值，则使用缺省的限制。要查看缺省的登录策略选项值列表，请参见“CREATE LOGIN POLICY 语句”一节第 453 页的注释部分。

注释

变更登录策略后，更改会立即应用到所有用户。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “修改登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》
- “ALTER USER 语句”一节第 385 页
- “COMMENT 语句”一节第 406 页
- “CREATE LOGIN POLICY 语句”一节第 453 页
- “CREATE USER 语句”一节第 518 页
- “DROP LOGIN POLICY 语句”一节第 548 页
- “DROP USER 语句”一节第 562 页
- “管理登录策略概述”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例变更登录策略 Test1。此示例更改锁定 locked 和 max_connections 选项。锁定的值表示禁止使用该策略的用户建立新的连接，max_connections 值表示允许的并发连接数。

```
ALTER LOGIN POLICY Test1
locked=ON
max_connections=5;
```

ALTER MATERIALIZED VIEW 语句

此语句用于变更实例化视图。

语法

ALTER MATERIALIZED VIEW [*owner.*]*materialized-view-name* {
SET HIDDEN
| { **ENABLE** | **DISABLE** }
| { **ENABLE** | **DISABLE** } **USE IN OPTIMIZATION**
| { **ADD PCTFREE** *percent-free-space* | **DROP PCTFREE** }
| [**NOT**] **ENCRYPTED**
[{ **IMMEDIATE** | **MANUAL** } **REFRESH**]
}

percent-free-space : integer

参数

- **SET HIDDEN 子句** SET HIDDEN 子句用于对实例化视图的定义进行模糊处理。*此设置是不可逆的*。有关详细信息，请参见“隐藏实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **ENABLE 子句** ENABLE 子句用于启用已被禁用的实例化视图，使之可供数据库服务器使用。此子句对于已启用的视图无效。使用此子句后，必须刷新以便对视图进行初始化，然后重新创建视图禁用时被删除的所有文本索引。
- **DISABLE 子句** DISABLE 子句用于禁止数据库服务器使用视图。禁用实例化视图时，数据库服务器会删除视图的数据及索引。
- **{ ENABLE | DISABLE } USE IN OPTIMIZATION 子句** 此子句用于指定是否希望实例化视图可供优化程序使用。如果指定 DISABLE USE IN OPTIMIZATION，则只能在执行显式引用视图的查询时使用实例化视图。缺省使用 ENABLE USE IN OPTIMIZATION。请参见“允许和禁止优化程序使用实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **ADD PCTFREE 子句** 指定希望在每页上保留的可用空间的百分比。如果数据更新时行大小增加，将占用可用空间。如果某页上没有可用空间，则每次增加该页上的行大小时，都需要将该行拆分到多个页中，从而导致产生行碎片并可能引起性能下降。

  percent-free-space 的值是一个介于 0 和 100 之间的整数。值为 0 表示每个页面上均没有可用空间—每个页面均完全填满。如果值很高，会使每行单独插入到页中。如果未设置 PCTFREE，或将其删除，则会根据数据库页面大小应用缺省 PCTFREE 设置（4 KB 页面大小应用 200 字节，2 KB 页面大小应用 100 字节）。
- **DROP PCTFREE 子句** 删除实例化视图的当前有效 PCTFREE 设置，并根据数据库页大小应用缺省 PCTFREE 值。
- **[NOT] ENCRYPTED 子句** 指定是否加密实例化视图数据。缺省情况下，创建时不加密实例化视图的数据。要加密实例化视图，请指定 ENCRYPTED。要解密实例化视图，请指定 NOT ENCRYPTED。
- **REFRESH 子句** 使用 REFRESH 子句更改实例化视图的刷新类型：
  - **IMMEDIATE REFRESH** 使用 IMMEDIATE REFRESH 子句将手动视图更改为快速视图。手动视图必须是有效的且未被初始化的，这样才能将刷新类型更改为 IMMEDIATE REFRESH。如果视图处于初始化状态，在执行 ALTER MATERIALIZED VIEW...IMMEDIATE REFRESH 前请执行 TRUNCATE 语句将状态更改为未初始化的状态。请参见“TRUNCATE 语句”一节第 727 页。

    有关在可将视图变更为 IMMEDIATE REFRESH 之前必须满足的条件的信息，请参见“快速视图的附加限制”一节 《SQL Anywhere 服务器 - SQL 的用法》。
  - **MANUAL REFRESH** MANUAL REFRESH 子句用于将快速视图更改为手动视图。

  有关刷新类型的详细信息，请参见“手动和快速实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  有关状态的详细信息，请参见“实例化视图状态和属性”一节 《SQL Anywhere 服务器 - SQL 的用法》。

注释

如果变更另一个用户拥有的实例化视图，必须通过包含该所有者来限定名称（例如，GROUPO.EmployeeConfidential）。如果不限定名称，数据库服务器就会查找具有您所拥有的那个名称的实例化视图，并对其进行变更。如果没有这样的视图，服务器将返回错误。

当禁用实例化视图（使用 DISABLE 子句）后，数据库服务器便无法再使用此视图来回应查询。此外，数据和索引被删除，刷新类型更改为手动。还会禁用所有相关的常规视图。

DISABLE 子句不仅要求对被禁用的视图具有独占访问权限，还要求对所有相关视图具有独占访问权限，因为这些视图也会被禁用。请参见“启用和禁用实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

若要加密实例化视图（使用 ENCRYPT 子句），必须在数据库中已启用表加密。然后系统会使用数据库创建时所指定的加密密钥和算法来加密实例化视图。请参见“加密和解密实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

要执行 ALTER MATERIALIZED VIEW 语句，您必须拥有视图或具有 DBA 权限。

如果您不具有 DBA 权限但想将实例化视图变更为快速视图（使用 ALTER MATERIALIZED VIEW ...IMMEDIATE REFRESH 语句），那么您必须拥有视图以及它所引用的所有表。

用户对实例化视图可执行的更改数据操作仅有刷新、截断和禁用。但是，快速视图可由数据库服务器自动更新。即启用快速视图并进行初始化后，数据库服务器会就自动维护它，不需要附加的权限检查。

副作用

自动提交。

另请参见

- “CREATE MATERIALIZED VIEW 语句”一节第 455 页
- “REFRESH MATERIALIZED VIEW 语句”一节第 665 页
- “sa_refresh_materialized_views 系统过程”一节第 878 页
- “TRUNCATE 语句”一节第 727 页
- “DROP MATERIALIZED VIEW 语句”一节第 549 页
- “使用实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “视图依赖性”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句创建 EmployeeConfid88 实例化视图，然后禁止它在优化中的使用：

```
CREATE MATERIALIZED VIEW EmployeeConfid88 AS
   SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary,
ManagerID,
      Departments.DepartmentName, Departments.DepartmentHeadID
   FROM Employees, Departments
   WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid88;
ALTER MATERIALIZED VIEW GROUPO.EmployeeConfid88 DISABLE USE IN OPTIMIZATION;
```


小心
处理该示例时，应删除所创建的实例化视图。否则，在试验其它示例时，将无法对其基础表 Employees 和 Departments 执行模式更改。无法变更具有已启用相关实例化视图的表的模式。请参见“删除实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

ALTER PROCEDURE 语句

此语句用于修改过程，或启用和禁用使用 Sybase 复制服务器对该过程进行复制。必须在 ALTER PROCEDURE 语句中包括整个新过程。

可使用 PROC 作为 PROCEDURE 的同义词。

语法 1

ALTER PROCEDURE [*owner*.]*procedure-name procedure-definition*

procedure-definition : CREATE PROCEDURE syntax

语法 2

ALTER PROCEDURE [*owner*.]*procedure-name*
REPLICATE { **ON** | **OFF** }

语法 3

ALTER PROCEDURE [*owner*.]*procedure-name*
SET HIDDEN

语法 4

ALTER PROCEDURE [*owner*.]*procedure-name*
RECOMPILE

注释

语法 1 除第一个单词不同外，ALTER PROCEDURE 语句的语法与 CREATE PROCEDURE 语句的语法相同。两种语法形式的 CREATE PROCEDURE 语句都可以这样更改。

系统会保留对过程的现有权限，因此不必重新指派。如果执行了 DROP PROCEDURE 和 CREATE PROCEDURE，则必须重新分配执行权限。

语法 2 如果要使用 Sybase 复制服务器将一个过程复制到其它站点，则必须对该过程设置 REPLICATE ON。

语法 3 使用 SET HIDDEN 对关联过程的定义进行模糊处理，使之不可读。可以卸载该过程，然后将其重装到其它数据库中。

*此设置是不可逆的。*如果再次需要原始源，则必须将它保存在数据库之外。

如果使用 SET HIDDEN，则使用调试程序进行调试将不会显示过程定义，也无法通过过程分析获得过程定义。

语法 2 不可与语法 1 合并使用。语法 3 不可与语法 1 合并使用，也不可与语法 2 合并使用。

语法 4 使用 RECOMPILE 语法重新编译存储的过程。当重新编译一个过程时，存储在目录中的定义被重新分析，其语法也被验证。对于生成结果集但不包含 RESULT 子句的过程，数据库服务器会尝试确定过程结果集的特点，并将信息存储在目录中。如果自过程创建以来，过程所引用的表发生变更，从而添加、删除或重命名了列，这些信息会很有用。

重新编译不会更改过程的定义。可以重新编译使用 SET HIDDEN 子句隐藏其定义的过程，但其定义仍是隐藏的。

权限

必须是过程的所有者或者具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE PROCEDURE 语句（Web 服务）”一节第 471 页
- “ALTER FUNCTION 语句”一节第 354 页
- “DROP PROCEDURE 语句”一节第 550 页
- “隐藏过程、函数、触发器和视图的内容”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

ALTER PUBLICATION 语句 [MobiLink] [SQL Remote]

此语句用于变更发布。在 MobiLink 中，发布可以标识 SQL Anywhere 远程数据库中的同步数据。在 SQL Remote 中，发布可以标识统一数据库和远程数据库中的复制数据。

语法

ALTER PUBLICATION [*owner*.]*publication-name alterpub-clause*, ...

alterpub-clause:
  ADD *article-definition*
| **ALTER** *article-definition*
| { **DELETE** | **DROP** } **TABLE** [*owner*.]*table-name*
| **RENAME** *publication-name*

article-definition :
TABLE *table-name* [**(** *column-name*, ... **)**]
[**WHERE** *search-condition*]
[**SUBSCRIBE BY** *expression*]
[**USING (** [**PROCEDURE**] [*owner*.][*procedure-name*]
  FOR UPLOAD { **INSERT** | **DELETE** | **UPDATE** }, ... **)**]

注释

此语句仅适用于 MobiLink 和 SQL Remote。

ALTER PUBLICATION 语句会变更数据库中的发布。发布中来自一个表的部分称为一个项目。通过添加、修改或删除项目，或者通过重命名发布，可以对发布进行更改。如果修改某个项目，必须输入已修改项目的完整定义。

建议在成功执行同步发布以后立即对其进行变更。

您不能对定义为 FOR DOWNLOAD ONLY 或 WITH SCRIPTED UPLOAD 的发布使用 WHERE 子句。

SUBSCRIBE BY 子句仅适用于 SQL Remote。

USING 子句仅用于脚本式上载。

应使用 ALTER SYNCHRONIZATION SUBSCRIPTION 语句或 CREATE SYNCHRONIZATION SUBSCRIPTION 语句中的 ADD OPTION 子句为 MobiLink 发布设置选项。

权限

必须具有 DBA 权限，或者是发布的所有者。要求可以对语句中引用的所有表进行独占访问。

副作用

自动提交。

另请参见

- “CREATE PUBLICATION 语句 [MobiLink] [SQL Remote]”一节第 476 页
- “DROP PUBLICATION 语句 [MobiLink] [SQL Remote]”一节第 551 页
- SQL Anywhere MobiLink 客户端： “发布数据”一节 《MobiLink - 客户端管理》
- UltraLite MobiLink 客户端： “设计 UltraLite 中的同步”一节 《UltraLite - 数据库管理和参考》
- “ALTER SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节第 370 页
- “CREATE SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节第 494 页
- “ISYSSYNC 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句将 Customers 表添加到 pub_contact 发布中。

```
ALTER PUBLICATION pub_contact
   ADD TABLE Customers;
```

ALTER REMOTE MESSAGE TYPE 语句 [SQL Remote]

对于一个已创建的消息类型，此语句用于更改发布者的消息系统，或者更改给定消息系统的发布者地址。

语法

ALTER REMOTE MESSAGE TYPE *message-system*
ADDRESS *address*

message-system: **FILE** | **FTP** | **SMTP**

address: *string*

参数

- **message-system** SQL Remote 支持的消息系统之一。它必须是以下值之一：
  - **address** 一个字符串，包含指定消息系统的有效地址。

注释

此语句更改给定消息类型的发布者地址。

消息代理通过所支持的消息链接之一，从数据库发送外发消息。抽取实用程序在远程数据库中执行 GRANT CONSOLIDATE 语句时使用此地址。

地址是指定的消息系统中发布者的地址。如果是电子邮件系统，则地址字符串必须是有效的电子邮件地址。如果是文件共享系统，则地址字符串是 SQLREMOTE 环境变量指定的目录的子目录，如果未设置该变量，则是当前目录的子目录。在远程数据库中，可以对 GRANT CONSOLIDATE 语句替换此设置。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE REMOTE MESSAGE TYPE 语句 [SQL Remote]”一节第 479 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句将 FILE 消息链接的发布者地址更改为 new_addr。

```
ALTER REMOTE MESSAGE TYPE file
ADDRESS 'new_addr';
```

ALTER SERVER 语句

此语句用于修改远程服务器的属性。

语法

ALTER SERVER *server-name*
[**CLASS** *server-class*]
[**USING** *connection-info*]
[**CAPABILITY** *cap-name* { **ON** | **OFF** }]
[**CONNECTION CLOSE** [**CURRENT** | **ALL** | *connection-id*]]

server-class :
SAODBC
| **ASEODBC**
| **DB2ODBC**
| **MSSODBC**
| **ORAODBC**
| **MSACCESSODBC**
| **MYSQLODBC**
| **ULODBC**
| **ADSODBC**
| **ODBC**
| **SAJDBC**
| **ASEJDBC**

connection-info :
computer-name:*port-number*[/*dbname*] | *data-source-name*

参数

- **CLASS 子句** 指定 CLASS 子句以更改服务器类。

  有关服务器的类及如何配置服务器的详细信息，请参见“用于远程数据访问的服务器类”《SQL Anywhere 服务器 - SQL 的用法》。

- **USING 子句** 指定 USING 子句可更改服务器连接信息。有关 *connection-info* 的信息，请参见“CREATE SERVER 语句”一节第 481 页。

- **CAPABILITY 子句** CAPABILITY 子句可将服务器功能设置为 ON 或 OFF。服务器功能存储在系统表 ISYSCAPABILITY 中。通过 SYSCAPABILITYNAME 系统视图可访问这些功能的名称。在与远程服务器建立了第一个连接后，系统表 ISYSCAPABILITY 和系统视图 SYSCAPABILITYNAME 中才会填充数据。对于后面的连接，可从系统表 ISYSCAPABILITY 中获取数据库服务器的功能。

  通常情况下，不需要变更服务器的功能。但可能需要变更属于 ODBC 类的通用服务器的功能。

- **CONNECTION CLOSE 子句** 当用户创建与远程服务器的连接时，此远程连接直到用户与本地数据库断开连接后才关闭。CONNECTION CLOSE 子句允许您显式关闭到远程服务器的连接。当远程连接处于非活动状态或不再需要时，您会发现此功能很有用。

  以下 SQL 语句是等效的，均可关闭到远程服务器的当前连接：

  ```
  ALTER SERVER server-name CONNECTION CLOSE;

  ALTER SERVER server-name CONNECTION CLOSE CURRENT;
  ```

  可以使用此语法关闭与远程服务器的 ODBC 连接和 JDBC 连接。执行这些语句不需要 DBA 权限。

  还可以通过指定连接 ID 来断开特定远程 ODBC 连接，或通过指定 ALL 关键字来断开所有远程 ODBC 连接。如果试图通过指定连接 ID 或 ALL 关键字来关闭 JDBC 连接，则会发生错误。如果由 *connection-id* 标识的连接不是当前本地连接，则用户必须具有 DBA 权限才能够关闭此连接。

注释

ALTER SERVER 语句修改服务器的属性。这些更改直到下次连接远程服务器之后才生效。

权限

必须具有 RESOURCE 权限。

副作用

自动提交。

另请参见

- “SYSCAPABILITY 系统视图”一节第 939 页
- “SYSCAPABILITYNAME 系统视图”一节第 940 页
- “CREATE SERVER 语句”一节第 481 页
- “DROP SERVER 语句”一节第 552 页
- “用于远程数据访问的服务器类” 《SQL Anywhere 服务器 - SQL 的用法》
- “远程数据访问疑难解答”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例更改名为 ase_prod 的 Adaptive Server Entreprise 服务器的服务器类，以使其与 SQL Anywhere 的连接基于 ODBC。其数据源名为 ase_prod。

```
ALTER SERVER ase_prod
CLASS 'ASEODBC'
USING 'ase_prod';
```

以下示例更改服务器 infodc 的功能。

```
ALTER SERVER infodc
CAPABILITY 'insert select' OFF;
```

以下示例关闭了与名为 rem_test 的远程服务器的所有连接。

```
ALTER SERVER rem_test
CONNECTION CLOSE ALL;
```

以下示例关闭与连接 ID 为 142536 名为 rem_test 的远程服务器的连接。

```
ALTER SERVER rem_test
CONNECTION CLOSE 142536;
```

ALTER SERVICE 语句

此语句可用于变更 Web 服务。

语法 1 - DISH 服务

ALTER SERVICE *service-name*
[**TYPE 'DISH'**]
[**GROUP** { *group-name* | **NULL** }]
[**FORMAT** { **'DNET'** | **'CONCRETE'** [**EXPLICIT** { **ON** | **OFF** }]| **'XML'** | **NULL** }]
[*common-attributes*]

语法 2 - SOAP 服务

ALTER SERVICE *service-name*
[**TYPE 'SOAP'**]
[**DATATYPE** { **ON** | **OFF** | **IN** | **OUT** }]
[**FORMAT** { **'DNET'** | **'CONCRETE'** [**EXPLICIT** { **ON** | **OFF** }] | **'XML'** | **NULL** }]
[*common-attributes*]
[**AS** *statement*]

语法 3 - 杂项服务

ALTER SERVICE *service-name*
[**TYPE** { **'RAW'** | **'HTML'** | **'XML'** }]
[**URL** [**PATH**] { **ON** | **OFF** | **ELEMENTS** }]
[*common-attributes*]
[**AS** { *statement* | **NULL** }]

common-attributes:
[**AUTHORIZATION** { **ON** | **OFF** }]
[**ENABLE** | **DISABLE**]
[**METHODS** '*method,...*']
[**SECURE** { **ON** | **OFF** }]
[**USER** { *user-name* | **NULL** }]

method:
DEFAULT
| **POST**
| **GET**
| **HEAD**
| **PUT**
| **DELETE**
| **NONE**
| *****

参数

对 ALTER SERVICE 参数的说明与对 CREATE SERVICE 语句的说明相同。请参见“CREATE SERVICE 语句”一节第 484 页。

注释

ALTER SERVICE 语句会更改系统表 ISYSWEBSERVICE，并允许数据库服务器充当 Web 服务器。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “使用 SOAP 服务”一节 《SQL Anywhere 服务器 - 编程》
- “CREATE SERVICE 语句”一节第 484 页
- “DROP SERVICE 语句”一节第 553 页
- “SYSWEBSERVICE 系统视图”一节第 987 页
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

若要快速设置 Web 服务器，请在启动数据库服务器时使用 -xs（http 或 https）选项，然后执行以下语句：

```
CREATE SERVICE tables TYPE 'HTML';

ALTER SERVICE tables
   AUTHORIZATION OFF
   USER DBA
   AS SELECT *
      FROM SYS.SYSTAB;
```

执行完这些语句之后，可使用任何 Web 浏览器打开 URL *http://localhost/tables*。

ALTER STATISTICS 语句

此语句用于控制是否自动更新表中的一列或多列统计信息。

语法

ALTER STATISTICS
[**ON**] *table* [**(** *column1* [**,** *column2* ...] **)**]
AUTO UPDATE { **ENABLE** | **DISABLE** }

参数

- **ON** ON 是可选项。包括 ON 对语句的执行没有影响。
- **AUTO UPDATE 子句** 指定启用还是禁用列统计信息的自动更新。

注释

在查询、DML 语句 和 LOAD TABLE 语句的正常执行过程中，数据库服务器会自动维护列统计信息以供优化程序使用。维护某些列的统计信息所获得的利益不会大于生成这些统计信息所需的开销。例如，如果不经常查询某列，或者此列受到最终会回退的定期大量更改的制约，则连续更新其统计信息的价值就很小。ALTER STATISTICS 语句可用于取消对这些类型列的统计信息的自动更新。

禁用自动更新之后，仍可以使用 CREATE STATISTICS 和 DROP STATISTICS 语句更新列的统计信息。但是，仅应在已确定更新这些统计信息会对性能有积极影响的情况下才进行更新。通常，不应禁用列统计信息。

权限

必须具有 DBA 权限。

副作用

如果禁用了自动更新，则统计信息可能会过期。重新启用自动更新不会立即使统计信息成为最新。如有必要，运行 CREATE STATISTICS 语句重新创建统计信息。

另请参见

- “CREATE STATISTICS 语句”一节第 491 页
- “DROP STATISTICS 语句”一节第 554 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例禁用 Customers 表中 Street 列统计信息的自动更新：

```
ALTER STATISTICS Customers ( Street ) AUTO UPDATE DISABLE;
```

ALTER SYNCHRONIZATION PROFILE 语句 [MobiLink]

此语句用于更改 SQL Anywhere 的同步配置文件。同步配置文件定义 SQL Anywhere 数据库如何与 MobiLink 服务器同步。

语法

ALTER SYNCHRONIZATION PROFILE *name*
{ **REPLACE** | **MERGE** } *string*

参数

- **name** 要变更的同步配置文件的名称。
- **REPLACE 子句** 此子句用于删除当前为配置文件定义的选项，并添加指定的替代选项。
- **MERGE 子句** 此子句用于更改同步配置文件的现有选项或向其中添加新选项。
- **string** 一个或多个同步选项值对的字符串用分号分隔。例如，`'option1=value1;option2=value2'`。

注释

有关 SQL Anywhere 所支持的同步配置文件选项的列表，请参见“CREATE SYNCHRONIZATION PROFILE 语句 [MobiLink]”一节第 493 页。

当在 ALTER SYNCHRONIZATION PROFILE 语句中使用了 **REPLACE** 时，整个同步配置文件将被指定的字符串替换。这与删除然后再创建效果相同。例如：

```
ALTER SYNCHRONIZATION PROFILE myProfile
REPLACE 'publication=p1;verbosity=high'
```


等效于：

```
DROP SYNCHRONIZATION PROFILE myProfile;
CREATE SYNCHRONIZATION PROFILE myProfile 'publication=p1;verbosity=high'
```

当在 ALTER SYNCHRONIZATION PROFILE 语句中使用了 **MERGE** 时，字符串中所指定的选项会被添加到同步配置文件中已存在的那些选项中。如果字符串中的选项在配置文件中已存在，那么字符串的值会替换已存储在配置文件中的值。

例如，执行以下语句会使配置文件 *myProfile* 具有值 *publication=p2;verbosity=high;uploadonly=on*。

```
CREATE SYNCHRONIZATION PROFILE myProfile 'publication=p1;verbosity=high';
ALTER SYNCHRONIZATION PROFILE myProfile MERGE 'publication=p2;uploadonly=on'
```

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE SYNCHRONIZATION PROFILE 语句 [MobiLink]”一节第 493 页
- “DROP SYNCHRONIZATION PROFILE 语句 [MobiLink]”一节第 556 页

标准和兼容性

- **SQL/2003** 服务商扩充。

ALTER SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]

在 SQL Anywhere 远程数据库中使用此语句可变更 MobiLink 用户对一发布所做预订的属性。

语法

ALTER SYNCHRONIZATION SUBSCRIPTION
TO *publication-name*
[**FOR** *ml_username*, ...]
[**TYPE** *network-protocol*]
[**ADDRESS** *protocol-options*]
[**ADD OPTION** *option=value*, ...]
[**ALTER OPTION** *option=value*, ...]
[**DELETE** { **ALL OPTION** | **OPTION** *option*, ... }]

ml_username: *identifier*

network-protocol: **http** | **https** | **tls** |**tcpip**

protocol-options: *string*

value: *string* | *integer*

参数

- **TO 子句** 指定发布名称。
- **FOR 子句** 指定一个或多个 MobiLink 用户名。

  忽略 FOR 子句以设置用于发布的协议类型、协议选项和扩展选项。

  有关 dbmlsync 如何处理在不同位置指定的选项的信息，请参见“优先级顺序”一节 《MobiLink - 客户端管理》。
- **TYPE 子句** 此子句指定同步中使用的网络协议。缺省协议为 tcpip。

  有关通信协议的详细信息，请参见“CommunicationType (ctp) 扩展选项”一节 《MobiLink - 客户端管理》。
- **ADDRESS 子句** 此子句指定网络协议选项，包括 MobiLink 服务器的位置。

  有关协议选项的完整列表，请参见“MobiLink 客户端网络协议选项汇总”一节 《MobiLink - 客户端管理》。
- **ADD OPTION、ALTER OPTION、DELETE OPTION 和 DELETE ALL OPTION 子句** 这些子句分别用来添加、变更、删除扩展选项或删除所有扩展选项。可以在每个子句中仅指定一个选项。

  每个选项的值都不能包含字符 "="、"," 或 ";"。

  有关选项的完整列表，请参见“MobiLink SQL Anywhere 客户端扩展选项” 《MobiLink - 客户端管理》。

注释

network-protocol、*protocol-options* 和 *options* 可在多处设置。

有关 dbmlsync 如何处理在不同位置指定的选项的信息，请参见“优先级顺序”一节 《MobiLink - 客户端管理》。

此语句会使各选项和其它信息存储在 SQL Anywhere ISYSSYNC 系统表中。具有数据库 DBA 权限的任何人都可查看这些信息，其中包括口令和加密证书。为避免发生这种潜在安全问题，可指定有关 dbmlsync 命令行的信息。

请参见“dbmlsync 语法”一节 《MobiLink - 客户端管理》。

权限

必须具有 DBA 权限。要求对发布中引用的所有表具有独占访问权限。

副作用

自动提交。

另请参见

- “CREATE PUBLICATION 语句 [MobiLink] [SQL Remote]”一节第 476 页
- “DROP PUBLICATION 语句 [MobiLink] [SQL Remote]”一节第 551 页
- SQL Anywhere MobiLink 客户端： “创建同步预订”一节 《MobiLink - 客户端管理》
- UltraLite MobiLink 客户端： “设计 UltraLite 中的同步”一节 《UltraLite - 数据库管理和参考》
- “ISYSSYNC 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例可更改 MobiLink 服务器的地址：

```
ALTER SYNCHRONIZATION SUBSCRIPTION
TO p1
FOR ml1
TYPE TCPIP
ADDRESS 'host=10.11.12.132;port=2439';
```

ALTER SYNCHRONIZATION USER 语句 [MobiLink]

此语句用于在 SQL Anywhere 远程数据库中变更 MobiLink 用户的属性。

语法

ALTER SYNCHRONIZATION USER *ml_username*
[**TYPE** *network-protocol*]
[**ADDRESS** *protocol-options*]
[**ADD OPTION** *option=value*, ...]
[**ALTER OPTION** *option=value*, ...]
[**DELETE** { **ALL OPTION** | **OPTION** *option* }]

ml_username: *identifier*

network-protocol: **http** | **https** | **tls** | **tcpip**

protocol-options: *string*

value: *string* | *integer*

参数

- **TYPE 子句** 此子句指定同步中使用的网络协议。

  有关通信协议的详细信息，请参见“CommunicationType (ctp) 扩展选项”一节 《MobiLink - 客户端管理》。

- **ADDRESS 子句** 此子句指定网络协议选项，包括 MobiLink 服务器的位置。

  有关协议选项的完整列表，请参见“MobiLink 客户端网络协议选项汇总”一节 《MobiLink - 客户端管理》。

- **ADD OPTION、ALTER OPTION、DELETE OPTION 和 DELETE ALL OPTION 子句** 这些子句分别用来添加、修改、删除扩展选项或删除所有扩展选项。可以在每个子句中仅指定一个选项。

  有关选项的完整列表，请参见“MobiLink SQL Anywhere 客户端扩展选项” 《MobiLink - 客户端管理》。

注释

network-protocol、*protocol-options* 和 *options* 可在多处设置。

有关 dbmlsync 如何处理在不同位置指定的选项的信息，请参见“优先级顺序”一节 《MobiLink - 客户端管理》。

此语句会使各选项和其它信息存储在 SQL Anywhere ISYSSYNC 系统表中。具有数据库 DBA 权限的任何人都可查看这些信息，其中包括口令和加密证书。为避免发生这种潜在安全问题，可指定有关 dbmlsync 命令行的信息。

请参见“dbmlsync 语法”一节 《MobiLink - 客户端管理》。

权限

必须具有 DBA 权限。要求对发布中引用的所有表具有独占访问权限。

副作用

自动提交。

另请参见

- “CREATE SYNCHRONIZATION USER 语句 [MobiLink]”一节第 496 页
- “DROP SYNCHRONIZATION USER 语句 [MobiLink]”一节第 558 页
- “MobiLink 用户” 《MobiLink - 客户端管理》
- “ISYSSYNC 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

ALTER TABLE 语句

此语句用于修改表定义、禁用相关视图或使表能够参与复制服务器复制。

语法

ALTER TABLE [*owner.*]*table-name* { *alter-clause*, ... }

alter-clause :
ADD *create-clause*
| **ALTER** *column-name column-alteration*
| **ALTER** [**CONSTRAINT** *constraint-name*] **CHECK (** *condition* **)**
| **DROP** *drop-object*
| **RENAME** *rename-object*
| *table-alteration*

create-clause :
column-name [**AS**] *column-data-type* [*new-column-attribute ...*]
| *table-constraint*
| **PCTFREE** *integer*

column-alteration :
{ *column-data-type* | *alterable-column-attribute* } [*alterable-column-attribute ...*]
| **SET COMPUTE (** *compute-expression* **)**
| **ADD** [*constraint-name*] **CHECK (** *condition* **)**
| **DROP** { **DEFAULT** | **COMPUTE** | **CHECK** | **CONSTRAINT** *constraint-name* }

drop-object :
column-name
| **CHECK**
| **CONSTRAINT** *constraint-name*
| **UNIQUE** [**CLUSTERED**] **(** *index-columns-list* **)**
| **FOREIGN KEY** *fkey-name*
| **PRIMARY KEY**

rename-object :
new-table-name
| *column-name* **TO** *new-column-name*
| **CONSTRAINT** *constraint-name* **TO** *new-constraint-name*

table-alteration :
PCTFREE DEFAULT
| **REPLICATE** { **ON** | **OFF** }
| [**NOT**] **ENCRYPTED**

new-column-attribute :
NULL
| **DEFAULT** *default-value*
| **COMPRESSED**
| **INLINE** { *inline-length* | **USE DEFAULT** }
| **PREFIX** { *prefix-length* | **USE DEFAULT** }
| [**NO**] **INDEX**
| **IDENTITY**
| **COMPUTE (** *expression* **)**
| *column-constraint*

table-constraint :
[**CONSTRAINT** *constraint-name*] {
  CHECK (*condition* **)**
  | **UNIQUE** [**CLUSTERED** | **NONCLUSTERED**] **(** *column-name* [**ASC** | **DESC**], ... **)**
  | **PRIMARY KEY** [**CLUSTERED** | **NONCLUSTERED**] **(** *column-name* [**ASC** | **DESC**], ... **)**
  | *foreign-key*
 }

column-constraint :
[**CONSTRAINT** *constraint-name*] {
  CHECK (*condition* **)**
  | **UNIQUE** [**CLUSTERED** | **NONCLUSTERED**] [**ASC** | **DESC**]
  | **PRIMARY KEY** [**CLUSTERED** | **NONCLUSTERED**] [**ASC** | **DESC**]
  | **REFERENCES** *table-name* [**(** *column-name* **)**]
    [**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]
    [*actions*][**CLUSTERED** | **NONCLUSTERED**]
  | **NOT NULL**
 }

alterable-column-attribute :
 [**NOT**] **NULL**
| **DEFAULT** *default-value*
| [**CONSTRAINT** *constraint-name*] **CHECK** { **NULL** | **(** *condition* **)** }
| [**NOT**] **COMPRESSED**
| **INLINE** { *inline-length* | **USE DEFAULT** }
| **PREFIX** { *prefix-length* | **USE DEFAULT** }
| [**NO**] **INDEX**

default-value :
special-value
| *string*
| *global variable*
| [**-**] *number*
| **(** *constant-expression* **)**
| *built-in-function* **(** *constant-expression* **)**
| **AUTOINCREMENT**
| **GLOBAL AUTOINCREMENT** [**(** *partition-size* **)**]
| **NULL**
| **TIMESTAMP**
| **UTC TIMESTAMP**
| **LAST USER**
| **USER**

special-value :
CURRENT {
DATABASE
| **DATE**
| **REMOTE USER**
| **TIME**
| **TIMESTAMP**
| **UTC TIMESTAMP**
| **USER**
| **PUBLISHER** }

foreign-key :
[**NOT NULL**] **FOREIGN KEY** [*role-name*]
[**(** *column-name* [**ASC** | **DESC**], ... **)**
REFERENCES *table-name*
[**(** *pkey-column-list* **)**]
[**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]
[*actions*] [**CHECK ON COMMIT**] [**CLUSTERED**]
[**FOR OLAP WORKLOAD**]

actions :
[**ON UPDATE** *action*] [**ON DELETE** *action*]

action :
CASCADE | **SET NULL** | **SET DEFAULT** | **RESTRICT**

语法 2 - 禁用视图依赖性

ALTER TABLE [*owner.*]*table-name* {
DISABLE VIEW DEPENDENCIES
}

参数

- **添加子句** 以下一节介绍用于为列或表添加定义的子句：
  - **ADD column-name [AS] column-data-type [new-column-attribute ...] 子句** 此语法用于向表中添加新列、为列指定数据类型和属性。有关要指定哪些数据类型的详细信息，请参见“SQL 数据类型”第 75 页。
  - **NULL 和 NOT NULL 子句** 此子句用于指定在列中是否允许 NULL。除了位类型列以外，新列允许 NULL 值。位类型列已在创建时自动应用 NOT NULL 约束。

- **DEFAULT 子句** 设置列的缺省值。列中的所有行都以此值填充。有关可能的缺省值的信息，请参见“CREATE TABLE 语句”一节第 497 页。
- **column-constraint 子句** 此子句用于向列添加约束。除 CHECK 约束以外，添加新约束后，数据库服务器会验证现有值以确认它们是否满足约束。仅对表变更完成之后的操作强制实行 CHECK 约束。可能的列约束包括：
  - **CHECK 子句** 此子句用于为列添加检查条件。
  - **UNIQUE 子句** 此子句用于指定列中的值必须唯一，并指定创建聚簇索引还是非聚簇索引。
  - **PRIMARY KEY 子句** 此次级子句用于将某列设为主键，并指定是否使用聚簇索引。有关聚簇索引的详细信息，请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。
  - **REFERENCES 子句** 此次级子句用于添加或变更对其它表的引用、指定如何处理匹配项以及指定是否使用聚簇索引。有关聚簇索引的详细信息，请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。
  - **NULL 和 NOT NULL 子句** 此子句用于指定在列中是否允许 NULL 值。缺省情况下，允许 NULL 值。
- **COMPRESSED 子句** 此子句用于压缩列。
- **INLINE 和 PREFIX 子句** 存储 BLOB（仅字符和二进制数据类型）时，INLINE 和 PREFIX 子句用于指定在行中保留多少 BLOB（以字节为单位）。有关详细信息，请参见“CREATE TABLE 语句”一节第 497 页中的 INLINE 和 PREFIX 子句。
- **INDEX 和 NO INDEX 子句** 此子句用于指定是否在此列中的大 BLOB 上构建索引。有关如何使用此子句的详细信息，请参见“CREATE TABLE 语句”一节第 497 页中 [NO] INDEX 子句的相应部分。
- **IDENTITY 子句** 此子句等效于 AUTOINCREMENT，且实现了与 T-SQL 的兼容。请参见“CREATE TABLE 语句”一节第 497 页中对 AUTOINCREMENT 的说明。
- **COMPUTE 子句** 此子句用于确保列中的值反映 *expression* 的值。有关 COMPUTE 子句允许哪些值的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。
- **ADD table-constraint 子句** 此子句用于添加表约束。表约束对表中可保存哪些列施加限制。添加或变更表约束时，可选约束名允许您修改或删除个别约束。以下是可添加的表约束列表。
  - **UNIQUE** 此次级子句用于指定 *column-list* 中所指定列的列值必须唯一，或者指定是否使用聚簇索引。有关此约束的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。
  - **PRIMARY KEY** 此次级子句用于添加或变更表的主键，及指定是否使用聚簇索引。表不得具有使用 CREATE TABLE 语句或其它 ALTER TABLE 语句创建的主键。有关此约束的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。

    有关聚簇索引的详细信息，请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **foreign-key** 此次级子句用于将外键作为约束添加。如果对具有相关实例化视图的表执行 ALTER TABLE 语句时使用了除 ADD FOREIGN KEY 之外的次级子句，则 ALTER TABLE 语句会失败。对于所有其它子句，必须禁用相关实例化视图，然后在更改完成后重新启用这些视图。

  有关此约束的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。

- **ADD PCTFREE 子句** 指定希望在每个表页中保留的可用空间的百分比。如果数据更新时行大小增加，将占用可用空间。如果表页中没有可用空间，则该页上的行大小每次增加时，行都需要在多个表页中拆分，从而导致行碎片并可能引起性能下降。如果可用空间百分比为 0，则指定每页不保留可用空间—每页均被完全填充。如果可用空间百分比较高，则每行都单独插入到页中。如果未设置 PCTFREE，或将其删除，则会根据数据库页大小应用缺省 PCTFREE 值（4 KB 及以上页大小应用 200 字节）。PCTFREE 的值存储在系统表 ISYSTAB 中。如果设置了 PCTFREE，所有表页的后续插入都使用新值，但是已插入的行不受影响。在重新设置之前，会持续使用该值。PCTFREE 规范可用于基表、全局临时表或局部临时表。

- **变更子句** 以下一节介绍用于变更列或表的定义的子句：

  - **ALTER column-name column-alteration 子句** 此子句用于更改指定列的属性。如果列包含在唯一约束、外键或主键中，则只能更改列的缺省属性。而对于任何其它更改，必须先删除键或约束，然后才能进行修改。以下是可进行的变更的列表。有关这些属性的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。

  - **column-data-type 子句** 此子句用于变更列的长度或数据类型。如有必要，会将已修改列中的数据转换为新的数据类型。如果发生转换错误，操作将失败，而表则保留不变。不能减小列大小。例如，不能将某列从 VARCHAR(100) 更改为 VARCHAR(50)。

  - **[NOT] NULL 子句** 此子句用于更改在列中是否允许 NULL。如果指定了 NOT NULL，而在任一现有行中列值为 NULL，则操作将失败，但表保持不变。

  - **CHECK NULL** 此子句用于删除列的所有检查约束。

  - **DEFAULT 子句** 此子句用于更改列的缺省值。

  - **DEFAULT NULL 子句** 此子句用于删除列的缺省值。

  - **[CONSTRAINT constraint-name] CHECK { NULL | (condition) } 子句** 此子句用于在列上添加 CHECK 约束。

  - **[NOT] COMPRESSED 子句** 此子句用于更改是否压缩列。

  - **INLINE 和 PREFIX 子句** INLINE 和 PREFIX 子句与包含 BLOB 的列一起使用，可以指定在一行中保留多少 BLOB（以字节为单位）。有关如何设置 INLINE 和 PREFIX 值的详细信息，请参见“CREATE TABLE 语句”一节第 497 页中 INLINE 和 PREFIX 子句的相应部分。

  - **INDEX 和 NO INDEX 子句** 此子句用于指定是否在此列中的大 BLOB 上构建索引。有关如何使用此子句的详细信息，请参见“CREATE TABLE 语句”一节第 497 页中 [NO] INDEX 子句的相应部分。

  - **SET COMPUTE 子句** 此子句用于更改与计算列相关联的表达式。当执行语句时，重新计算列中的值，如果新表达式无效，则语句执行失败。有关 COMPUTE 表达式允许哪些值的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。

- **ALTER CONSTRAINT constraint-name CHECK 子句** 此子句用于变更表的已命名检查约束。

- **删除子句** 以下一节介绍 DROP 子句：
  - **DROP DEFAULT** 删除表或指定列的缺省值设置。现有值不发生更改。
  - **DROP COMPUTE** 删除指定列的 COMPUTE 属性。此语句不更改表中的任何现有值。
  - **DROP CHECK** 删除表或指定列的所有 CHECK 约束。也可以使用 DELETE CHECK。
  - **DROP CONSTRAINT constraint-name** 删除表或指定列的已命名约束。也可以使用 DELETE CONSTRAINT。
  - **DROP column-name** 从表中删除指定列。也可以使用 DELETE *column-name*。如果列包含在任何索引、唯一约束、外键或主键中，则必须删除索引、约束或键后才能删除该列。这不会删除引用该列的 CHECK 约束。
  - **DROP UNIQUE (column-name ...)** 删除指定列上的唯一约束。同时也删除引用此唯一约束的任何外键。也可以使用 DELETE UNIQUE (*column-name* ...)。
  - **DROP FOREIGN KEY fkey-name** 删除指定的外键。也可以使用 DELETE FOREIGN KEY *fkey-name*。
  - **DROP PRIMARY KEY** 删除主键。同时也删除引用此表主键的所有外键。也可以使用 DELETE PRIMARY KEY。

- **重命名子句** 以下一节介绍用于为列或表的部分定义进行重命名的子句：
  - **RENAME new-table-name** 将表名更改为 *new-table-name*。注意，必要时必须修改任何使用旧表名的应用程序。重命名操作成功执行之后，必须删除并重新创建具有 ON UPDATE 或 ON DELETE 操作的外键，因为用于实现这些操作的系统创建的触发器将继续引用旧名称。
  - **RENAME column-name TO new-column-name** 将列名更改为 *new-column-name*。请注意，必要时需修改任何使用旧列名的应用程序。重命名操作成功执行之后，必须删除并重新创建具有 ON UPDATE 或 ON DELETE 操作的外键，因为用于实现这些操作的系统创建的触发器将继续引用旧名称。
  - **RENAME CONSTRAINT constraint-name TO new-constraint-name** 将约束名更改为 *new-constraint-name*。

- **变更表子句** 此子句用于变更表的以下属性。
  - **PCTFREE DEFAULT** 此子句用于将表的可用百分比设置更改为缺省值（4 KB 及以上页大小时为 200 字节）。
  - **REPLICATE { ON | OFF }** 此子句用于更改在复制期间是否包括表。当表有 REPLICATE ON 时，表的所有更改都发送到复制服务器进行复制。复制服务器中的复制定义用于决定将哪些表更改发送到其它站点。
  - **[NOT] ENCRYPTED** 此子句用于更改是否加密表。要对某个表加密，必须已在数据库中启用表加密。将使用在数据库创建时间所指定的加密密钥和算法来加密表。请参见“启用数据库中的表加密”一节 《SQL Anywhere 服务器 - 数据库管理》。对表进行加密之后，加密前的临时文件或事务日志中的此表的任何数据仍以未加密形式存在。为解决这个问题，重新启动数据库将临时文件删除。使用 -o 选项运行备份实用程序 (dbbackup)，或使用 BACKUP

语句备份事务日志并启动新事务日志。请参见“备份实用程序 (dbbackup)”一节 《SQL Anywhere 服务器 - 数据库管理》或“BACKUP 语句”一节第 390 页。

启用表加密时，将对加密表的表页、相关联的索引页、临时文件页，以及包含有关加密表的事务的事务日志页加密。

- **DISABLE VIEW DEPENDENCIES 子句** 使用此子句禁用相关常规视图。相关实例化视图不会被禁用；必须通过执行 ALTER MATERIALIZED VIEW ...DISABLE 语句禁用这些视图。请参见“ALTER MATERIALIZED VIEW 语句”一节第 358 页。

注释

ALTER TABLE 语句可更改现有表中的表属性（列定义、约束等等）。

数据库服务器会在数据库中跟踪对象依赖性。变更表的模式可能会影响相关视图。此外，如果存在依赖于要变更的表的实例化视图，必须先使用 ALTER MATERIALIZED VIEW ...DISABLE 语句禁用这些视图。有关视图依赖性的信息，请参见“视图依赖性”一节 《SQL Anywhere 服务器 - SQL 的用法》。

对于局部临时表不能使用 ALTER TABLE。

只要 ALTER TABLE 语句影响了当前正由其它连接使用的表，就会禁止该语句。ALTER TABLE 可能很耗时，并且在处理该语句时，数据库服务器不会处理引用表的其它请求。

有关使用 CLUSTERED 选项的详细信息，请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

如果变更在其上构建了定义为 IMMEDIATE REFRESH 文本索引的列，则会立即重建该文本索引。如果文本索引被定义为 AUTO REFRESH 或 MANUAL REFRESH，则会在下次刷新时重建文本索引。

执行 ALTER TABLE 语句后，数据库服务器会尝试恢复对自动重新编译的相关视图的列权限。对于在重新编译过的视图中不再存在的那些列，相应的列权限将会丢失。

权限

必须是以下之一：

- 表的所有者。
- 具有 DBA 权限的用户。
- 已被授予表的 ALTER 权限的用户。

ALTER TABLE 要求能够独占访问表。

除非已引用临时表的所有用户都断开了连接，否则无法变更全局临时表。

不能用于快照事务内。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

副作用

自动提交。

在 ALTER TABLE 操作的开始会执行检查点操作，且在 ALTER 操作完成之前会一直挂起其它检查点。

变更列或表之后，引用该变更列的任何存储过程、视图或其它项则不再有效。

如果更改某列已声明的长度或类型，或删除某列，则会删除该列的统计信息。有关如何生成新统计信息的信息，请参见“更新列统计信息以提高优化程序性能”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “CREATE TABLE 语句”一节第 497 页
- “DROP TABLE 语句”一节第 558 页
- “SQL 数据类型”第 75 页
- “变更表”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “特殊值”一节第 56 页
- “使用表和列约束”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “allow_nulls_by_default 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “启用数据库中的表加密”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** ADD COLUMN 是核心特性。其它子句是服务商扩充或对 SQL/2003 特定、已命名扩展的实现。

示例

以下示例向 Customers 表添加了新的时间戳列 TimeStamp。

```
ALTER TABLE Customers
   ADD TimeStamp AS TIMESTAMP DEFAULT TIMESTAMP;
```

以下示例删除了在前一个示例中添加的新时间戳列 TimeStamp。

```
ALTER TABLE Customers
DROP TimeStamp;
```

Customers 表中的 Street 列目前最多可以保存 35 个字符。要使该列最多可以保存 50 个字符，请执行以下语句：

```
ALTER TABLE Customers
ALTER Street CHAR(50);
```

以下示例在 Customers 表中添加一列，该列为每个客户指派一个销售联系人。

```
ALTER TABLE Customers
ADD SalesContact INTEGER
REFERENCES Employees ( EmployeeID )
ON UPDATE CASCADE
ON DELETE SET NULL;
```

此外键通过级联更新构造，并在删除时设置为 NULL。如果雇员更改他们的雇员 ID，列将更新以反映这一更改。如果雇员离开公司并且删除了其雇员 ID，则列设置为 NULL。

以下示例在 SalesOrders.SalesRepresentative 列创建了外键 FK_SalesRepresentative_EmployeeID2，并将它链接到 Employees.EmployeeID：

```
ALTER TABLE GROUPO.SalesOrders
   ADD CONSTRAINT FK_SalesRepresentative_EmployeeID2
```



```
FOREIGN KEY ( SalesRepresentative )
REFERENCES GROUPO.Employees (EmployeeID);
```

ALTER TEXT CONFIGURATION 语句

变更文本配置对象。

语法

ALTER TEXT CONFIGURATION [*owner.*]*config-name*
STOPLIST *stoplist*
| **DROP STOPLIST**
| { **MINIMUM** | **MAXIMUM** } **TERM LENGTH** *integer*
| **TERM BREAKER** { **GENERIC** | **NGRAM** }

stoplist : *string-expression*

参数

- **STOPLIST 子句** 此语句用于创建或替换构建文本索引时要忽略的术语列表。查询中也会忽略在列表中指定的术语。使用空格将非索引字表隔离。例如，`STOPLIST 'because about therefore only'`。非索引字表术语不能包含白空格。

  不同语言的非索引字表示例位于 *samples-dir\SQLAnywhere\SQL* 子目录中。有关 *samples-dir* 的位置，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。

  非索引字表术语不得包含非数字字母字符。

  请认真考虑是否将术语置于非索引字表中。有关详细信息，请参见“文本配置对象设置”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **DROP STOPLIST 子句** 此子句用于删除文本配置对象的非索引字表。

- **MINIMUM TERM LENGTH 子句** 使用 NGRAM 文本索引时将忽略在 MINIMUM TERM LENGTH 子句中指定的值。

  文本索引中包含的术语的最小长度（以字符数表示）。构建或刷新文本索引时，将忽略长度短于该设置的术语。该选项的值必须大于 0。如果将此选项设置为大于 MAXIMUM TERM LENGTH，则 MAXIMUM TERM LENGTH 的值会自动进行调整到与新的 MINIMUM TERM LENGTH 值相等。

- **MAXIMUM TERM LENGTH 子句** 对于 NGRAM 文本索引，在 MAXIMUM TERM LENGTH 子句中指定的值为 N。

  对于 GENERIC 文本索引，是文本索引中包含的术语的最大长度（以字符为单位）。构建或刷新文本索引时，将忽略长度大于该设置的术语。MAXIMUM TERM LENGTH 的值必须小于或等于 60。如果将此项设置为小于 MINIMUM TERM LENGTH，则 MINIMUM TERM LENGTH 的值会自动进行调整到与新的 MAXIMUM TERM LENGTH 的值相等。

- **TERM BREAKER 子句** 将列值分隔为术语时所用算法的名称。可以选择 GENERIC（缺省值）或 NGRAM。GENERIC 算法将所有由一个或多个字母数字构成并由非字母数字分隔的字符串均视为一个术语。NGRAM 算法将字符串分成 n 元语法词。一个 n 元语法词就是由较大字符串的 *n* 个字符组成的子串。模糊（近似）匹配或不使用空白字符或非字母数字字符分隔术语的文档

需要 NGRAM 术语断开器。有关这两个算法以及如何在它们之间进行选择的详细信息，请参见“文本配置对象设置”一节 《SQL Anywhere 服务器 - SQL 的用法》。

注释

在更改术语长度设置之前，请阅读有关各种设置对建立了索引的对象以及解释查询术语的方式的影响。请参见“文本配置对象设置”一节 《SQL Anywhere 服务器 - SQL 的用法》和“文本配置对象示例”一节 《SQL Anywhere 服务器 - SQL 的用法》。

文本索引依赖于文本配置对象。必须先截断相关文本索引，然后才可更改基础文本配置对象。如果相关文本索引属于快速刷新类型，则不能被截断，因此必须先删除这些索引，然后才能更改文本配置对象。

要确定引用了文本配置对象的文本索引，请参见“查看数据库中的文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

要查看文本配置对象的设置，请查询 SYSTEXTCONFIG 系统视图。请参见“SYSTEXTCONFIG 系统视图”一节第 979 页。

权限

必须是文本配置对象的所有者或者具有 DBA 权限。

副作用

自动提交

另请参见

- “文本配置对象设置”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “变更文本配置对象”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “教程：对 GENERIC 文本索引执行全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “教程：执行模糊全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE TEXT CONFIGURATION 语句”一节第 508 页
- “DROP TEXT CONFIGURATION 语句”一节第 559 页
- “sa_char_terms 系统过程”一节第 798 页
- “sa_nchar_terms 系统过程”一节第 866 页
- “sa_refresh_text_indexes 系统过程”一节第 877 页
- “sa_text_index_stats 系统过程”一节第 906 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句创建了文本配置对象 maxTerm16，然后将术语的最大长度更改为 16：

```
CREATE TEXT CONFIGURATION maxTerm16 FROM default_char;
ALTER TEXT CONFIGURATION maxTerm16
   MAXIMUM TERM LENGTH 16;
```

以下语句向 maxTerm16 配置对象添加了非索引字表：

```
ALTER TEXT CONFIGURATION maxTerm16
   STOPLIST 'because about therefore only';
```


ALTER TEXT INDEX 语句

变更文本索引的定义。

语法

ALTER TEXT INDEX [*owner.*]*text-index-name*
ON [*owner.*]*table-name*
alter-clause

alter-clause :
rename-object
| *refresh-alteration*

rename-object :
RENAME { **AS** | **TO** } *new-name*

refresh-alteration :
{ **MANUAL REFRESH**
| **AUTO REFRESH** [**EVERY** *integer* { **MINUTES** | **HOURS** }] }

参数

- **RENAME 子句** 使用 RENAME 子句重新命名文本索引。
- **REFRESH 子句** 指定 REFRESH 子句设置文本索引的刷新类型。有关此子句的选项的详细信息，请参见“CREATE TEXT INDEX 语句”一节第 509 页。

注释

文本索引创建后，不能将其更改为 IMMEDIATE REFRESH，或从 IMMEDIATE REFRESH 更改为其它值。如果要进行任何一种更改，必须先删除文本索引然后再重新创建。

要查看文本索引及其引用的文本配置对象，请参见“查看数据库中的文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

必须是基础表的所有者，或者具有 DBA 权限，或者具有 REFERENCES 权限。

当存在使用 WITH HOLD 子句打开的使用语句快照或事务快照的游标时，不能执行此语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

副作用

自动提交

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “更改文本索引概述”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “教程：对 GENERIC 文本索引执行全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “教程：执行模糊全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE TEXT INDEX 语句”一节第 509 页
- “ALTER TEXT INDEX 语句”一节第 383 页
- “DROP TEXT INDEX 语句”一节第 560 页
- “REFRESH TEXT INDEX 语句”一节第 668 页
- “TRUNCATE TEXT INDEX 语句”一节第 728 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

第一条语句创建 txt_index_manual 文本索引，同时将它定义为 MANUAL REFRESH。第二条语句将文本索引变更为每天自动刷新。第三条语句将文本索引重新命名为 txt_index_daily。

```
CREATE TEXT INDEX txt_index_manual ON MarketingInformation ( Description )
   MANUAL REFRESH;
ALTER TEXT INDEX txt_index_manual ON MarketingInformation
   AUTO REFRESH EVERY 24 HOURS;
ALTER TEXT INDEX txt_index_manual ON MarketingInformation
   RENAME AS txt_index_daily;
```

ALTER TRIGGER 语句

此语句用于用修改的版本替换触发器定义。

在 ALTER TRIGGER 语句中必须包含整个新的触发器定义。

语法 1

ALTER TRIGGER *trigger-name trigger-definition*

trigger-definition : CREATE TRIGGER syntax

语法 2

ALTER TRIGGER *trigger-name* **ON** [*owner.*] *table-name* **SET HIDDEN**

注释

语法 1 除第一个单词不同以外，ALTER TRIGGER 语句的语法与 CREATE TRIGGER 语句的语法相同。有关 *trigger-definition* 的信息，请参见“CREATE TRIGGER 语句”一节第 511 页和“CREATE TRIGGER 语句 [T-SQL]”一节第 516 页。

可以使用 CREATE TRIGGER 语法的 Transact-SQL 形式或 Watcom-SQL 形式。

语法 2 可以使用 SET HIDDEN 对关联触发器的定义进行模糊处理，使之不可读。可以卸载该触发器，然后将其重装到其它数据库中。如果使用 SET HIDDEN，则使用调试程序进行调试将不会显示触发器定义，也无法通过过程分析获得触发器定义。

注意
SET HIDDEN 操作是不可逆的。

权限

必须是定义触发器的表的所有者，或者是用户 DBA，或者具有表的 ALTER 权限以及具有 RESOURCE 权限。

副作用

自动提交。

另请参见

- “CREATE TRIGGER 语句”一节第 511 页
- “CREATE TRIGGER 语句 [T-SQL]”一节第 516 页
- “DROP TRIGGER 语句”一节第 561 页
- “隐藏过程、函数、触发器和视图的内容”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

ALTER USER 语句

使用此语句变更用户设置。

语法 1

ALTER USER *user-name* [**IDENTIFIED BY** *password*]
[**LOGIN POLICY** *policy-name*]
[**FORCE PASSWORD CHANGE** { **ON** | **OFF** }]

语法 2

ALTER USER *user-name*
[**RESET LOGIN POLICY**]

参数

- **user-name** 用户的名称。
- **IDENTIFIED BY 子句** 用户的口令。
- **policy-name** 指派给用户的登录策略的名称。如果不指定 LOGIN POLICY 子句则不进行任何更改。
- **FORCE PASSWORD CHANGE 子句** 控制用户登录时是否必须指定新口令。此设置将覆盖他们策略中的 password_expiry_on_next_login 选项设置。

- **RESET LOGIN POLICY 子句** 将用户登录策略的设置恢复为原始值。重置登录策略后，用户就可以访问由于超出登录策略选项的限制（如 max_failed_login_attempts 或 max_days_since_login）被锁定的帐户。

注释

用户 ID 和口令不能：

- 以空格、单引号或双引号开头
- 以空格结尾
- 含有分号

口令必须是有效的标识符，或者是用单引号引上的字符串（最大 255 字节）。口令区分大小写。建议采用由 7 位 ASCII 字符组成的口令，因为如果数据库服务器不能将客户端的字符集转换为 UTF-8，其它字符可能无法正常工作。

verify_password_function 选项可用于指定函数，以实现口令规则（例如，口令至少必须包含一位）。如果使用口令验证函数，则不能在 GRANT CONNECT 语句中指定多个用户 ID 和口令。请参见“verify_password_function 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

如果将 password_expiry_on_next_login 值设置为 ON，则当用户下次登录时用户口令会立即过期，即使为用户指派了同一策略。可使用 ALTER USER 和 LOGIN POLICY 子句强制用户更改他们下次登录的口令。

权限

任何用户都可以更改自己的口令。而所有其它的更改则要求具有 DBA 权限。

副作用

无。

另请参见

- “ALTER LOGIN POLICY 语句”一节第 357 页
- “COMMENT 语句”一节第 406 页
- “CREATE LOGIN POLICY 语句”一节第 453 页
- “CREATE USER 语句”一节第 518 页
- “DROP LOGIN POLICY 语句”一节第 548 页
- “DROP USER 语句”一节第 562 页
- “管理登录策略概述”一节 《SQL Anywhere 服务器 - 数据库管理》
- “为现有用户分配登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》
- “GRANT 语句”一节第 595 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例变更了一个名为 SQLTester 的用户。口令设置为 "welcome"。为 SQLTester 用户指派了 Test1 登录策略，而且下次登录时口令不到期。


```
ALTER USER SQLTester IDENTIFIED BY welcome
LOGIN POLICY Test1
FORCE PASSWORD CHANGE off;
```

ALTER VIEW 语句

此语句用于用修改的版本替换视图定义。

语法 1

ALTER VIEW
[*owner.*]*view-name* [**(** *column-name*, ... **)**] **AS** *select-statement*
[**WITH CHECK OPTION**]

语法 2

ALTER VIEW
[*owner.*]*view-name* { **SET HIDDEN** | **RECOMPILE** | **DISABLE** | **ENABLE** }

参数

- **AS 子句** 此子句的用途和语法与 CREATE VIEW 语句相同。请参见“CREATE VIEW 语句”一节第 520 页。
- **WITH CHECK OPTION 子句** 此子句的用途和语法与 CREATE VIEW 语句相同。请参见“CREATE VIEW 语句”一节第 520 页。
- **SET HIDDEN 子句** SET HIDDEN 子句用于将视图的定义进行模糊处理，并使该视图隐藏（例如，在 Sybase Central 中）。不过，对视图的显式引用仍有效。

  注意
  SET HIDDEN 操作是不可逆的。

- **RECOMPILE 子句** RECOMPILE 子句用于为视图重新创建列定义。此子句在功能上与 ENABLE 子句相同，只不过可以决定在未被禁用的视图上使用此子句。重新编译某个视图时，数据库服务器会根据在新视图定义中指定的列名称恢复列权限。如果某个列在重新编译后不再存在，则现有权限将会丢失。
- **DISABLE 子句** DISABLE 子句用于禁止数据库服务器使用视图。
- **ENABLE 子句** ENABLE 子句用于启用已禁用的视图。启用视图会使数据库服务器为视图重新创建列定义。启用视图之前，必须启用它所依赖的任何视图。

注释

如果变更另一用户拥有的视图，必须通过包含该所有者来限定名称（例如，GROUPO.ViewSalesOrders）。如果不限定名称，数据库服务器就会查找您拥有的该名称的视图，并对其进行变更。如果没有这样的视图，服务器将返回错误。

变更视图时，系统会保留视图的现有权限，不必重新指派。还可以分别使用 DROP VIEW 和 CREATE VIEW 语句代替 ALTER VIEW 语句，删除并重新创建视图。不过如果这样做，需要重新指派针对视图的权限。

使用语法 1 完成视图变更之后，数据库服务器会重新编译视图。根据所做更改的类型，如果存在相关视图，数据库服务器还将尝试重新编译这些视图。如果已进行了影响相关视图的更改，可能还需要变更相关视图的定义。有关视图变更及其如何影响视图依赖性的详细信息，请参见“视图依赖性”一节 《SQL Anywhere 服务器 - SQL 的用法》。

小心
如果定义视图的 SELECT 语句包含一个星号 (*)，则视图中的列数会随着基础表中列的增加或删除而变化。视图中列的名称和数据类型也会发生更改。

语法 1 此语法用于变更视图的结构。变更视图结构与变更表不同（变更表时，仅限于对个别列进行更改），它要求用新定义替换整个视图定义，与创建视图很相似。有关用于定义视图结构的参数的说明，请参见“CREATE VIEW 语句”一节第 520 页。

语法 2 此语法用于更改视图的属性，例如是否隐藏视图定义。

使用 SET HIDDEN 时，视图可被卸载并重装到其它数据库中。如果使用 SET HIDDEN，则使用调试程序进行调试将不会显示视图定义，也无法通过过程分析获得视图定义。如果需要更改隐藏视图的定义，则必须删除该视图并使用 CREATE VIEW 语句重新创建。

使用 DISABLE 子句时，数据库服务器便无法再使用此视图来回应查询。禁用视图与删除视图类似，只是视图定义仍保留在数据库中。禁用视图还会禁用任何相关视图。因此，DISABLE 子句不仅要求对被禁用的视图具有独占访问权限，还要求对相关视图具有独占访问权限，因为这些相关视图也会被禁用。

权限

必须是视图的所有者或者有 DBA 权限。

副作用

自动提交。

所有的过程和触发器都从内存中卸载，这样，引用视图的任何过程或触器都反映新的视图定义。如果定期变更视图，则卸载和装载过程和触发器会影响性能。

另请参见

- “CREATE VIEW 语句”一节第 520 页
- “DROP VIEW 语句”一节第 563 页
- “隐藏过程、函数、触发器和视图的内容”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “视图依赖性”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE MATERIALIZED VIEW 语句”一节第 455 页
- “ALTER MATERIALIZED VIEW 语句”一节第 358 页

标准和兼容性

- **SQL/2003** 服务商扩充。

ATTACH TRACING 语句

此语句用于启动诊断跟踪会话。即开始向诊断表发送诊断信息。

语法

ATTACH TRACING TO { **LOCAL DATABASE** | *connect-string* }
[**LIMIT** { *size* | *history* }]

connect-string : the connection string for the database

size : **SIZE** *nnn* { **MB** | **GB** }

history : **HISTORY** *nnn* { **MINUTES** | **HOURS** | **DAYS** }

nnn : *integer*

参数

- **connect-string** 与接收跟踪信息的数据库连接所需的连接字符串。仅当要分析的数据库与接收数据的数据库不同时才需要此参数。

  connect-string 中允许使用以下参数：DBF、DBKEY、DBN、ENG、LINKS、PWD、UID。

  相对于数据库服务器位置指定 DBF。否则，数据库服务器会尝试在当前服务器计算机上启动具有该名称的数据库。

  如果指定带有 LINKS 或 ENG 连接参数的 DBF 参数，则会返回一条错误。

- **LIMIT 子句** 存储在跟踪数据库中的数据量限制（按大小或是按时间长度）。

注释

ATTACH TRACING 语句主要由 Sybase Central 中的跟踪向导使用。但是，也可以手工运行它。必须从要分析的数据库中运行它。

ATTACH TRACING 语句用于为要分析的数据库启动跟踪会话。仅在设置了跟踪级别之后才可使用此语句。可以使用 Sybase Central 或使用 sa_set_tracing_level 系统过程设置跟踪级别。

启动会话之后，会根据 sa_diagnostic_tracing_level 表中所设置的跟踪级别生成跟踪信息。通过指定 LOCAL DATABASE，可以向要分析的同一数据库内的跟踪表发送跟踪数据。或者，可通过指定连接此数据库的连接字符串 (*connect-string*)，将跟踪数据发送到单独的跟踪数据库中。跟踪数据库必须已经存在，且您必须具有访问该数据库权限。

使用 LIMIT SIZE 或 LIMIT HISTORY 子句可限制要存储的跟踪数据的量。若要将跟踪数据量限制为一定大小，可使用 LIMIT SIZE 子句（以兆字节或千兆字节计量）。LIMIT HISTORY 子句用于将跟踪数据量限制为一段时间（以分、小时或天计量）。例如，`HISTORY 8 DAYS` 将存储在跟踪数据库中的跟踪数据量限制为 8 天的数据量。

要启动跟踪会话，TCP/IP 必须在运行跟踪数据库和生产数据库的数据库服务器上运行。请参见“使用 TCP/IP 协议”一节 《SQL Anywhere 服务器 - 数据库管理》。

包含潜在敏感数据的包在网络接口上是可见的，即使跟踪本地数据库时也是如此。为安全起见，可以在连接字符串中指定加密。

若要确认数据库的当前跟踪级别，请查看 sa_diagnostic_tracing_level 表。请参见“sa_diagnostic_tracing_level 表”一节第 780 页。

要查看发送跟踪数据的目标位置，请检查 SendingTracingTo 数据库属性。请参见“数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

必须连接到要分析的数据库，且必须具有 DBA 或 PROFILE 权限。

副作用

无。

另请参见

- “DETACH TRACING 语句”一节第 539 页
- “REFRESH TRACING LEVEL 语句”一节第 669 页
- “使用诊断跟踪进行高级应用程序分析”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_set_tracing_level 系统过程”一节第 898 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例使用 sa_set_tracing_level 系统过程将跟踪级别设置为 1。然后它会启动跟踪会话。为本地数据库生成的跟踪数据将被发送到其它计算机上的 mytracingdb（如指定的连接字符串所显示）跟踪数据库中。跟踪会话期间，系统最多会保留两小时的跟踪数据量。请注意，ATTACH TRACING 语句示例的所有内容都在一行上。

```
CALL sa_set_tracing_level( 1 );
ATTACH TRACING TO
'uid=DBA;pwd=sql;eng=remotedbsrv11;dbn=mytracingdb;links=tcpip'
 LIMIT HISTORY 2 HOURS;
```

BACKUP 语句

此语句用于备份数据库和事务日志。

语法 1（映像备份）

BACKUP DATABASE
DIRECTORY *backup-directory*
[**WAIT BEFORE START**]
[**WAIT AFTER END**]
[**DBFILE ONLY**]
[**TRANSACTION LOG ONLY**]
[**TRANSACTION LOG RENAME** [**MATCH**]]
[**TRANSACTION LOG TRUNCATE**]
[**ON EXISTING ERROR**]
[**WITH COMMENT** *comment string*]
[**HISTORY** { **ON** | **OFF** }]
[**AUTO TUNE WRITERS** { **ON** | **OFF** }]
[**WITH CHECKPOINT LOG** { **AUTO** | **COPY** | **NO COPY** | **RECOVER** }]

backup-directory : { *string* | *variable* }

语法 2（档案备份）

BACKUP DATABASE TO *archive-root*
[**WAIT BEFORE START**]
[**WAIT AFTER END**]
[**DBFILE ONLY**]
[**TRANSACTION LOG ONLY**]
[**TRANSACTION LOG RENAME** [**MATCH**]]
[**TRANSACTION LOG TRUNCATE**]
[**ATTENDED** { **ON** | **OFF** }]
[**WITH COMMENT** *comment string*]
[**HISTORY** { **ON** | **OFF** }]
[**WITH CHECKPOINT LOG** [**NO**] **COPY**]
[**MAX WRITE** { *number-of-writers* | **AUTO** }]

archive-root : { *string* | *variable* }

comment-string : *string*

number-of-writers : *integer*

参数

- **DIRECTORY 子句** 备份文件在磁盘上的目标位置，相对于数据库服务器启动时的当前目录。如果该目录不存在，则会进行创建。如果指定空字符串作为目录，则不必先复制日志就可以重命名或截断该日志。使用数据库镜像时，请不要使用此子句。请参见“数据库镜像和事务日志文件”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **WAIT BEFORE START 子句** 此子句用于确保数据库的备份副本不包含恢复所需的任何信息。特别是，它确保每个连接的回退日志为空。

  如果使用此子句执行了备份，则可以采用只读模式启动数据库的备份副本并对其进行校验。通过启用备份数据库校验，可以避免生成其它的数据库副本。
- **WAIT AFTER END 子句** 如果要重命名或截断事务日志，则使用此子句。此语句确保在重命名或截断日志前完成所有事务。如果使用此子句，则备份必须等到其它连接提交或者回退任何打开的事务时才能结束。
- **DBFILE ONLY 子句** 此子句可用于生成主数据库文件和任何关联的 dbspace 的备份副本。但不复制事务日志。不能将 DBFILE ONLY 子句与 TRANSACTION LOG RENAME 或 TRANSACTION LOG TRUNCATE 子句一起使用。
- **TRANSACTION LOG ONLY 子句** 此子句用于生成事务日志的备份副本。但不复制其它数据库文件。
- **TRANSACTION LOG RENAME [MATCH] 子句** 使用此子句会使数据库服务器在完成备份时重命名当前事务日志。如果省略关键字 MATCH，则日志的备份副本将与数据库的当前事务日志同名。如果使用了关键字 MATCH，则事务日志的备份副本的名称格式为 *YYMMDDnn.log*，以与当前事务日志的重命名副本匹配。使用关键字 MATCH 使得同一语句可执行多次而不会覆盖旧的数据。

- **TRANSACTION LOG TRUNCATE 子句** 如果使用该子句，当前事务日志将被截断，并在完成备份时重新启动。使用数据库镜像时，请不要使用此子句。请参见“数据库镜像和事务日志文件”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **archive-root 子句** 档案文件的文件名或磁带驱动器设备名。

  要备份到磁带，必须指定磁带驱动器的设备名。例如，在 NetWare 上，第一个磁带驱动器是 \\.*tape0*。每执行一次档案备份，自动附加到档案文件名末尾的编号就会增加。

  反斜线 (\ ) 是 SQL 字符串中的转义字符，因此每次必须使用两个反斜线。有关转义字符和字符串的详细信息，请参见“字符串”一节第 9 页。

- **ON EXISTING ERROR 子句** 此子句仅适用于映像备份。缺省情况下，现有文件将在执行 BACKUP DATABASE 语句时被覆盖。如果使用了此子句，则当其中任一将由备份创建的文件已存在时，会出现错误。

- **ATTENDED 子句** 该子句仅在备份到磁带设备时适用。ATTENDEDATTENDED ATTENDED ON（缺省值）表示有人可以监视磁带驱动器的状态，必要时在驱动器中放入新磁带。如果磁带驱动器要求干预，系统会将一条消息将发送到发出 BACKUP DATABASE 语句的应用程序。然后数据库服务器等待驱动器就绪。这在某些情况下会发生，例如在需要新磁带时。

  如果指定 ATTENDED OFF 且需要新磁带，或者驱动器尚未就绪，则不发送消息，并给出错误。

- **WITH COMMENT 子句** 此子句会将注释记录到备份历史记录文件中。对于档案备份，还会将注释记录到档案文件中。

- **HISTORY 子句** 缺省情况下，每次备份操作都会在 *backup.syb* 文件中附加一行。通过指定 HISTORY OFF 可以阻止更新 *backup.syb* 文件。如果符合以下任意条件，最好阻止更新该文件：

  - 经常备份
  - 没有定期存档或删除 *backup.syb* 文件的过程
  - 磁盘空间非常有限

- **AUTO TUNE WRITERS 子句** 开始备份后，有一个线程专用于将备份文件写到备份目录中。但是，如果备份目录所在的设备（例如 RAID 阵列）可处理递增写入程序装载，则可通过增加用作写入程序的线程数来提高整体备份性能。如果此子句设置为 ON（缺省值），则数据库服务器会定期检查所有参与备份的设备的读写性能。如果通过创建其它写入程序可提高整体备份速度，则数据库服务器会创建其它写入程序。

- **WITH CHECKPOINT LOG 子句** 此子句指定备份在将数据库文件写入目标目录之前处理这些数据库文件的方式。选择在备份期间是应用前映像还是复制检查点日志会对性能产生影响。缺省设置为 AUTO（映像备份）和 COPY（档案备份）。

- **COPY 子句** 此选项不能与 BACKUP 语句的 WAIT BEFORE START 子句一起使用。

  如果指定 COPY，则备份期间会读取数据库文件，但不应用任何已修改的页。整个检查点日志以及系统 dbspace 会复制到备份目录中。下次启动数据库服务器时，数据库服务器会自动将数据库恢复到备份开始时在检查点所处的状态。

  因为页无需写入临时文件，所以使用此选项可实现更好的备份性能，并能减少服务器对备份期间正在运行的其它连接的内部争用。但是，由于检查点日志包含修改页的原始映像，因此若数据库进行了更新，该日志就会增大。若指定 COPY，则数据库文件的备份副本可能要比备份开始时的数据库文件大。COPY 选项应在目标目录不存在磁盘空间问题的情况下使用。

- **NO COPY 子句** 如果指定 NO COPY，则不会将检查点日志作为备份的一部分进行复制。如果选择此选项，经过修改的页会被保存在临时文件中，这样便可在备份过程中应用这些页。数据库文件的备份副本与备份操作开始时的数据库大小相同。

  使用此选项可以减小备份的数据库文件，但会减缓备份过程，并有可能降低数据库服务器中其它操作的性能。它在目标驱动器空间有限的情况下非常有用。

- **RECOVER 子句** 如果指定 RECOVER，数据库服务器会复制检查点日志（与 COPY 选项一样），但会在备份结束时将检查点日志应用于数据库。这样会将备份的数据库文件恢复到备份操作开始时的状态（和大小）。此选项在备份驱动器空间有限的情况下非常有用（它备份检查点日志所需的空间量与 COPY 选项相同，但生成的文件会小一些）。

- **AUTO 子句** 如果指定 AUTO，数据库服务器会检查备份目录所在卷的可用磁盘空间量。备份开始时，如果可用磁盘空间至少是数据库大小的两倍，则使用此选项会像指定了 COPY 一样工作。否则，会像指定了 NO COPY 一样工作。AUTO 为缺省行为。

- **MAX WRITE 子句** 对于档案备份，缺省情况下有一个线程专用于写备份文件。如果备份目录所在的设备（例如 RAID 阵列）可处理递增写入程序装载，则可通过增加用作写入程序的线程数来提高整体备份性能。

  如果指定了 AUTO，将会为每个读取程序线程分别创建一个输出流。值 *n* 指定可创建的最多输出流数，最大数目为读取程序线程数。该子句的缺省值为 1。如果要备份到磁带，则只能使用一个写入程序。

  第一个流（流 0）将生成名为 *myarchive.X* 的文件，其中 *X* 是从 1 开始并一直增大为所需文件数的一个数字。所有其它流将生成名为 *myarchive.Y.Z* 的文件，其中 *Y* 是流编号（从 1 开始），*Z* 是从 1 开始并一直增大为所需文件数的一个数字。

注释

BACKUP 语句执行服务器端备份。要执行客户端备份，请使用 dbbackup 实用程序。请参见“备份实用程序 (dbbackup)”一节 《SQL Anywhere 服务器 - 数据库管理》。

每个备份操作（无论是映像还是档案）都更新名为 *backup.syb* 的历史记录文件。此文件记录已在数据库服务器上执行的 BACKUP 和 RESTORE 操作。有关如何确定 *backup.syb* 文件位置的信息，请参见“SALOGDIR 环境变量”一节 《SQL Anywhere 服务器 - 数据库管理》。

要创建不必通过恢复就可以在只读服务器上启动的备份，必须同时使用 WAIT BEFORE START 和 WITH CHECKPOINT LOG NO COPY 子句。WAIT BEFORE START 子句可确保回退日志为空，WITH CHECKPOINT LOG NO COPY 子句可确保检查点日志为空。如果两个文件中任何文件丢失，则需要恢复。

语法 1（映像备份） 映像备份创建每个数据库文件的副本，采用的方式与备份实用程序 (dbbackup) 相同。缺省情况下，备份实用程序将在客户端计算机上制作备份，但在使用备份实用程序时也可以通过指定 -s 选项在数据库服务器上创建备份。但如果使用 BACKUP DATABASE 语句，则只能在数据库服务器上制作备份。

或者，仅可保存数据库文件或事务日志。备份完成后也可以重命名或截断日志。

或者，可以指定空字符串作为目录，这样不必事先复制日志就可以重命名或截断它。这在需要考虑空间大小的复制环境中非常有用。您可以将此功能与事务日志大小的事件处理程序结合使用，以便在日志达到给定大小时将其重命名，还可以将此功能与 delete_old_logs 选项结合使用，以便删除不再需要的日志。

要从映像备份恢复，请将保存的文件复制回原来的位置并重新应用事务日志，如“通过多个事务日志恢复数据库”一节 《SQL Anywhere 服务器 - 数据库管理》中所述。

语法 2（档案备份） 档案备份创建单个文件来保存所有必需的备份信息。目标可以是文件名或磁带驱动器设备名。

一个给定的磁带上仅能有一个备份。磁带在备份结束时弹出。

每个磁带上只能有一个档案，但是一个档案可以跨转多个磁带。要从档案备份恢复数据库，可使用 RESTORE DATABASE 语句。

如果某个 RESTORE DATABASE 语句引用了仅包含一个事务日志的档案文件，则该语句必须指定一个文件名（表明恢复的数据库文件所在的位置），即使该文件不存在。例如，若要从仅包含一个日志的档案恢复到目录 *C:\MYNEWDB*，RESTORE DATABASE 语句为：

```
RESTORE DATABASE 'c:\mynewdb\my.db' FROM archive-root
```

小心

不得以任何方式更改数据库和事务日志的备份副本。如果在备份过程中未执行任何事务，或者指定了 BACKUP DATABASE WITH CHECKPOINT LOG RECOVER 或 WITH CHECKPOINT LOG NO COPY，则可以使用只读模式或通过验证备份数据库的副本来验证备份数据库的有效性。

但是，如果有正在执行的事务，或者指定了 BACKUP DATABASE WITH CHECKPOINT LOG COPY，则当您启动数据库服务器时，数据库服务器必须执行数据库恢复。恢复功能会修改备份副本，这是不希望出现的情况。

权限

必须具有 DBA、REMOTE DBA 或 BACKUP 权限。

副作用

导致检查点。

另请参见

- “备份实用程序 (dbbackup)”一节 《SQL Anywhere 服务器 - 数据库管理》
- “映像备份”一节 《SQL Anywhere 服务器 - 数据库管理》
- “RESTORE DATABASE 语句”一节第 676 页
- “备份和数据恢复” 《SQL Anywhere 服务器 - 数据库管理》
- “EXECUTE IMMEDIATE 语句 [SP]”一节第 570 页
- “了解并行数据库备份”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。
- **Windows Mobile** 在 Windows Mobile 上仅支持 BACKUP DATABASE DIRECTORY 语法（上面的语法 1）。

示例

将当前数据库和事务日志分别备份到不同文件，并重命名现有事务日志。会创建一个映像备份。


```
BACKUP DATABASE
DIRECTORY 'd:\\temp\\backup'
TRANSACTION LOG RENAME;
```

用于重命名事务日志的选项在仍需要旧事务日志的复制环境中特别有用。

将当前数据库和事务日志备份到磁带：

```
BACKUP DATABASE
TO '\\\\.\\tape0';
```

重命名日志，但不创建副本：

```
BACKUP DATABASE DIRECTORY ' '
TRANSACTION LOG ONLY
TRANSACTION LOG RENAME;
```

使用动态构建的目录名执行 BACKUP DATABASE 语句：

```
CREATE EVENT NightlyBackup
SCHEDULE
START TIME '23:00' EVERY 24 HOURS
HANDLER
BEGIN
    DECLARE dest LONG VARCHAR;
    DECLARE day_name CHAR(20);

    SET day_name = DATENAME( WEEKDAY, CURRENT DATE );
    SET dest = 'd:\\backups\\' || day_name;
            BACKUP DATABASE DIRECTORY dest
    TRANSACTION LOG RENAME;
END;
```

BEGIN 语句

此语句用于将 SQL 语句组合在一起。

语法

[*statement-label* :]
BEGIN [[**NOT**] **ATOMIC**]
  [*local-declaration;* ...]
  statement-list
  [**EXCEPTION** [*exception-case* ...]]
END [*statement-label*]

local-declaration :
 variable-declaration
| *cursor-declaration*
| *exception-declaration*
| *temporary-table-declaration*

variable-declaration :
DECLARE *variable-name data-type*

exception-declaration :
DECLARE *exception-name* **EXCEPTION**
FOR SQLSTATE [**VALUE**] *string*

exception-case :
WHEN *exception-name* [, ...] **THEN** *statement-list*
| **WHEN OTHERS THEN** *statement-list*

参数

- **local-declaration** 紧接在 BEGIN 后，复合语句可以具有适用于仅在复合语句内使用的对象的本地声明。复合语句可以有变量、游标、临时表或异常的本地声明。局部声明可由该复合语句中的任何语句引用，或者可由该复合语句内嵌套的任何复合语句中的任何语句引用。本地声明对于从复合语句中调用的其它过程不可见。
- **statement-label** 如果指定结尾 *statement-label*，则它必须与起始 *statement-label* 匹配。LEAVE 语句可用于在复合语句后的第一个语句处继续执行。复合语句是过程或触发器的主体，它具有与过程或触发器同名的隐式标签。

  有关复合语句和异常处理的完整说明，请参见“过程和触发器中的错误和警告”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **ATOMIC 子句** 原子语句是完全执行或根本不执行的语句。例如，更新数千行的 UPDATE 语句在更新许多行后可能会遇到错误。如果该语句没有完成，则所有更改都恢复为其原始状态。同样，如果指定 BEGIN 语句为原子语句，则此语句要么完全执行，要么根本不执行。

注释

过程或触发器的主体是复合语句。复合语句也可以用于过程或触发器内的控制语句。

复合语句允许一个或多个 SQL 语句组合在一起，按一个单元处理。复合语句以关键字 BEGIN 开始，以关键字 END 结束。

权限

无。

副作用

无。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “DECLARE LOCAL TEMPORARY TABLE 语句”一节第 529 页
- “CONTINUE 语句 [T-SQL]”一节第 413 页
- “SIGNAL 语句”一节第 709 页
- “RESIGNAL 语句”一节第 675 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》
- “原子复合语句”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

过程或触发器的主体是复合语句。


```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
   DECLARE err_notfound EXCEPTION FOR
      SQLSTATE '02000';
   DECLARE curThisCust CURSOR FOR
      SELECT CompanyName, CAST(
            sum( SalesOrderItems.Quantity *
            Products.UnitPrice ) AS INTEGER) VALUE
      FROM Customers
            LEFT OUTER JOIN SalesOrders
            LEFT OUTER JOIN SalesOrderItems
            LEFT OUTER JOIN Products
      GROUP BY CompanyName;
   DECLARE ThisValue INT;
   DECLARE ThisCompany CHAR( 35 );
   SET TopValue = 0;
   OPEN curThisCust;
   CustomerLoop:
   LOOP
      FETCH NEXT curThisCust
         INTO ThisCompany, ThisValue;
      IF SQLSTATE = err_notfound THEN
         LEAVE CustomerLoop;
      END IF;
      IF ThisValue > TopValue THEN
         SET TopValue = ThisValue;
         SET TopCompany = ThisCompany;
      END IF;
   END LOOP CustomerLoop;
   CLOSE curThisCust;
END;
```

BEGIN SNAPSHOT 语句

此语句用于以指定的时间间隔启动快照，以与快照隔离事务一起使用。

语法

BEGIN SNAPSHOT

注释

缺省情况下，当事务开始时，数据库服务器会推迟创建快照，直到应用程序导致表的第一行被读取。可使用 BEGIN SNAPSHOT 语句在事务中更早启动快照。当快照事务执行 BEGIN SNAPSHOT 语句时，数据库服务器即创建快照。

当遇到以下任一种情况时语句失败并返回错误：

- 尚未启用数据库的快照事务支持。请参见“allow_snapshot_isolation 选项 [数据库]”一节《SQL Anywhere 服务器 - 数据库管理》。
- 对当前事务已启动了快照。

对于非快照事务此语句也很有用，因为此语句允许非快照事务启动随后可在针对语句级快照操作的事务中使用的快照。

权限

无。

副作用

无。

另请参见

- “快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

BEGIN TRANSACTION 语句 [T-SQL]

此语句用于开始用户定义的事务。

语法

BEGIN TRAN[SACTION] [*transaction-name*]

注释

可选参数 *transaction-name* 是指派给此事务的名称。它必须是有效的标识符。仅在最外层的嵌套 BEGIN/COMMIT 语句对或 BEGIN/ROLLBACK 语句对中使用事务名称。

在事务内部执行 BEGIN TRANSACTION 语句时，该语句会将事务嵌套级别增加一级。通过 COMMIT 语句可减少嵌套级别。当事务被嵌套时，只有最外层的 COMMIT 可使数据库更改永久生效。

Adaptive Server Enterprise 和 SQL Anywhere 都具有两种事务模式。

缺省的 Adaptive Server Enterprise 事务模式称为非链接模式，它单独提交每个语句，除非执行显式 BEGIN TRANSACTION 语句启动事务。相反，与 ISO SQL/2003 兼容的链接模式仅当执行显式 COMMIT 时或执行能完成自动提交的语句（如数据定义语句）时才提交事务。

通过设置 chained 数据库选项可以控制模式。在 SQL Anywhere 中，ODBC 和嵌入式 SQL 连接的缺省设置是 ON，这种情况下，SQL Anywhere 在链接模式下运行。（ODBC 用户还应该检查 AutoCommit ODBC 设置）。TDS 连接的缺省设置是 Off。请参见“chained 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

在非链接模式下，事务在任何数据检索或修改语句前隐式启动。这些语句包括：DELETE、INSERT、OPEN、FETCH、SELECT 和 UPDATE。但仍必须用 COMMIT 或 ROLLBACK 语句显式结束事务。

不能在事务中变更 chained 选项。

> **小心**
> 当调用存储过程时，应该确保它在要求的事务模式下正确运行。

当前嵌套级别保存在全局变量 @@trancount 中。在执行第一个 BEGIN TRANSACTION 语句前，@@trancount 变量的值为 0，而仅当 @@trancount 等于 1 时执行 COMMIT 才会对数据库进行永久更改。

没有事务名称或保存点名称的 ROLLBACK 语句总是将语句回退到最外层的 BEGIN TRANSACTION（显式或隐式）语句，并且取消整个事务。

权限

无。

副作用

无。

另请参见

- “COMMIT 语句”一节第 408 页
- “isolation_level 选项 [数据库] [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “ROLLBACK 语句”一节第 684 页
- “SAVEPOINT 语句”一节第 688 页
- “事务内的保存点”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下批处理语句会报告 @@trancount 的连续值为 0、1、2、1 和 0。这些值显示在数据库服务器消息窗口上。

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT
PRINT @@trancount
COMMIT
PRINT @@trancount
```

@@trancount 的值只可用于跟踪已发出的显式 BEGIN TRANSACTION 语句的数量。

当 Adaptive Server Enterprise 隐式启动事务时，会将 @@trancount 变量设置为 1。而当 SQL Anywhere 隐式启动事务时，不会将 @@trancount 变量值设置为 1。因此，在任何 BEGIN TRANSACTION 语句前（即使存在当前事务），SQL Anywhere @@trancount 变量的值均为 0，而在 Adaptive Server Enterprise（在链接模式下）中，该变量的值为 1。

对于使用 BEGIN TRANSACTION 语句启动的事务，在执行第一个 BEGIN TRANSACTION 语句之后，在 SQL Anywhere 和 Adaptive Server Enterprise 中，@@trancount 的值均为 1。如果事务使用其它语句隐式启动，然后执行 BEGIN TRANSACTION 语句，则执行 BEGIN TRANSACTION 语句后，在 SQL Anywhere 和 Adaptive Server Enterprise 中，@@trancount 的值均为 2。

BREAK 语句 [T-SQL]

此语句用于退出复合语句或循环。

语法

BREAK

注释

BREAK 语句是一个控制语句，可利用它退出循环。执行过程从该循环之后的第一个语句开始继续。

权限

无。

副作用

无。

另请参见

- “WHILE 语句 [T-SQL]”一节第 748 页
- “CONTINUE 语句 [T-SQL]”一节第 413 页
- “BEGIN 语句”一节第 395 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** Transact-SQL 扩展。

示例

在此示例中，如果最贵的产品价格大于 $50，BREAK 语句会中断 WHILE 循环。否则，循环将继续直到平均价格高于或等于 $30：

```
WHILE ( SELECT AVG( UnitPrice ) FROM Products ) < $30
BEGIN
   UPDATE Products
   SET UnitPrice = UnitPrice + 2
   IF ( SELECT MAX(UnitPrice) FROM Products ) > $50
      BREAK
END
```

CALL 语句

此语句用于调用过程。

语法 1

[*variable* =] **CALL** *procedure-name* **(** [*expression*, ...] **)**

语法 2

[*variable* =] **CALL** *procedure-name* **(** [*parameter-name* = *expression*, ...] **)**

注释

CALL 语句调用以前用 CREATE PROCEDURE 语句创建的过程。当过程完成后，会复制回任意 INOUT 或 OUT 参数值。

可以按位置或使用关键字格式指定参数列表。通过位置，参数与过程的参数列表中相对应的参数匹配。如果使用关键字，参数与已命名参数匹配。

在 CREATE PROCEDURE 语句中可以为过程参数指派缺省值，为缺少的参数指派缺省值。如果未设置缺省值，也没有提供参数，则会给出错误。

在过程内部，当过程返回结果集时，CALL 语句可以用在 DECLARE 语句中。请参见“从过程返回结果”一节 《SQL Anywhere 服务器 - SQL 的用法》。

使用 RETURN 语句，过程可返回整数值（例如作为状态指示符）。可以使用赋值运算符等号将此返回值保存在变量中：

```
CREATE VARIABLE returnval INT;
returnval = CALL proc_integer ( arg1 = val1, ... )
```

注意
如果所调用的过程返回了 INT 并且值为 NULL，则会返回错误状态值 0。无法区分这种情况和返回实际值为 0 的情况。

权限

必须是过程的所有者，拥有过程的 EXECUTE 权限或者有 DBA 权限。

副作用

无。

另请参见

- “CREATE FUNCTION 语句（Web 服务）”一节第 445 页
- “CREATE PROCEDURE 语句（Web 服务）”一节第 471 页
- “GRANT 语句”一节第 595 页
- “EXECUTE 语句 [T-SQL]”一节第 568 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

调用 ShowCustomers 过程。此过程没有参数，并返回结果集。

```
CALL ShowCustomers();
```

以下 Interactive SQL 示例创建一个可返回客户（已提供其客户 ID）所下订单数的过程，创建一个用于保存结果的变量，调用此过程并显示结果。

```
CREATE PROCEDURE OrderCount (IN customer_ID INT, OUT Orders INT)
BEGIN
   SELECT COUNT(SalesOrders.ID)
   INTO Orders
   FROM Customers
   KEY LEFT OUTER JOIN SalesOrders
   WHERE Customers.ID = customer_ID;
END
go
 -- Create a variable to hold the result
CREATE VARIABLE Orders INT
go
-- Call the procedure, FOR customer 101
CALL OrderCount ( 101, Orders )
go
--  Display the result
SELECT Orders FROM DUMMY
go
```

CASE 语句

此语句用于根据多种情况选择执行路径。

语法 1

CASE *value-expression*
WHEN [*constant* | **NULL**] **THEN** *statement-list* ...
[**WHEN** [*constant* | **NULL**] **THEN** *statement-list*] ...
[**ELSE** *statement-list*]
END [**CASE**]

语法 2

CASE
WHEN [*search-condition* | **NULL**] **THEN** *statement-list* ...
[**WHEN** [*search-condition* | **NULL**] **THEN** *statement-list*] ...
[**ELSE** *statement-list*]
END [**CASE**]

注释

语法 1 CASE 语句是一个控制语句，可用于根据表达式的值选择要执行的 SQL 语句列表。 *value-expression* 是具有单值的表达式，它的值可以是字符串、数字、日期或其它 SQL 数据类型。如果 *value-expression* 的值存在匹配的 WHEN 子句，则执行 WHEN 子句中的 *statement-list*。如果没有合适的 WHEN 子句而有 ELSE 子句，则执行 ELSE 子句中的 *statement-list*。执行过程从 END CASE 之后的第一个语句开始继续。

如果 *value-expression* 可以为空，可使用 ISNULL 函数用不同的表达式替换值为 NULL 的 *value-expression*。

语法 2 在这种语法形式格式下，执行 CASE 语句中第一个满足 *search-condition* 的语句。如果不满足任何 *search-conditions*，则执行 ELSE 子句。

如果表达式可以为 NULL，则对第一个 *search-condition* 使用以下语法：

```
WHEN search-condition IS NULL THEN statement-list
```


CASE 语句不同于 CASE 表达式
不要混淆 CASE 语句与 CASE 表达式的语法。请参见“CASE 表达式”一节第 18 页。

权限

无。

副作用

无。

另请参见

- “ISNULL 函数 [Miscellaneous]”一节第 222 页
- “未知值：NULL”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “BEGIN 语句”一节第 395 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

以下过程使用 CASE 语句，将 SQL Anywhere 示例数据库 Products 表中列出的产品分为衬衣、帽子、短裤或未知几大类。

```
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
   DECLARE prod_name CHAR(20);
   SELECT Name INTO prod_name FROM Products
   WHERE ID = product_ID;
   CASE prod_name
   WHEN 'Tee Shirt' THEN
      SET type = 'Shirt'
   WHEN 'Sweatshirt' THEN
      SET type = 'Shirt'
   WHEN 'Baseball Cap' THEN
      SET type = 'Hat'
   WHEN 'Visor' THEN
      SET type = 'Hat'
   WHEN 'Shorts' THEN
      SET type = 'Shorts'
   ELSE
      SET type = 'UNKNOWN'
   END CASE;
END;
```

以下示例使用语法 2 生成关于 SQL Anywhere 示例数据库中产品数量的消息。

```
CREATE PROCEDURE StockLevel (IN product_ID INT)
BEGIN
   DECLARE qty INT;
   SELECT Quantity INTO qty FROM Products
   WHERE ID = product_ID;
   CASE
   WHEN qty < 30 THEN
      MESSAGE 'Order Stock' TO CLIENT;
   WHEN qty > 100 THEN
```

```
        MESSAGE 'Overstocked' TO CLIENT;
    ELSE
        MESSAGE 'Sufficient stock on hand' TO CLIENT;
    END CASE;
END;
```

CHECKPOINT 语句

此语句用于对数据库执行检查点操作。

语法

CHECKPOINT

注释

CHECKPOINT 语句强制数据库服务器执行检查点操作。数据库服务器也会根据内部算法自动执行检查点操作。应用程序通常不需要发出 CHECKPOINT 语句。

权限

对网络数据库执行检查点操作需要有 DBA 权限。

对个人数据库服务器执行检查点操作不需要任何权限。

副作用

无。

另请参见

- “备份和数据恢复” 《SQL Anywhere 服务器 - 数据库管理》
- “了解检查点日志”一节 《SQL Anywhere 服务器 - 数据库管理》
- “checkpoint_time 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “recovery_time 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

CLEAR 语句 [Interactive SQL]

此语句用于清除 Interactive SQL 窗格中的内容。

语法

CLEAR

注释

CLEAR 语句用于清除 [**SQL 语句**] 和 [**结果**] 窗格。

权限

无。

副作用

关闭与清除的数据关联的游标。

另请参见

- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

CLOSE 语句 [ESQL] [SP]

此语句用于关闭游标。

语法

CLOSE *cursor-name*

cursor-name : *identifier* | *hostvar*

注释

此语句关闭指定的游标。

权限

游标在此前必须已打开。

副作用

无。

另请参见

- “OPEN 语句 [ESQL] [SP]”一节第 647 页
- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “PREPARE 语句 [ESQL]”一节第 657 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

以下示例关闭嵌入式 SQL 中的游标。

```
EXEC SQL CLOSE employee_cursor;
EXEC SQL CLOSE :cursor_var;
```

以下过程使用游标。

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
   DECLARE err_notfound EXCEPTION
      FOR SQLSTATE '02000';
   DECLARE curThisCust CURSOR FOR
   SELECT CompanyName, CAST(    sum(SalesOrderItems.Quantity *
   Products.UnitPrice) AS INTEGER) VALUE
   FROM Customers
   LEFT OUTER JOIN SalesOrders
   LEFT OUTER JOIN SalesOrderItems
   LEFT OUTER JOIN Products
   GROUP BY CompanyName;
DECLARE ThisValue INT;
   DECLARE ThisCompany CHAR(35);
   SET TopValue = 0;
   OPEN curThisCust;
   CustomerLoop:
   LOOP
      FETCH NEXT curThisCust
      INTO ThisCompany, ThisValue;
         IF SQLSTATE = err_notfound THEN
            LEAVE CustomerLoop;
         END IF;
         IF ThisValue > TopValue THEN
            SET TopValue = ThisValue;
            SET TopCompany = ThisCompany;
         END IF;
      END LOOP CustomerLoop;
   CLOSE curThisCust;
END
```

COMMENT 语句

此语句用于在系统表中存储关于数据库对象的注释。

语法

COMMENT ON {
  COLUMN [*owner*.]*table-name*.*column-name*
 | **DBSPACE** *dbspace-name*
 | **EVENT** [*owner*.]*event-name*
 | **EXTERNAL ENVIRONMENT** *environment-name*
 | **EXTERNAL OBJECT** *object-name*
 | **FOREIGN KEY** [*owner*.]*table-name*.*role-name*
 | **INDEX** [[*owner*.] *table*.]*index-name*
 | **INTEGRATED LOGIN** *integrated-login-id*
 | **JAVA CLASS** *java-class-name*
 | **JAVA JAR** *java-jar-name*
 | **KERBEROS LOGIN "***client-Kerberos-principal***"**
 | **LOGIN POLICY** *policy-name*
 | **MATERIALIZED VIEW** [*owner*.]*materialized-view-name*
 | **PRIMARY KEY ON** [*owner*.]*table-name*
 | **PROCEDURE** [*owner*.]*procedure-name*
 | **SERVICE** *web-service-name*
 | **TABLE** [*owner*.]*table-name*
 | **TEXT CONFIGURATION** [*owner*.]*text-config-name*
 | **TEXT INDEX** *text-index-name* **ON** [*owner*.]*table-name*
 | **TRIGGER** [[*owner*.]*tablename*.]*trigger-name*


```
  | USER userid
  | VIEW [ owner.]view-name
}
IS comment

comment : string | NULL

environment-name :
JAVA
| PERL
| PHP
| CLR
| C_ESQL32
| C_ESQL64
| C_ODBC32
| C_ODBC64
```

注释

COMMENT 语句可用于在数据库中为对象设置注释。COMMENT 语句可更新系统表 ISYSREMARKS 中所列出的注释。通过将注释设置成 NULL 可以删除注释。对于索引或触发器的注释，其所有者是定义该索引或触发器的表的所有者。

不能为本地临时表添加注释。

environment-name 为 JAVA、PERL、PHP、CLR、C_ESQL32、C_ESQL64、C_ODBC32 或 C_ODBC64 中之一。

如果使用 [**数据库文档生成器**] 生成 SQL Anywhere 数据库文档，则可以选择在输出中包括对过程、函数、触发器、事件和视图的注释。请参见“记录数据库”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

必须是要注释的数据库对象的所有者，或者必须有 DBA 权限。

副作用

自动提交。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例显示如何添加和删除注释。

向 Employees 表添加注释。

```
COMMENT
ON TABLE Employees
IS 'Employee information';
```

从 Employees 表中删除注释。

```
COMMENT
ON TABLE Employees
IS NULL;
```

要查看对象的注释集，可使用与以下类似的 SELECT 语句。此示例检索 SQL Anywhere 示例数据库中 ViewSalesOrders 视图的注释集。

```
SELECT remarks
FROM SYSTAB t, SYSREMARK r
WHERE t.object_id = r.object_id
AND t.table_name = 'ViewSalesOrders';
```

COMMIT 语句

此语句用于使数据库更改永久生效，或用于终止用户定义的事务。

语法 1

COMMIT [**WORK**]

语法 2

COMMIT TRAN[SACTION] [*transaction-name*]

参数

- **transaction-name** 指派给此事务的可选名称。它必须是有效的标识符。只应在最外层的嵌套 BEGIN/COMMIT 语句对或 BEGIN/ROLLBACK 语句对中使用事务名。

  有关 Adaptive Server Enterprise 和 SQL Anywhere 中嵌套事务的详细信息，请参见“BEGIN TRANSACTION 语句 [T-SQL]”一节第 398 页。有关保存点的详细信息，请参见“SAVEPOINT 语句”一节第 688 页。

  可以使用一组选项来控制 COMMIT 语句的行为细节。请参见：

  - “cooperative_commit_timeout 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “cooperative_commits 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “delayed_commits 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “delayed_commit_timeout 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

  可以使用 Commit 连接属性返回当前连接的提交数。请参见“连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》。

注释

语法 1 COMMIT 语句结束事务，并使在该事务期间所做的所有更改在数据库中永久生效。

数据定义语句全都自动执行提交。有关信息，请参见每个 SQL 语句的 [副作用] 列表。

如果数据库服务器检测到任何无效的外键，则 COMMIT 语句将失败。这产生的结果是：不可能通过任何无效外键结束事务。通常，在每个数据处理操作中会检查外键的完整性。但是，如果数据库选项 wait_for_commit 设置为 On，或者使用 CHECK ON COMMIT 选项定义了具体的外键，则数据库服务器会将完整性检查推迟到执行 COMMIT 语句之后。

语法 2 可以使用 BEGIN TRANSACTION 和 COMMIT TRANSACTION 语句对来构造嵌套事务。嵌套事务类似于保存点。当在一组嵌套事务的最外层执行时，此语句使数据库更改永久生效。当在事务内部执行时，COMMIT TRANSACTION 语句使事务嵌套级别减少一级。当事务被嵌套时，只有最外层的 COMMIT 可使数据库更改永久生效。

语法 2 是 Transact-SQL 扩展。

权限

无。

副作用

除了使用 WITH HOLD 打开的游标之外，关闭所有游标。

删除此连接上声明的临时表的所有行，除非这些表是用 ON COMMIT PRESERVE ROWS 声明的。

另请参见

- “SAVEPOINT 语句”一节第 688 页
- “BEGIN TRANSACTION 语句 [T-SQL]”一节第 398 页
- “PREPARE TO COMMIT 语句”一节第 658 页
- “ROLLBACK 语句”一节第 684 页

标准和兼容性

- **SQL/2003** 语法 1 是核心特性。语法 2 是 Transact-SQL 扩充。

示例

下面的语句提交当前事务：

```
COMMIT;
```

以下 Transact-SQL 批处理语句报告 @@trancount 的连续值：0、1、2、1、0。

```
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
BEGIN TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
COMMIT TRANSACTION
PRINT @@trancount
go
```

CONFIGURE 语句 [Interactive SQL]

此语句用于打开 Interactive SQL 的 [选项] 窗口。

语法

CONFIGURE

注释

CONFIGURE 语句用于打开 Interactive SQL [选项] 窗口。此窗口显示所有 Interactive SQL 选项的当前设置。它不显示数据库选项，也不允许对它们进行修改。在此窗口中可以配置 Interactive SQL 的设置。

权限

无。

副作用

无。

另请参见

- “SET OPTION 语句”一节第 702 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

CONNECT 语句 [ESQL] [Interactive SQL]

此语句用于建立到数据库的连接。

语法 1

CONNECT
[**TO** *database-server-name*]
[**DATABASE** *database-name*]
[**AS** *connection-name*]
[**USER**] *userid* [**IDENTIFIED BY** *password*]

database-server-name, *database-name*, *connection-name*, *userid*, *password* :
{ *identifier* | *string* | *hostvar* }

语法 2

CONNECT USING *connect-string*

connect-string : { *identifier* | *string* | *hostvar* }

参数

- **AS 子句** 通过指定 AS 子句可对连接进行命名（可选）。这允许建立到同一个数据库的多个连接，或者到同一个或不同的多个数据库服务器的多个连接，所有连接都是同时发生的。每个连接都有自己的关联事务。事务之间有时甚至会发生锁定冲突。例如，当试图从两个不同的连接修改同一数据库中的同一记录时，就会出现这种情况。

语法 2 *connect-string* 是格式为 keyword=*value* 的参数设置列表，必须用单引号将它括起来，并用分号隔开。

有关连接字符串的详细信息，请参见“连接参数”一节 《SQL Anywhere 服务器 - 数据库管理》。

注释

CONNECT 语句建立与数据库之间的连接，该数据库运行在由 *database-server-name* 标识的数据库服务器上，由 *database-name* 来标识。过程、触发器、事件或批处理不支持此语句。

嵌入式 SQL 行为 在嵌入式 SQL 中，如果未指定 *database-server-name*，则会采用缺省的本地数据库服务器（启动的第一个数据库服务器）。如果未指定 *database-name*，则会采用给定服务器上的第一个数据库。

WHENEVER 语句、SET SQLCA 和一些 DECLARE 语句不生成代码，因此它们在源文件中可以出现在 CONNECT 语句之前。否则，在成功执行一条 CONNECT 语句之前，不允许执行任何语句。

用户 ID 和口令用于对所有动态 SQL 语句执行权限检查。

有关连接算法的详细说明，请参见“对连接进行故障排除”一节 《SQL Anywhere 服务器 - 数据库管理》。

注意
对于 SQL Anywhere，只有语法 1 在嵌入式 SQL 中有效。对于 UltraLite，语法 1 和语法 2 都可以用于嵌入式 SQL。

Interactive SQL 行为 如果在 CONNECT 语句中未指定任何数据库或服务器，Interactive SQL 将保持与当前数据库（而不是与缺省的服务器和数据库）的连接。如果指定了数据库名但未指定服务器名，Interactive SQL 会尝试连接到当前服务器上的指定数据库。如果指定了服务器名但未指定数据库名，Interactive SQL 会连接到指定服务器上的缺省数据库。

例如，如果在与数据库连接时执行以下批处理语句，则将在同一个数据库中创建两个表。

```
CREATE TABLE t1( c1 int );
CONNECT DBA IDENTIFIED BY sql;
CREATE TABLE t2 (c1 int );
```

在成功执行一条 CONNECT 语句之前，不允许执行其它数据库语句。

当 Interactive SQL 在窗口模式下运行时，会提示您提供缺少的连接参数。

当 Interactive SQL 在命令提示符模式（从命令行启动 Interactive SQL 时指定了 -nogui）或批处理模式下运行时，或者未使用 AS 子句执行 CONNECT 时，将打开一个未命名的连接。如果已有其它未命名的连接打开，则以前的那个连接将自动关闭。否则，在运行 CONNECT 时不关闭现有连接。

对多个连接的管理是通过当前连接的概念来实现的。在成功执行连接语句后，新连接变成当前连接。要切换到不同的连接，请使用 SET CONNECTION 语句。DISCONNECT 语句用于删除连接。

当与 Interactive SQL 连接时，指定 CONNECT [USER] *userid* 与执行 SETUSER WITH OPTION *userid* 语句相同。请参见“SETUSER 语句”一节第 708 页。

在 Interactive SQL 中，连接信息（包括数据库名、用户 ID 和数据库服务器）出现在 [SQL 语句] 窗格上方的标题栏中。如果未连接到数据库，标题栏中将显示 [未连接]。

注意
除了 Interactive SQL 不支持 *hostvar* 参数以外，语法 1 和语法 2 在 Interactive SQL 中都有效。

权限

无。

副作用

无。

另请参见

- “GRANT 语句”一节第 595 页
- “DISCONNECT 语句 [ESQL] [Interactive SQL]”一节第 540 页
- “SET CONNECTION 语句 [Interactive SQL] [ESQL]”一节第 700 页
- “SETUSER 语句”一节第 708 页
- “连接参数”一节 《SQL Anywhere 服务器 - 数据库管理》
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 语法 1 是核心 SQL 外部的 SQL/基础特性。语法 2 是服务商扩充。

示例

以下是 CONNECT 在嵌入式 SQL 中的用法示例。

```
EXEC SQL CONNECT AS :conn_name
USER :userid IDENTIFIED BY :password;
EXEC SQL CONNECT USER "DBA" IDENTIFIED BY "sql";
```

以下示例假定 SQL Anywhere 示例数据库已经启动。

从 Interactive SQL 连接到数据库。Interactive SQL 会提示您输入用户 ID 和口令。

```
CONNECT;
```

以 DBA 身份从 Interactive SQL 连接到缺省数据库。Interactive SQL 提示您输入口令。

```
CONNECT USER "DBA";
```

以用户 DBA 的身份从 Interactive SQL 连接到示例数据库。

```
CONNECT
TO demo11
USER DBA
IDENTIFIED BY sql;
```

使用连接字符串从 Interactive SQL 连接到示例数据库。

```
CONNECT
USING 'UID=DBA;PWD=sql;DBN=demo';
```

连接到示例数据库后，数据库名、用户 ID 和服务器名会出现在标题栏中： **demo (DBA) on demo11**.

CONTINUE 语句 [T-SQL]

此语句用于重新启动循环。

语法

CONTINUE [*statement-label*]

注释

CONTINUE 语句是一个控制语句，可用于重新启动循环。在循环中的第一个语句继续执行。当 CONTINUE 出现在使用 Watcom-SQL 的一组语句中时，必须指定 *statement-label*。

当 CONTINUE 出现在使用 Transact-SQL 的一组语句中时，不得使用 *statement-label*。

权限

无。

副作用

无。

另请参见

- “LOOP 语句”一节第 637 页
- “WHILE 语句 [T-SQL]”一节第 748 页
- “FOR 语句”一节第 577 页
- “BEGIN 语句”一节第 395 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

以下代码段显示如何使用 CONTINUE 语句重新启动循环。此示例显示 1 到 10 之间的奇数。

```
BEGIN
   DECLARE i INT;
   SET i = 0;
   lbl:
   WHILE i < 10 LOOP
      SET i = i + 1;
      IF mod( i, 2 ) = 0 THEN
         CONTINUE lbl
      END IF;
      MESSAGE 'The value ' || i || ' is odd.' TO CLIENT;
   END LOOP lbl;
END
```

CREATE DATABASE 语句

此语句用于创建数据库。数据库存储为操作系统文件。

语法

CREATE DATABASE *db-filename-string* [*create-option ...*]

create-option :
  [**ACCENT** { **RESPECT** | **IGNORE** | **FRENCH** }]
  [**ASE** [**COMPATIBLE**]]
  [**BLANK PADDING** { **ON** | **OFF** }]
  [**CASE** { **RESPECT** | **IGNORE** }]
  [**CHECKSUM** { **ON** | **OFF** }]
  [**COLLATION** *collation-label*[**(** *collation-tailoring-string* **)**]]
  [**DATABASE SIZE** *size* { **KB** | **MB** | **GB** | **PAGES** | **BYTES** }]
  [**DBA USER** *userid*]
  [**DBA PASSWORD** *password*]
  [**ENCODING** *encoding-label*]
  [**ENCRYPTED** [**TABLE**] { *algorithm-key-spec* | **OFF** }]
  [**JCONNECT** { **ON** | **OFF** }]
  [**PAGE SIZE** *page-size*]
  [**NCHAR COLLATION** *nchar-collation-label*[**(** *collation-tailoring-string* **)**]]
  [[**TRANSACTION**] { **LOG OFF** | **LOG ON** [*log-filename-string*]
    [**MIRROR** *mirror-filename-string*] }]

page-size :
2048 | **4096** | **8192** | **16384** | **32768**

algorithm-key-spec:
ON
| [**ON**] **KEY** *key* [**ALGORITHM** *AES-algorithm*]
| [**ON**] **ALGORITHM** *AES-algorithm* **KEY** *key*
| [**ON**] **ALGORITHM 'SIMPLE'**

AES-algorithm :
'AES' | **'AES256'** | **'AES_FIPS'** | **'AES256_FIPS'**

key : quoted string

参数

文件名（*db-filename-string*、*log-filename-string* 和 *mirror-filename-string*）是包含操作系统文件名的字符串。作为字符串，它们必须用单引号括起来。

- 如果指定路径，则任何后面跟有 n 或 x 的反斜杠字符 (\) 都必须使用两个。按照 SQL 中的字符串规则，将它们转义可阻止将其解释为换行字符 (\n) 或十六进制数 (\x)。

  对反斜线字符进行转义始终是较为安全的方法。例如：

  ```
  CREATE DATABASE 'c:\\databases\\my_db.db'
  LOG ON 'e:\\logdrive\\my_db.log';
  ```

- 如果指定相对路径或者未指定路径，则将在相对于数据库服务器工作目录的位置上创建数据库文件。如果没有指定事务日志文件的路径，则该文件在数据库文件所在的同一目录中创建。建议您在计算机的不同磁盘上分别存储数据库文件和事务日志。
- 如果未提供文件扩展名，则创建的文件将带有不同的扩展名：对于数据库为 *.db*，对于事务日志为 *.log*，对于事务日志镜像为 *.mlg*。

不可以将 utility_db 指定为 *db-filename-string*。该名称已保留供实用程序数据库使用。请参见“使用实用程序数据库”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **ACCENT 子句** 此子句用于为数据库指定区分重音。不再支持此子句。使用为 COLLATION 和 NCHAR COLLATION 子句提供的归类定制选项指定区分重音。

  仅当将 UCA（Unicode 归类算法）用于 COLLATION 或 NCHAR COLLATION 子句中指定的归类时，ACCENT 子句才会适用。ACCENT RESPECT 会导致在进行 UCA 字符串比较时考虑字母间重音的差异。例如，e 小于 é。ACCENT FRENCH 类似于 ACCENT RESPECT，只是按照法语规则从右向左来比较重音。ACCENT IGNORE 会导致在进行字符串比较时忽略重音。例如，e 等于 é。

  如果创建数据库时未指定区分重音，用于比较和排序的重音区分缺省设置为*不区分*，但有一点例外；对于用 UCA 归类创建的日文数据库，重音区分的缺省设置为*区分*。

  有关字符集的详细信息，请参见“国际语言和字符集” 《SQL Anywhere 服务器 - 数据库管理》。

- **ASE COMPATIBLE 子句** 不会创建 SYS.SYSCOLUMNS 和 SYS.SYSINDEXES 视图。缺省情况下，创建这些视图是为了与 Watcom SQL（此软件的版本 4 及更早版本）中可用的系统表兼容。这些视图与 Sybase Adaptive Server Enterprise 兼容性视图 dbo.syscolumns 和 dbo.sysindexes 冲突。

- **BLANK PADDING 子句** SQL Anywhere 在比较所有字符串时都会将它们视为具有可变的长度并存储在 VARCHAR 域。这其中包括与固定长度 CHAR 或 NCHAR 列有关的字符串比较。此外，当数值存储于数据库中时，SQL Anywhere 不会截断或添加尾随空白。

  缺省情况下，SQL Anywhere 将空白视为有效字符。例如，值 'a '（字符 'a' 后跟一个空格）与单个字符串 'a' 不相等。不等比较还会将空白与归类中的任何其它字符同等看待。

  如果启用空格填充（指定 BLANK PADDING ON），则字符串比较的语义更符合 ANSI/ISO SQL 标准。启用空格填充后，SQL Anywhere 会在所有比较中忽略尾随空白。

  在上例中，在采用空格填充的数据库中 'a ' 与 'a' 的相等比较会返回 TRUE。对于采用空格填充的数据库，固定长度的字符串值在由应用程序读取时会以空白进行填充。应用程序针对此类赋值是否会收到字符串截断警告是通过 ansi_blanks 连接选项来控制的。请参见“ansi_blanks 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **CASE 子句** 此子句用于为数据库指定是否区分大小写。不再支持此子句。使用为 COLLATION 和 NCHAR COLLATION 子句提供的归类定制选项指定区分大小写。

  CASE RESPECT 会导致在对所有 CHAR 和 NCHAR 数据类型进行字符串比较时区分大小写。使用 UCA 进行比较时，只有在基本字母和重音均相等的情况下，才会考虑字母的大小写。对于所有其它归类，大写字母和小写字母是不同的，例如 a 小于 A，而 A 小于 b，等等。CASE IGNORE 会导致在进行字符串比较时不区分大小写。将大写字母和小写字母视为完全相等。

  如果创建数据库时未指定区分大小写，用于比较和排序的大小写区分缺省设置为*不区分*，但有一点例外；对于用 UCA 归类创建的日文数据库，大小写区分的缺省设置为*区分*。

  提供 CASE RESPECT 是为了符合 ISO/ANSI SQL 标准。数据库中的标识符始终不区分大小写，即使在区分大小写的数据库中也是如此。

  有关字符集的详细信息，请参见“国际语言和字符集” 《SQL Anywhere 服务器 - 数据库管理》。

- **CHECKSUM 子句** 校验和用于确定是否在磁盘上已修改了数据库页。在您创建启用校验和的数据库时，在将每一页写入磁盘前为其计算校验和。下次从磁盘中读取该页时，就会重新计算该页的校验和，并将其与该页上存储的校验和进行比较。如果校验和不同，说明页面在磁盘中已被修改，并发生错误。在启用了校验和的情况下创建的数据库，也可使用校验和进行校验。可通过执行以下语句来检查创建数据库时是否启用了校验和：

  ```
  SELECT DB_PROPERTY ( 'Checksum' );
  ```

  如果开启了校验和，则此查询将返回 ON，否则将返回 OFF。缺省情况下，校验和是处于关闭状态的，因此如果省略了 CHECKSUM 子句，则会应用 OFF。

  不管此子句的设置如何，数据库服务器总是为在存储设备（例如可移动驱动器）和 Windows Mobile 上运行的数据库启用校验和，以帮助在数据库文件损坏时尽早地将其检测出来。它也在校验活动进行期间为某些关键页计算校验和。请参见“使用校验和检测损坏”一节 《SQL Anywhere 服务器 - 数据库管理》、“校验实用程序 (dbvalid)”一节 《SQL Anywhere 服务器 - 数据库管理》、“sa_validate 系统过程”一节第 912 页或“VALIDATE 语句”一节第 744 页。

- **COLLATION 子句** 由 COLLATION 子句指定的归类用于字符数据类型（CHAR、VARCHAR 和 LONG VARCHAR）的排序和比较。归类为所使用的编码（字符集）提供字符比较和排序信息。如果不指定 COLLATION 子句，SQL Anywhere 会根据操作系统语言和编码选择一种归类。

  可以从使用 SQL Anywhere 归类算法（或 Unicode 归类算法 (UCA)）的归类列表中选择归类。如果已指定 UCA，则您还应指定 ENCODING 子句。

  谨慎选择归类非常重要。数据库创建完毕之后，将无法再更改归类。请参见“选择归类”一节 《SQL Anywhere 服务器 - 数据库管理》。

  有关支持的归类列表，请参见“建议的字符集和归类”一节 《SQL Anywhere 服务器 - 数据库管理》和“支持的归类和替代归类”一节 《SQL Anywhere 服务器 - 数据库管理》。

  您还可以指定归类定制选项 (*collation-tailoring-string*)，以便能够对字符的排序和比较进行更多的控制。这些选项采用 "关键字=值" 对的形式，包括在圆括号内，位于归类名后。例如，`... CHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)'`。

- **DATABASE SIZE 子句** 预先为数据库分配空间有助于降低数据库所在驱动器空间不足的风险。而且，它还可以在数据库服务器有数据库扩充需要之前增加数据库的可存储数据量，从而有助于提高各项性能。扩充数据库是一项非常耗时的操作。可以使用 KB、MB、GB 或 PAGES 将单位分别指定为千字节、兆字节、千兆字节或页。

- **DBA USER 子句** 此子句用于为数据库指定 DBA 用户。使用此子句时，可以不再以缺省 DBA 用户的身份连接到数据库。如果不指定此子句，则会创建缺省的 DBA 用户 ID。

- **DBA PASSWORD 子句** 可以为 DBA 数据库用户指定一个不同的口令。如果不指定此子句，则 DBA 用户将使用缺省口令 (**sql**)。

- **ENCODING 子句** 在 COLLATION 子句中指定的多数归类会规定编码（字符集）和排序。对于这些归类，不应指定 ENCODING 子句。但是，如果 COLLATION 子句中指定的值为 UCA（Unicode 归类算法），则使用 ENCODING 子句指定特定于区域的编码，并利用 UCA 进行比较和排序。ENCODING 子句可能为 CHARA 数据类型指定 UTF-8 或任何单字节编码。ENCODING 可能不会指定 UTF-8 以外的多字节编码。

  如果选择 UCA 归类，则可以有选择地指定归类定制选项。请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

如果将 COLLATION 设置为 UCA 但未指定 ENCODING，SQL Anywhere 将使用 UTF-8。有关建议的编码和归类的详细信息，请参见“建议的字符集和归类”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关如何获得 SQL Anywhere 支持的编码列表的详细信息，请参见“支持的字符集”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **ENCRYPTED 或 ENCRYPTED TABLE 子句** 加密会使存储数据变为不可读。当您想要加密整个数据库时，请使用 ENCRYPTED 关键字（无 TABLE）。当您只想启用表加密时，请使用 ENCRYPTED TABLE 子句。启用表加密将意味着，会使用您在创建数据库时指定的设置，对随后使用 ENCRYPTED 子句创建或变更的表进行加密。请参见“表加密”一节 《SQL Anywhere 服务器 - 数据库管理》。

  有两个级别的数据库和表加密：简单加密和高度加密。简单加密等效于模糊处理。虽然这些数据无法阅读，但了解加密技术的人可能会破解数据。高度加密使数据无法阅读，而且基本上无法被破解。

  对于简单加密，请指定 ENCRYPTED ON ALGORITHM SIMPLE 或 ENCRYPTED ALGORITHM SIMPLE，或者指定 ENCRYPTED ON 子句而不指定算法或密钥。

  对于高度加密，请指定使用 128 位或 256 位 AES 算法的 ENCRYPTED ON ALGORITHM，并指定 KEY 子句来指定加密密钥。建议选择满足以下条件的密钥值：长度至少为 16 个字符，混合使用大小写并包含数字、字母和特殊字符。

  在 Windows Mobile 上，仅 ARM 处理器支持 AES_FIPS 和 AES256_FIPS 算法。

  > **小心**
  > 对于高度加密的数据库，请务必将密钥的副本保存在安全的位置。如果丢失了加密密钥，就没有办法访问数据，即使有技术支持人员协助也不行。此时必须放弃该数据库并创建一个新的数据库。

  有关高度数据库加密的详细信息，请参见“高度加密”一节 《SQL Anywhere 服务器 - 数据库管理》。

  还可以使用 CREATE ENCRYPTED DATABASE 语句为现有数据库创建一个加密副本。请参见“CREATE ENCRYPTED DATABASE 语句”一节第 425 页。

- **JCONNECT 子句** 要允许 jConnect JDBC 驱动程序访问系统目录信息，请指定 JCONNECT ON。这将会安装提供 jConnect 支持的系统对象。如果要排除 jConnect 系统对象，请指定 JCONNECT OFF。只要不访问系统信息，您就仍然可以使用 JDBC。缺省情况下，JCONNECT 为 ON。

  如果要创建在 Windows Mobile 上使用的数据库，请参见“在 Windows Mobile 上使用 jConnect”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **NCHAR COLLATION 子句** 由 NCHAR COLLATION 子句指定的归类用于国家字符数据类型（NCHAR、NVARCHAR 和 LONG NVARCHAR）的排序和比较。归类为用于国家字符的 UTF-8 编码（字符集）提供字符排序信息。如果不指定 NCHAR COLLATION 子句，则 SQL Anywhere 将使用 Unicode 归类算法 (UCA)。唯一允许使用的其它归类为 UTF8BIN，它提供编码大于 0x7E 的所有字符的二进制排序。请参见“选择归类”一节 《SQL Anywhere 服务器 - 数据库管理》。

您还可以指定归类定制选项 (*collation-tailoring-string*)，以便能够对字符的排序和比较进行更多的控制。这些选项采用 "关键字=值" 对的形式，包括在带引号的字符串中，位于归类名后。例如，`... NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)'`。如果指定 ACCENT 或 CASE 子句以及包含大小写和重音设置的归类定制字符串，ACCENT 和 CASE 子句的值将只作为缺省值使用。

有关支持的归类定制选项的列表，请参见“归类定制选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

注意
如果指定 UCA 归类，将支持所有归类定制选项。对于所有其它归类，仅支持区分大小写定制选项。

注意
使用归类定制选项创建的数据库，无法使用 10.0.1 之前版本的数据库服务器启动。

- **PAGE SIZE 子句** 数据库的页面大小可以是 2048、4096、8192、16384 或 32768 个字节。缺省的页面大小是 4096 个字节。大型数据库通常可从较大的页面大小中获益，但也将负担与较大的页面大小相关的额外开销。

  例如，

  ```
  CREATE DATABASE 'c:\\databases\\my_db.db'
  PAGE SIZE 4096;
  ```

  页面大小限制
  页面大小不能超出当前服务器所使用的页面大小。服务器页面大小可从启动的第一个数据库集中获取，也可在服务器命令行上使用 -gp 选项来设置。请参见“-gp 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **TRANSACTION LOG 子句** 事务日志是数据库服务器用来记录对数据库进行的所有更改的文件。在备份和恢复以及数据复制中，事务日志起着重要的作用。

  如果要使用事务日志镜像，可利用 TRANSACTION 子句的 MIRROR 子句提供文件名。事务日志镜像是与事务日志完全相同的一个副本，通常在单独的设备上进行维护，以便更好地保护您的数据。缺省情况下，SQL Anywhere 不使用事务日志镜像。

注释

创建具有所提供名称和属性的数据库文件。此语句在过程、触发器、事件或批处理中不受支持。

权限

执行此语句所需的权限是在服务器命令行上用 -gu 选项来设置的。缺省设置为要求具有 DBA 权限。

运行数据库服务器时所使用的帐户必须对创建文件的目录具有写权限。

副作用

创建操作系统文件。

另请参见

- “ALTER DATABASE 语句”一节第 344 页
- “DROP DATABASE 语句”一节第 542 页
- “初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》
- “DatabaseKey 连接参数 [DBKEY]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “事务日志”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句在 *C:* 目录中创建名为 *mydb.db* 的数据库文件。

```
CREATE DATABASE 'C:\\mydb.db'
TRANSACTION LOG ON
CASE IGNORE
PAGE SIZE 4096
ENCRYPTED OFF
BLANK PADDING OFF;
```

以下语句使用代码页 1252 创建数据库，并将 UCA 用于 CHAR 和 NCHAR 数据类型。在比较和排序期间考虑重音和大小写。

```
CREATE DATABASE 'c:\\uca.db'
COLLATION 'UCA'
ENCODING 'CP1252'
NCHAR COLLATION 'UCA'
ACCENT RESPECT
CASE RESPECT;
```

以下语句创建数据库 *myencrypteddb.db*，该数据库使用简单加密进行加密：

```
CREATE DATABASE 'myencrypteddb.db'
ENCRYPTED ON;
```

以下语句创建数据库 *mystrongencryptdb.db*，该数据库使用密钥 gh67AB2 进行加密（高度加密）：

```
CREATE DATABASE 'mystrongencryptdb.db'
ENCRYPTED ON KEY 'gh67AB2';
```

以下语句使用简单加密创建数据库 *mytableencryptdb.db*，且该数据库启用了表加密。请注意，在 ENCRYPTED 之后插入的关键字 TABLE 指示使用表加密而非数据库加密：

```
CREATE DATABASE 'mytableencryptdb.db'
ENCRYPTED TABLE ON;
```

以下语句使用密钥 gh67AB2（高度加密）和 AES_FIPS 加密算法创建数据库 *mystrongencrypttabledb.db*，且该数据库启用了表加密。

```
CREATE DATABASE 'mystrongencrypttabledb.db'
ENCRYPTED TABLE ON KEY 'gh67AB2'
ALGORITHM 'AES_FIPS';
```

以下语句创建使用归类 1252LATIN1 的名为 *mydb.db* 的数据库文件。NCHAR 归类设置为 UCA，locale 设置为 es，并且启用区分大小写和区分重音设置：

```
CREATE DATABASE 'my2.db'
      COLLATION '1252LATIN1(case=respect)'
      NCHAR COLLATION 'UCA(locale=es;case=respect;accent=respect)';
```

CREATE DBSPACE 语句

此语句用于定义新的数据库空间并创建关联的数据库文件。

语法

CREATE DBSPACE *dbspace-name* **AS** *filename*

参数

- **dbspace-name** 指定 dbspace 的名称。这不是您使用 *filename* 指定的实际数据库文件名。*dbspace-name* 是可以引用的内部名，例如在语句或过程中引用。不能将以下名称用于 dbspace，因为它们为预定义 dbspace 保留的：system、temporary、temp、translog 和 translogmirror。请参见“预定义 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》。

  如果指定的值含有句点 (.) 则返回错误。

- **filename** 可指定数据库文件的名称，还可选择包括文件的路径。如果未指定路径，则在主数据库文件所在的同一位置（目录）中创建数据库文件。如果指定了不同位置，则该路径是相对于数据库服务器的路径。反斜线 (\) 是 SQL 字符串中的转义字符，因此每次必须使用两个反斜线。有关转义字符和字符串的详细信息，请参见“字符串”一节第 9 页。

  filename 参数必须是字符串文字或变量。

注释

CREATE DBSPACE 语句创建新的数据库文件。创建数据库时，它仅包含一个文件。所有创建的表和索引都放在这个文件中。CREATE DBSPACE 在数据库中添加新文件。此文件可以位于与主文件不同的磁盘驱动器上，这意味着数据库的大小可以超过一个物理设备。

对于每个数据库，除了主文件外，最多还可以有 12 个 dbspace。

每个对象（如表或索引）都完全包含在一个 dbspace 之中。CREATE 语句的 IN 子句指定将对象放在哪一个 dbspace 中。缺省情况下，对象放在系统数据库文件中。通过在创建表之前设置 default_dbspace 选项，还可指定将表创建到哪个 dbspace 表中。

有关如何确定数据库对象的缺省 dbspace 的详细信息，请参见“使用附加 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

必须具有 DBA 权限。

副作用

自动提交。自动执行检查点操作。

另请参见

- “default_dbspace 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “DROP DBSPACE 语句”一节第 543 页
- “使用附加 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下 CREATE DBSPACE 语句创建名为 libbooks 的 dbspace。dbspace 的数据库文件名为 *library.db*，它位于 *c:* 目录下。随后的 CREATE TABLE 语句在 libbooks dbspace 中创建了名为 LibraryBooks 的表。

```
CREATE DBSPACE libbooks
AS 'c:\\library.db';
CREATE TABLE LibraryBooks (
  title char(100),
  author char(50),
  isbn char(30),
) IN libbooks;
```

CREATE DECRYPTED DATABASE 语句

此语句会为现有数据库创建一份解密副本，包括所有的事务日志和 dbspace。

语法

CREATE DECRYPTED DATABASE *newfile*
FROM *oldfile*
[**KEY** *key*]

参数

- **FROM 子句** 使用此子句指定要复制的数据库的名称 (*oldfile*)。
- **KEY 子句** 使用此子句指定解密数据库所需的加密密钥。如果现有数据库是使用 SIMPLE 加密进行加密的，则不必指定 KEY 子句，因为这种情况不需要密钥。

注释

CREATE DECRYPTED DATABASE 语句生成一个新的数据库文件 (*newfile*)，且不替换或删除原始数据库文件 (*oldfile*)。

oldfile 中所有已加密的表在 *newfile* 中未被加密，且没有启用表加密。

> **注意**
> 对于使用 SQL Anywhere 11.0.0 及更高版本创建的数据库，ISYSCOLSTAT、ISYSUSER 和 ISYSEXTERNLOGIN 系统表将始终保持加密状态以保护数据，防止对数据库文件进行未经授权的访问。

如果 *oldfile* 使用事务日志或事务日志镜像文件，它们将分别被重命名为 *newfile.log* 和 *newfile.mlg*。

如果 *oldfile* 包含 dbspace 文件，文件名中将添加一个 D（已解密）。例如，在执行 CREATE DECRYPTED DATABASE 语句时，如果 *oldfile* 为 *mydbspace.dbs*，*newfile* 将变为 *mydbspace.dbsD*。

不能对需要恢复的数据库执行此语句。另外，过程、触发器、事件或批处理中均不支持此语句。

权限

必须是具有 DBA 权限的用户。

副作用

无。

另请参见

- “加密和解密数据库”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CREATE ENCRYPTED DATABASE 语句”一节第 425 页
- “CREATE DECRYPTED FILE 语句”一节第 422 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面第一个语句将为 *demo.db* 创建一个 AES256 加密的副本，名为 *demoEncrypted.db*。第二个语句为 *demoEncrypted.db* 创建一个解密副本，名为 *demoDecrypted.db*。

```
CREATE ENCRYPTED DATABASE 'demoEncrypted.db'
   FROM 'demo.db'
   KEY 'Sd8f6654*Mnn'
   ALGORITHM 'AES256';
CREATE DECRYPTED DATABASE 'demoDecrypted.db'
   FROM 'demoEncrypted.db'
   KEY 'Sd8f6654*Mnn';
```

CREATE DECRYPTED FILE 语句

此语句用于为高度加密的数据库创建一个解密副本。还可以使用此语句为与数据库相关的其它文件（例如事务日志、事务日志镜像以及 dbspace）创建解密副本。

语法

CREATE DECRYPTED FILE *newfile*
FROM *oldfile* **KEY** *key*

参数

- **FROM 子句** 列出已加密文件的文件名。
- **KEY 子句** 列出访问加密文件所需的密钥。

注释

当您的数据库需要恢复以及出于支持的原因需要为数据库创建解密副本时，请使用此语句。还必须要使用此语句对任何与数据库相关的文件（例如事务日志、事务日志镜像或 dbspace 文件）进行解密。

原始数据库文件必须用加密密钥高度加密。得到的文件是加密文件的相同副本，但并未加密，因此不需要加密密钥。

如果使用此语句对数据库进行解密，则相应的事务日志文件（以及任何 dbspace）也必须被解密才能使用数据库。

如果需要恢复的数据库是解密的，则其事务日志必须也被解密，并且仍需要恢复新数据库。在上述过程中，事务日志文件的名称保持不变。因此，如果数据库和事务日志文件被重命名，则需要对得到的数据库运行 dblog -t 命令。

无法在启用表加密的数据库上使用此语句。如果具有要解密的表，则使用 ALTER TABLE 语句的 NOT ENCRYPTED 子句来解密这些表。请参见“ALTER TABLE 语句”一节第 373 页。

注意
对于使用 SQL Anywhere 11.0.0 及更高版本创建的数据库，ISYSCOLSTAT、ISYSUSER 和 ISYSEXTERNLOGIN 系统表将始终保持加密状态以保护数据，防止对数据库文件进行未经授权的访问。

此语句在过程、触发器、事件或批处理中不受支持。

权限

必须是具有 DBA 权限的用户。

副作用

无。

另请参见

- “CREATE ENCRYPTED FILE 语句”一节第 427 页
- “CREATE DECRYPTED DATABASE 语句”一节第 421 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例对 contacts 数据库进行解密，并新建一个名为 contacts2 的解密数据库。

```
CREATE DECRYPTED FILE 'contacts2.db'
FROM 'contacts.db'
KEY 'Sd8f6654*Mnn';
```

CREATE DOMAIN 语句

此语句用于在数据库中创建域。

语法

CREATE { **DOMAIN** | **DATATYPE** } [**AS**] *domain-name data-type*
[[**NOT**] **NULL**]
[**DEFAULT** *default-value*]
[**CHECK (** *condition* **)**]

domain-name : *identifier*

data-type : *built-in data type*, *with precision and scale*

参数

- **DOMAIN | DATATYPE 子句** 建议使用 CREATE DOMAIN 而不是 CREATE DATATYPE，因为 CREATE DOMAIN 是 ANSI/ISO SQL3 术语。
- **NULL 子句** 此子句可用于指定域的为空性。当使用域定义列时，将按如下方式确定为空性：
  - 在列定义中指定为空性
  - 在域定义中指定为空性
  - 如果未在列定义或域定义中显式指定为空性，则使用 allow_nulls_by_default 选项的设置。
- **CHECK 子句** 创建 CHECK 条件时，可以在条件中使用一个以 @ 符号为前缀的变量名。如果使用该数据类型定义列，这样的变量将被列名称替换。这样，CHECK 条件便可在数据类型上定义并由任何名称的列使用。

注释

域是内置数据类型的别名，在适用的时候还会包括精度值和小数位数值。它们提高了方便性并促进了数据库的一致性。

域是数据库中的对象。其名称必须遵守标识符规则。与内置数据类型名一样，域名始终不区分大小写。

创建数据类型的用户会自动成为此数据类型的所有者。在 CREATE DATATYPE 语句中不能指定所有者。域名必须唯一，且所有用户都可以访问此数据类型，而不必使用所有者作为前缀。

域可以有 CHECK 条件和 DEFAULT 值，您可以指示数据类型是否允许使用 NULL 值。在数据类型上创建的任何列都将继承这些条件和值。在列上显式指定的任何条件或值会替换为数据类型指定的条件或值。

要从数据库中删除数据类型，请使用 DROP 语句。您必须是数据类型的所有者或者具有 DBA 权限才能删除域。

权限

必须具有 RESOURCE 权限。

副作用

自动提交。

另请参见

- “DROP DOMAIN 语句”一节第 544 页
- “SQL 数据类型”第 75 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL/基础特性。

示例

以下语句创建名为 address 的数据类型，它包含长度为 35 个字符的字符串并且可以为 NULL。

```
CREATE DOMAIN address CHAR( 35 ) NULL;
```

以下语句创建名为 ID 的数据类型，它不允许使用 NULL，且在缺省情况下自动递增。

```
CREATE DOMAIN ID INT
NOT NULL
DEFAULT AUTOINCREMENT;
```

CREATE ENCRYPTED DATABASE 语句

为现有数据库创建一份加密副本，其中包括所有的事务日志和 dbspace。还可以使用此语句为数据库创建一份副本以及在该副本中启用表加密。

语法 1 - 创建数据库的加密副本

CREATE ENCRYPTED DATABASE *newfile*
FROM *oldfile*
[**KEY** *newkey*]
[**OLD KEY** *oldkey*]
[**ALGORITHM** *algorithm*

algorithm :
| **'SIMPLE'**
| **'AES'**
| **'AES256'**
| **'AES_FIPS'**
| **'AES256_FIPS'**

语法 2 - 创建启用表加密的数据库的副本

CREATE ENCRYPTED TABLE DATABASE *newfile*
FROM *oldfile*
[**KEY** *newkey*]
[**OLD KEY** *oldkey*]
[**ALGORITHM** *algorithm*]

参数

- **ENCRYPTED DATABASE 子句** 使用此子句指定新加密数据库的名称。

- **ENCRYPTED TABLE DATABASE 子句** 使用此子句指定新数据库的名称。该新数据库未加密，但已启用表加密。
- **FROM 子句** 使用此子句指定原始数据库的名称 (*oldfile*)。
- **KEY 子句** 如果 *algorithm-key* 不是 SIMPLE，使用此子句指定 *newfile* 的加密密钥。
- **OLD KEY 子句** 使用此子句指定 *oldfile* 的加密密钥。仅当 *oldfile* 使用 SIMPLE 之外的任何其它加密方法进行加密时才需要此子句。
- **ALGORITHM 子句** 使用此子句指定要用于 *newfile* 的加密算法。如果指定 KEY 子句但未指定 ALGORITHM 子句，缺省情况下将使用 AES（128 位加密）。如果指定 SIMPLE 作为 *algorithm*，则不必指定 KEY 子句。

注释

oldfile 可以是未加密数据库、已加密数据库或启用了表加密的数据库。

语法 1 使用现有数据库 *oldfile*，并为其创建了一个加密副本 *newfile*。

语法 2 使用现有数据库 *oldfile*，并为其创建了一个启用了表加密的副本 *newfile*。使用此语法时，在 *oldfile* 中加密的所有表同样在 *newfile* 中进行了加密。如果在 *oldfile* 中未加密任何表，但又想加密它们，则可对要加密的各个表执行 ALTER TABLE ...ENCRYPTED 语句。请参见“ALTER TABLE 语句”一节第 373 页。

这两种语法都不会替换或移除 *oldfile*。

如果 *oldfile* 使用事务日志或事务日志镜像文件，它们将分别被重命名为 *newfile.log* 和 *newfile.mlg*。

如果 *oldfile* 包含 dbspace 文件，文件名中将添加一个 E（代表已加密）。例如，在执行 CREATE ENCRYPTED DATABASE 语句后，文件 *mydbspace.dbs* 将更改为 *mydbspace.dbsE*。

可使用此语句来更改数据库的加密算法和密钥。不过，CREATE ENCRYPTED DATABASE 语句会生成一个新文件 (*newfile*)，而不是替换或移除先前版本的文件 (*oldfile*)。

CREATE ENCRYPTED DATABASE 和 CREATE ENCRYPTED TABLE DATABASE 不能针对需要恢复的数据库运行。另外，过程、触发器、事件或批处理中均不支持这两个语句。

有关简单加密和高度加密的详细信息，请参见“简单加密”一节 《SQL Anywhere 服务器 - 数据库管理》和“高度加密”一节 《SQL Anywhere 服务器 - 数据库管理》。

通过卸载数据库然后联合使用 dbunload -an 选项与 -ek 或 -ep 重装数据库，还可以加密现有数据库或更改现有加密密钥。请参见“使用 dbunload 实用程序重建数据库”一节 《SQL Anywhere 服务器 - SQL 的用法》。

还可以使用 CREATE DATABASE 语句创建一个加密的数据库或创建一个启用表加密的数据库。请参见“CREATE DATABASE 语句”一节第 413 页。

注意
并非所有平台上都可以使用 FIPS。有关受支持的平台的列表，请参见 http://www.sybase.com/detail?id=1062617。

权限

必须是具有 DBA 权限的用户。

副作用

无。

另请参见

- “加密和解密数据库”一节 《SQL Anywhere 服务器 - 数据库管理》
- “表加密”一节 《SQL Anywhere 服务器 - 数据库管理》
- “简单加密”一节 《SQL Anywhere 服务器 - 数据库管理》
- “高度加密”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CREATE DECRYPTED DATABASE 语句”一节第 421 页
- “CREATE DATABASE 语句”一节第 413 页
- “ALTER TABLE 语句”一节第 373 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例将为演示数据库创建一个加密副本，名为 *demoEnc.db*。新数据库使用 AES256 加密进行了加密。

```
CREATE ENCRYPTED DATABASE 'demoEnc.db'
   FROM 'demo.db'
   KEY 'Sd8f6654*Mnn'
   ALGORITHM 'AES256';
```

以下示例将为演示数据库创建一个副本，名为 *demoTableEnc.db*。新数据库启用了表加密。由于指定密钥时没有采用任何算法，因此使用 AES 加密。

```
CREATE ENCRYPTED TABLE DATABASE 'demoTableEnc.db'
   FROM 'demo.db'
   KEY 'Sd8f6654';
```

CREATE ENCRYPTED FILE 语句

为数据库文件、事务日志、事务日志镜像或 dbspace 创建一个高度加密的副本。

语法

CREATE ENCRYPTED FILE *newfile*
FROM *oldfile*
{ **KEY** *key* | **KEY** *key* **OLD KEY** *oldkey* }
[**ALGORITHM** {
  'AES'
  | **'AES256'**
  | **'AES_FIPS'**
  | **'AES256_FIPS'** }]

参数

- **FROM 子句** 指定要在其上执行 CREATE ENCRYPTED FILE 语句的现有文件 (*oldfile*) 的名称。
- **KEY 子句** 指定要使用的加密密钥。

- **OLD KEY 子句** 指定用于加密文件的当前密钥。
- **ALGORITHM 子句** 指定用于加密文件的算法。如果未指定算法，将使用缺省的 AES（128 位加密）算法。

注释

当您的数据库需要恢复以及出于支持的原因需要为数据库创建加密副本时，请使用此语句。还必须要使用此语句对任何与数据库相关的文件（例如事务日志、事务日志镜像或 dbspace 文件）进行加密。

对与数据库相关的文件进行加密时，必须对与该同一数据库相关的所有文件指定同一算法和密钥。

如果 oldfile 有 dbspace 或日志与其相关联，且您还要对它们进行加密，则必须确保这些文件的新名称和位置与新数据库存储在一起。为此，请：

- 对新数据库运行 dblog -t 以更改事务日志的名称和位置
- 对新数据库运行 dblog -m 以更改事务日志镜像的名称和位置
- 对新数据库执行 ALTER DBSPACE 语句以更改 dbspace 文件的位置和名称

可使用此语句来更改数据库的加密算法和密钥。不过，CREATE ENCRYPTED FILE 语句会生成一个新文件 (*newfile*)，而不是替换或移除先前版本的文件 (*oldfile*)。

在上述过程中，事务日志文件的名称保持不变。因此，如果数据库和事务日志文件被重命名，则需要对得到的数据库运行 dblog -t 命令。

通过卸载数据库然后联合使用 dbunload -an 选项与 -ek 或 -ep 重装数据库，还可以加密现有数据库或更改现有加密密钥。

如果具有已启用表加密的数据库，则无法使用此语句对数据库进行加密。但是，可以使用此语句更改用于表加密的密钥。要对已启用表加密的数据库进行加密，请使用 CREATE ENCRYPTED DATABASE 语句。请参见“CREATE ENCRYPTED DATABASE 语句”一节第 425 页。

此语句在过程、触发器、事件或批处理中不受支持。

> **注意**
> 并非所有平台上都可以使用 FIPS。有关受支持的平台的列表，请参见 http://www.sybase.com/detail?id=1062617。

权限

必须是具有 DBA 权限的用户。

在 Windows Mobile 上，仅 ARM 处理器支持 AES_FIPS 和 AES256_FIPS 算法。

副作用

无。

另请参见

- “加密和解密数据库”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CREATE ENCRYPTED DATABASE 语句”一节第 425 页
- “CREATE DECRYPTED FILE 语句”一节第 422 页
- “卸载实用程序 (dbunload)”一节 《SQL Anywhere 服务器 - 数据库管理》
- “事务日志实用程序 (dblog)”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例加密 contacts 数据库，并创建一个用 AES_FIPS 加密技术加密的名为 contacts2 的新数据库。

```
CREATE ENCRYPTED FILE 'contacts2.db'
FROM 'contacts.db'
   KEY 'Sd8f6654*Mnn'
   ALGORITHM AES_FIPS;
```

以下示例加密 contacts 数据库和 contacts 日志文件，并重命名这两个文件。您需要运行 `dblog -ek Sd8f6654*Mnn -t contacts2.log contacts.db`，因为虽然日志已重命名，但数据库文件仍指向旧的日志。

```
CREATE ENCRYPTED FILE 'contacts2.db'
   FROM 'contacts.db'
   KEY 'Sd8f6654*Mnn';
CREATE ENCRYPTED FILE 'contacts2.log'
   FROM 'contacts.db'
   KEY 'Sd8f6654*Mnn';
```

以下示例加密 contacts 数据库和 contacts 日志文件，以使原始日志文件名保持不变。在这种情况下，不需要运行 dblog，因为文件的名称并未改变。

```
CREATE ENCRYPTED FILE 'newpath\contacts.db'
   FROM 'contacts.db'
   KEY 'Sd8f6654*Mnn';
CREATE ENCRYPTED FILE 'newpath\contacts.log'
   FROM 'contacts.log'
   KEY 'Sd8f6654*Mnn';
```

若要更改数据库的加密密钥，首先使用新的密钥创建数据库文件的副本，如以下语句中所示：

```
CREATE ENCRYPTED FILE 'newcontacts.db'
 FROM 'contacts.db'
 KEY 'newkey' OLD KEY 'oldkey';
```

加密文件创建完成后，删除 *contacts.db* 然后将 *newcontacts.db* 重命名为 *contacts.db*。

CREATE EVENT 语句

此语句用于定义事件及其自动执行预定义操作的关联处理程序，以及用于定义调度操作。

语法

CREATE EVENT [*owner.*]*event-name*
[**TYPE** *event-type*
[**WHERE** *trigger-condition* [**AND** *trigger-condition*] ...]
| **SCHEDULE** *schedule-spec*, ...]
[**ENABLE** | **DISABLE**]
[**AT** { **CONSOLIDATED** | **REMOTE** | **ALL** }]
[**HANDLER**
BEGIN
...
END]

event-type :
BackupEnd
| **Connect**
| **ConnectFailed**
| **DatabaseStart**
| **DBDiskSpace**
| **Deadlock**
| **"Disconnect"**
| **GlobalAutoincrement**
| **GrowDB**
| **GrowLog**
| **GrowTemp**
| **LogDiskSpace**
| **MirrorFailover**
| **MirrorServerDisconnect**
| **RAISERROR**
| **ServerIdle**
| **TempDiskSpace**

trigger-condition :
event_condition(*condition-name* **)** {
=
| **<**
| **>**
| **!=**
| **<=**
| **>=**
} *value*

schedule-spec :
[*schedule-name*]
{ **START TIME** *start-time* | **BETWEEN** *start-time* **AND** *end-time* }
[**EVERY** *period* { **HOURS** | **MINUTES** | **SECONDS** }]
[**ON** { **(** *day-of-week*, ... **)** | **(** *day-of-month*, ... **)** }]
[**START DATE** *start-date*]

event-name : *identifier*

schedule-name : *identifier*

day-of-week : *string*

day-of-month : *integer*

value : *integer*

period : *integer*

start-time : *time*

end-time : *time*

start-date : *date*

参数

- **CREATE EVENT 子句** 事件名是一个标识符。事件都有创建者，即创建事件的用户。事件处理程序只能在创建者的许可下执行。这与存储过程的执行是一样的。无法创建其他用户所拥有的事件。
- **TYPE 子句** 可以指定带可选 WHERE 子句的 TYPE 子句，或者指定 SCHEDULE。

  event-type 是所列出的一组系统定义事件类型中的一种。事件类型不区分大小写。要指定该 *event-type* 触发事件的条件，请使用 WHERE 子句。有关未在下文中列出的事件类型的说明，请参见“了解系统事件”一节 《SQL Anywhere 服务器 - 数据库管理》。

  - **DiskSpace 事件类型** 如果数据库包含 DiskSpace 类型之一的事件处理程序，则数据库服务器每隔 30 秒对每一个与相关文件关联的设备上的可用空间进行检查。

    如果数据库有多个 dbspace 位于不同的驱动器，则 DBDiskSpace 会检查每个驱动器并根据最低的可用空间执行操作。

    LogDiskSpace 事件类型检查事务日志和任何事务日志镜像的位置，并根据最小的可用空间进行报告。

    Windows Mobile 不支持磁盘空间事件类型。

    TempDiskSpace 事件类型检查临时磁盘空间的大小。

    如果定义了相应的事件处理程序（DBDiskSpace、LogDiskSpace 或 TempDiskSpace），则数据库服务器每隔 30 秒便会检查一次每个与数据库文件关联的设备的可用空间。同样，在定义了处理系统事件类型 ServerIdle 的事件后，如果在前 30 秒内未处理任何请求，则数据库服务器将通知该处理程序。

    您可以在启动数据库服务器时指定 -fc 选项，以在数据库服务器遇到文件系统已满时执行回调函数。

    请参见“-fc 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

  - **GlobalAutoIncrement 事件类型** 如果 GLOBAL AUTOINCREMENT 的剩余值数目低于其范围末尾的 1%，则*每次*进行插入时都将触发此事件。处理程序的典型操作可能是：基于表和提供给此事件作为参数的剩余值数，为 global_database_id 选项请求新值。

    对这种事件类型，可以使用以 RemainingValues 为参数的 event_condition 函数。

  - **ServerIdle 事件类型** 如果数据库包含 ServerIdle 类型的事件处理程序，则数据库服务器每隔 30 秒便会检查一次服务器的活动。

- **数据库镜像事件类型** 当主数据库服务器与镜像服务器或仲裁服务器之间的连接丢失时会触发 MirrorServerDisconnect 事件，且只要服务器取得数据库所有权就会触发 MirrorFailover 事件。请参见“数据库镜像系统事件”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **WHERE 子句** 此触发条件确定在什么情况下触发事件。例如，如果希望在含有事务日志的磁盘上有 80% 以上已满时执行操作，请使用以下触发条件：

```
...
WHERE event_condition( 'LogDiskSpacePercentFree' ) < 20
...
```

  event_condition 函数的参数必须对该事件类型有效。

  可以使用多个 AND 条件来构成 WHERE 子句，但不能使用 OR 条件或其它条件。

  有关有效参数的信息，请参见“EVENT_CONDITION 函数 [System]”一节第 189 页。

- **SCHEDULE 子句** 此子句指定调度操作发生的时间。时间序列充当事件处理程序中定义的关联操作的一组触发条件。

  可以为给定事件及其关联的处理程序创建多个调度。这样可以实现复杂的调度。有多个调度时必须提供调度名称，而仅提供一个调度时，这是可选的。

  如果调度事件的定义中含有 EVERY 或 ON，则调度事件将反复出现；如果未使用这些保留字，则事件最多会执行一次。如果试图创建开始时间为过去时间的非反复出现调度事件，将会出现错误。在非反复出现的调度事件已过去之后，将删除其调度，但不删除事件处理程序。

  调度事件的时间从调度创建时开始计算，并在事件处理程序完成执行时重新计算。调度事件的时间从调度创建时开始计算，并在事件处理程序完成执行时重新计算。如果指示事件处理程序在 9:00 和 5:00 之间每隔一小时运行一次，并且它需要 65 分钟来执行，则它将分别在 9:00、11:00、1:00、3:00 和 5:00 运行。如果要重叠执行，则必须创建多个事件。

以下是调度定义的子句：

- **START TIME 子句** 调度事件的每天中的最初调度时间。*start-time* 参数为字符串，不能为变量或表达式，如 `NOW()`。如果指定了 START DATE，则 START TIME 引用该日期。如果未指定 START DATE，则 START TIME 在当天（除非该时间已经过去）和随后的每一天（如果调度中含有 EVERY 或 ON）。

- **BETWEEN ...AND 子句** 显示一天中的一段时间，在该时间段之外，没有调度的时间。*start-time* 和 *end-time* 参数为字符串，不能为变量或表达式，如 `NOW()`。如果指定了 START DATE，则调度的时间直到该日期才存在。

- **EVERY 子句** 连续调度事件之间的间隔。调度的事件仅在当天的 START TIME 之后或在 BETWEEN ...AND 指定的范围内发生。

- **ON 子句** 调度事件发生日的列表。如果指定了 EVERY，则缺省为每天。事件发生日可指定为周内某日或月内某日。

  周内某日是周一、周二等等。您还可以使用完整形式的日期，例如 Monday。当您所使用的语言不是英语、不是连接字符串中客户端请求的语言、也不是数据库服务器消息窗口中出现的语言时，您必须使用完整形式的日期名。

  月内某日是从 0 到 31 的整数。0 值表示任何月的最后一天。

- **START DATE 子句** 调度事件开始发生的日期。此参数为字符串，不能为变量或表达式，如 TODAY()。缺省为当前日期。

  每当调度事件的处理程序完成后，即开始计算下一个调度时间和日期。

  1. 如果使用了 EVERY 子句，则查看下次调度时间是否在当天，以及是否在 BETWEEN ...AND 范围的开始时间。如果是，即为下次调度时间。
  2. 如果下一个调度时间不在当天，请查找下一个执行事件的日期。
  3. 查找此日期的 START TIME，或 BETWEEN ...AND 范围的开始时间。

- **ENABLE | DISABLE 子句** 缺省情况下，启用事件处理程序。指定 DISABLE 后，即使是在调度时间内或满足触发条件时，事件处理程序也不会执行。但是，TRIGGER EVENT 语句不会导致已禁用的事件处理程序被执行。

- **AT 子句** 应只在以下情况下使用此语句：在 SQL Remote 设置中，可对远程或统一数据库使用此子句，以限制在哪一个数据库中处理事件。

  如果为 SQL Remote 创建事件时没有使用 AT 子句，那么所有数据库都会执行事件。当在统一数据库上执行时，此语句不会影响已被提取的远程数据库。

- **HANDLER 子句** 每个事件都有一个处理程序。

注释

主要有两种使用事件的方式：

- **调度操作** 数据库服务器按时间表执行操作。可使用此功能完成计划任务，如备份、有效性检查和用于将数据添加到报告表的查询。
- **事件处理操作** 数据库服务器在发生预定义事件时执行操作。可使用此功能完成计划任务，例如在磁盘使用超过了指定的百分比时限制磁盘空间。如果在执行期间未检测到错误，则提交事件处理程序操作；如果检测到错误，则回退。

事件定义包括两个截然不同的方面。触发器条件可以是一个事件，如磁盘使用超出了定义的阈值。调度是一组时间，其中每个时间充当一个触发器条件。当满足触发器条件时，执行事件处理程序。事件处理程序包含一个或多个在复合语句 (BEGIN...END) 内声明 SQL 变量。

如果未提供触发条件或调度说明，则只有显式 TRIGGER EVENT 语句才能触发事件。在开发过程中，您可能要使用 TRIGGER EVENT 测试事件处理程序，并可能要在测试完成后添加调度或 WHERE 子句。

任何事件错误都会记录到数据库服务器消息日志中。请参见“记录数据库服务器操作”一节《SQL Anywhere 服务器 - 数据库管理》。

每次执行事件处理程序后，如果没有出现错误便会发生 COMMIT。如果出现错误，则会发生 ROLLBACK。

触发事件处理程序后，数据库服务器通过使用 event_parameter 函数，来使上下文信息（例如导致事件被触发的连接 ID）可供事件处理程序使用。有关 event_parameter 的详细信息，请参见“EVENT_PARAMETER 函数 [System]”一节第 191 页。

权限

必须具有 DBA 权限。

在事件所有者的许可下，事件处理程序在一个单独的连接上执行。要使用 DBA 以外的权限进行执行，可以从事件处理程序内调用过程：使用过程所有者的权限执行过程。此单独连接不计入个人数据库服务器只能有 10 个连接的限制中。

副作用

自动提交。

另请参见

- “BEGIN 语句”一节第 395 页
- “ALTER EVENT 语句”一节第 351 页
- “COMMENT 语句”一节第 406 页
- “DROP EVENT 语句”一节第 545 页
- “TRIGGER EVENT 语句”一节第 726 页
- “EVENT_PARAMETER 函数 [System]”一节第 191 页
- “了解系统事件”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

指示数据库服务器在每天凌晨 1 点，使用第一个磁带驱动器执行自动的磁带备份。

```
CREATE EVENT DailyBackup
SCHEDULE daily_backup
START TIME '1:00AM' EVERY 24 HOURS
HANDLER
   BEGIN
      BACKUP DATABASE TO '\\\\.\\tape0'
      ATTENDED OFF
   END;
```

指示数据库服务器从星期一到星期五，在每天上午 8 点到下午 6 点之间，每隔一小时执行一次仅对事务日志的自动备份。

```
CREATE EVENT HourlyLogBackup
SCHEDULE hourly_log_backup
BETWEEN '8:00AM' AND '6:00PM'
EVERY 1 HOURS ON
   ('Monday','Tuesday','Wednesday','Thursday','Friday')
HANDLER
   BEGIN
      BACKUP DATABASE DIRECTORY 'c:\\database\\backup'
      TRANSACTION LOG ONLY
      TRANSACTION LOG RENAME
   END;
```

请参见“定义事件的触发条件”一节 《SQL Anywhere 服务器 - 数据库管理》。

CREATE EXISTING TABLE 语句

此语句用于创建新的代理表。代理表表示远程服务器上的现有对象。

语法

CREATE EXISTING TABLE [*owner*.]*table-name*
[**(***column-definition*, ...**)**]
AT *location-string*

column-definition :
column-name data-type [**NOT NULL**]

location-string :
remote-server-name.[*db-name*].[*owner*].*object-name*
| *remote-server-name*;[*db-name*];[*owner*];*object-name*

参数

- **AT 子句** AT 子句指定远程对象的位置。AT 子句支持将分号 (;) 用作分隔符。如果分号出现在 *location-string* 字符串中的任何位置，则分号将用作字段分隔符。如果没有分号，则将句号用作字段分隔符。这样一来，便可在数据库和所有者字段中使用文件名和扩展名。例如，下列语句将表 a1 映射到 MS Access 文件 *mydbfile.mdb*：

```
CREATE EXISTING TABLE a1
AT 'access;d:\mydbfile.mdb;;a1';
```

注释

CREATE EXISTING TABLE 语句创建新的本地代理表，该代理表映射到处于外部位置的表。CREATE EXISTING TABLE 语句是 CREATE TABLE 语句的变化形式。EXISTING 关键字与 CREATE TABLE 一起使用时，指定已存在于远程位置的表，并指定其元数据将导入到 SQL Anywhere 中。以这种方式可将远程表建立为 SQL Anywhere 用户能够看得见的实体。SQL Anywhere 在创建表之前，会校验它是否存在于外部位置。

如果对象不存在（主机数据文件或远程服务器对象），此语句将被拒绝并伴随出现错误消息。

从主机数据文件或远程服务器表中抽取索引信息，并将其用于创建系统表 ISYSIDX 的行。这定义了服务器术语中的索引和键，并使查询优化程序考虑此表上可能存在的任何索引。

参照约束在适当的时候传递到远程位置。

如果未指定列定义，SQL Anywhere 会根据它从远程表中获得的元数据来派生列的列表。如果指定了列定义，SQL Anywhere 会校验列定义。将对列名、数据类型、长度、标识属性和空值属性进行以下检查：

- 列名称必须完全匹配（但忽略大小写）。
- CREATE EXISTING TABLE 语句中的数据类型必须匹配或者可转换成远程位置的列的数据类型。例如，本地列的数据类型定义为货币，而远程列的数据类型则为数字。
- 检查每列的 NULL 属性。如果本地列的 NULL 属性与远程列的 NULL 属性不同，则会发出警告消息，但不会中止语句。

- 检查每列的长度。如果 char、varchar、二进制、varbinary、十进制和数字类型的列的长度不匹配，则会发出警告消息，但不会中止命令。

  可以选择在 CREATE EXISTING 语句中仅包含实际远程列列表的子集。

权限

必须具有 RESOURCE 权限。要为其他用户创建表，必须具有 DBA 权限。

Windows Mobile 上不支持。

副作用

自动提交。

另请参见

- “CREATE TABLE 语句”一节第 497 页
- “指定代理表位置”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在远程服务器 server_a 上，为 blurbs 表创建名为 blurbs 的代理表。

```
CREATE EXISTING TABLE blurbs
( author_id ID not null,
copy text not null)
AT 'server_a.db1.joe.blurbs';
```

在远程服务器 server_a 上，为 blurbs 表创建名为 blurbs 的代理表。SQL Anywhere 会根据它从远程表中获得的元数据来派生列的列表。

```
CREATE EXISTING TABLE blurbs
AT 'server_a.db1.joe.blurbs';
```

在 SQL Anywhere 远程服务器 demo11 上，为 Employees 表创建名为 rda_employees 的代理表。

```
CREATE EXISTING TABLE rda_employees
AT 'demo11...Employees';
```

CREATE EXTERNLOGIN 语句

此语句用于指派与远程服务器通信时使用的替代登录名和口令。

语法

CREATE EXTERNLOGIN *login-name*
TO *remote-server*
[**REMOTE LOGIN** *remote-user* [**IDENTIFIED BY** *remote-password*]]

参数

- **login-name** 指定本地用户登录名。使用集成登录时，*login-name* 是 Windows 用户或组所映射到的数据库用户。
- **TO 子句** TO 子句指定远程服务器的名称。
- **REMOTE LOGIN 子句** REMOTE LOGIN 子句在 *remote-server* 上为本地用户 *login-name* 指定用户帐户。
- **IDENTIFIED BY 子句** IDENTIFIED BY 子句为 *remote-user* 指定 *remote-password*。该 *remote-user* 和 *remote-password* 组合必须是远程服务器上的有效组合。

  如果省略 IDENTIFIED BY 子句，则会将 NULL 作为口令发送给远程服务器。但是，如果指定 IDENTIFIED BY ""（空字符串），则所发送的口令为空字符串。

注释

缺省情况下，SQL Anywhere 每次代表其客户端连接到远程服务器时都会使用这些客户端的名称和口令。CREATE EXTERNLOGIN 指派与远程服务器通信时要使用的备用登录名和口令。

仅当远程服务器需要用户 ID 和口令进行连接时，才需要 REMOTE LOGIN 子句。使用外部登录而非远程登录时，DBA 可控制能够访问远程服务器的人员，并告知远程访问层，登录到远程服务器不需要用户 ID 和口令。例如，目录访问服务器类需要外部登录来限制对目录服务器的访问，但不需要远程登录，因为目录服务器不执行用户 ID 和口令验证。

口令以加密形式存储在内部。*remote-server* 必须通过 ISYSSERVER 表中的条目告知本地服务器。请参见“CREATE SERVER 语句”一节第 481 页。

具有自动口令失效功能的站点应该就定期更新外部登录口令作出计划。

CREATE EXTERNLOGIN 不能从事务内部使用。

权限

只有具有 DBA 权限的用户才能添加或修改 *login-name* 的外部登录。

Windows Mobile 上不支持。

副作用

自动提交。

另请参见

- “DROP EXTERNLOGIN 语句”一节第 545 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

连接到服务器 sybase1 时，使用口令 Plankton 将名为 DBA 的本地用户映射到用户 sa。

```
CREATE EXTERNLOGIN DBA
TO sybase1
```

```
REMOTE LOGIN sa
IDENTIFIED BY Plankton;
```

CREATE FUNCTION 语句

使用此语句在数据库中创建用户定义的 SQL 函数。要创建外部函数接口，请参见“CREATE FUNCTION 语句（外部过程）”一节第 441 页。要创建 Web 服务函数，请参见“CREATE FUNCTION 语句（Web 服务）”一节第 445 页。

CREATE [**OR REPLACE** | **TEMPORARY**] **FUNCTION** [*owner*.]*function-name*
([*parameter*, ...] **)**
RETURNS *data-type*
[**SQL SECURITY** { **INVOKER** | **DEFINER** }]
[**ON EXCEPTION RESUME**]
[[**NOT**] **DETERMINISTIC**]
compound-statement | **AS** *tsql-compound-statement*

parameter :
  [**IN**] *parameter-name data-type* [**DEFAULT** *expression*]

tsql-compound-statement:
sql-statement
sql-statement
...

参数

- **OR REPLACE 子句** 指定 CREATE OR REPLACE FUNCTION 将创建一个新函数或替换同名的现有函数。如果是替换，此子句将更改函数的定义，但保留现有权限。

  不能将 OR REPLACE 子句与临时函数一起使用。

- **TEMPORARY 关键字** 如果指定 CREATE TEMPORARY FUNCTION，则将意味着该函数仅对创建它的连接可见，并在删除该连接时随之自动删除。也可以显式删除临时函数。无法在临时函数上执行 ALTER、GRANT 或 REVOKE，而且与其它函数不同，临时函数不会在目录或事务日志中予以记录。

  临时函数使用其创建者（当前用户）或指定所有者的权限进行执行。可在以下情况下为临时函数指定所有者：

  - 临时函数是在永久存储过程中创建的
  - 临时函数与永久存储过程的所有者相同

  要删除临时函数的所有者，必须首先删除该临时函数。

  临时函数可在连接到只读数据库时加以创建和删除。

  不能将 OR REPLACE 子句与临时函数一起使用。

- **SQL SECURITY 子句** SQL SECURITY 子句定义该函数是作为 INVOKER（调用该函数的用户）执行还是作为 DEFINER（拥有该函数的用户）执行。缺省值为 DEFINER。

- **compound-statement** 一组用 BEGIN 和 END 括起来的 SQL 语句，中间用分号分隔。请参见“BEGIN 语句”一节第 395 页。

- **tsql-compound-statement** 一批 Transact-SQL 语句。请参见“Transact-SQL 批处理概述”一节 《SQL Anywhere 服务器 - SQL 的用法》和“CREATE PROCEDURE 语句 [T-SQL]”一节 第 464 页。
- **ON EXCEPTION RESUME 子句** 使用 Transact-SQL -like 错误处理。请参见“CREATE PROCEDURE 语句”一节第 458 页。
- **[NOT] DETERMINISTIC 子句** 使用此子句指明函数是确定型函数还是非确定型函数。如果忽略此子句，便没有指定函数的确定型行为（缺省设置）。

  如果将某个函数声明为 DETERMINISTIC，则在每次使用同一组参数调用该函数时，应返回同一个值。

  如果将某个函数声明为 NOT DETERMINISTIC，则无法保证对于同一组参数会返回同一个值。声明为 NOT DETERMINISTIC 的函数每次在查询中调用时都将重新求值。如果已知给定一组参数的函数结果可以不同，则必须使用此子句。

  另外，对于具有副作用（如修改基础数据）的函数，应将其声明为 NOT DETERMINISTIC。例如，一个生成主键值且用于 INSERT ... SELECT 语句的函数应声明为 NOT DETERMINISTIC：

```
CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC
BEGIN
  DECLARE keyval INTEGER;
  UPDATE counter SET x = x + increment;
  SELECT counter.x INTO keyval FROM counter;
  RETURN keyval
END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table;
```

  如果函数对给定输入参数总是返回相同的值，则该函数可以声明为 DETERMINISTIC。

注释

CREATE FUNCTION 语句在数据库中创建函数。通过指定所有者名称，可以为其他用户创建函数。根据权限，可通过与其它非集合函数完全相同的方法使用函数。

参数名必须符合数据库标识符规则。它们必须是有效的 SQL 数据类型，且必须以关键字 IN 作为前缀，以表明参数是为函数提供值的表达式。

执行函数时，不必指定所有参数。如果在 CREATE FUNCTION 语句中提供了缺省值，则会为缺少的参数指派缺省值。如果调用程序既未提供参数又未设置缺省值，则会给出错误。

指定 SQL SECURITY INVOKER 后，必须对每个调用该过程的用户加以标注，因此会使用更多内存。另外，指定 SQL SECURITY INVOKER 后，也会作为调用者进行名称解析。因此，应注意用适合的所有者限定所有对象名称（表、过程等）。

除非所有函数声明为 NOT DETERMINISTIC，否则它们将被视为确定型。确定型函数为相同的参数返回一致的结果，并且没有副作用。即，数据库服务器假定对具有相同参数的同一函数连续进行两次调用将返回相同的结果，并且不会对查询的语义产生任何不良的副作用。

如果函数返回一个结果集，则它不能同时设置输出参数或返回一个返回值。

权限

除非创建临时函数，否则必须具有 RESOURCE 权限。

外部函数（包括 Java 函数）必须有 DBA 权限。

副作用

自动提交。

另请参见

- “ALTER FUNCTION 语句”一节第 354 页
- “CREATE FUNCTION 语句（外部过程）”一节第 441 页
- “CREATE FUNCTION 语句（Web 服务）”一节第 445 页
- “BEGIN 语句”一节第 395 页
- “CREATE PROCEDURE 语句”一节第 458 页
- “DROP FUNCTION 语句”一节第 546 页
- “RETURN 语句”一节第 678 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

下面的函数将 firstname 字符串和 lastname 字符串串联在一起。

```
CREATE FUNCTION fullname(
   firstname CHAR(30),
   lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
   DECLARE name CHAR(61);
   SET name = firstname || ' ' || lastname;
   RETURN (name);
END;
```

以下示例将替换在第一个示例中创建的 fullname 函数。在代替该函数后，将移除局部变量 `name`：

```
CREATE OR REPLACE FUNCTION fullname(
   firstname CHAR(30),
   lastname CHAR(30) )
RETURNS CHAR(61)
BEGIN
   RETURN = firstname || ' ' || lastname;
END;
```

以下示例说明 fullname 函数的用法。

从两个提供的字符串中返回完整的名称：

```
SELECT fullname ( 'joe', 'smith' );
```


fullname('joe', 'smith')
joe smith

列出所有雇员的姓名：

```
SELECT fullname ( GivenName, Surname )
FROM Employees;
```

fullname (GivenName, Surname)
Fran Whitney
Matthew Cobb
Philip Chin
Julie Jordan
...

以下函数使用 Transact-SQL 语法：

```
CREATE FUNCTION DoubleIt( @Input INT )
RETURNS INT
AS
BEGIN
  DECLARE @Result INT
  SELECT @Result = @Input * 2
  RETURN @Result
END;
```

语句 `SELECT DoubleIt( 5 )` 返回值 `10`。

CREATE FUNCTION 语句（外部过程）

使用此语句创建一个连接本地或外部函数的接口。要创建用户定义的 SQL 函数，请参见“CREATE FUNCTION 语句”一节第 438 页。

语法

CREATE [**OR REPLACE**] **FUNCTION** [*owner*.]*function-name*
([*parameter*, ...] **)**
RETURNS *data-type*
[**SQL SECURITY** { **INVOKER** | **DEFINER** }]
[[**NOT**] **DETERMINISTIC**]
EXTERNAL NAME *external-call* [**LANGUAGE** *environment-name*]

parameter :
[**IN**] *parameter-name data-type* [**DEFAULT** *expression*]

environment-name :
C_ESQL32
| **C_ESQL64**
| **C_ODBC32**
| **C_ODBC64**
| **CLR**
| **JAVA**
| **PERL**
| **PHP**

参数

- **CREATE FUNCTION** 您可以创建用于调用使用各种编程语言编写的外部或本地函数的永久存储函数。

  参数名必须符合其它数据库标识符（如列名）的规则。它们必须是有效的 SQL 数据类型。有关有效数据类型的列表，请参见“SQL 数据类型”第 75 页。

  参数可以使用关键字 IN 作为前缀。但在缺省情况下，函数参数是 IN。

  ○ **IN** 此参数是一个为函数提供值的表达式。

  执行函数时，不必指定所有参数。如果在 CREATE FUNCTION 语句中提供了缺省值，则会为缺少的参数指派缺省值。如果在执行函数时未提供参数也未设置缺省值，则会给出错误。

  指定 OR REPLACE (CREATE OR REPLACE FUNCTION) 将创建一个新函数或替换同名的现有函数。此子句将更改函数的定义，但保留现有权限。

  TEMPORARY 函数不支持 EXTERNAL NAME 子句。

- **[NOT] DETERMINISTIC 子句** 使用此子句指明函数是确定型函数还是非确定型函数。如果忽略此子句，便没有指定函数的确定型行为（缺省设置）。

  如果将某个函数声明为 DETERMINISTIC，则在每次使用同一组参数调用该函数时，应返回同一个值。

  如果将某个函数声明为 NOT DETERMINISTIC，则无法保证对于同一组参数会返回同一个值。声明为 NOT DETERMINISTIC 的函数每次在查询中调用时都将重新求值。如果已知给定一组参数的函数结果可以不同，则必须使用此子句。

  另外，对于具有副作用（如修改基础数据）的函数，应将其声明为 NOT DETERMINISTIC。例如，一个生成主键值且用于 INSERT ... SELECT 语句的函数应声明为 NOT DETERMINISTIC：

```
CREATE FUNCTION keygen( increment INTEGER )
RETURNS INTEGER
NOT DETERMINISTIC
BEGIN
  DECLARE keyval INTEGER;
  UPDATE counter SET x = x + increment;
  SELECT counter.x INTO keyval FROM counter;
  RETURN keyval
END
INSERT INTO new_table
SELECT keygen(1), ...
FROM old_table;
```

  如果函数对给定输入参数总是返回相同的值，则该函数可以声明为 DETERMINISTIC。

- **SQL SECURITY 子句** SQL SECURITY 子句定义该函数是作为 INVOKER（调用该函数的用户）执行还是作为 DEFINER（拥有该函数的用户）执行。缺省值为 DEFINER。对于外部调用，此子句用于为外部环境中的非限定对象引用建立所有权上下文。

  指定 SQL SECURITY INVOKER 后，必须对每个调用该函数的用户加以标注，因此会使用更多内存。另外，指定 SQL SECURITY INVOKER 后，也会作为调用者进行名称解析。因此，应注意用适合的所有者限定所有对象名称（表、过程等）。例如，假定 user1 创建了以下函数：

```
CREATE FUNCTION user1.myFunc()
   RETURNS INT
   SQL SECURITY INVOKER
   BEGIN
     DECLARE res INT;
     SELECT COUNT(*) INTO res FROM table1;
     RETURN res;
   END;
```

  如果 user2 试图运行此函数，而表 user2.table1 *不*存在，则会产生表查寻错误。另外，如果 user2.table1 *确实*存在，则使用该表而不使用预定的 user1.table1。为了防止出现这种情况，请在语句中限定表引用（user1.table1，而不只是 table1）。

- **EXTERNAL NAME *native-call* 子句**

  native-call :
  [*operating-system*:]*function-name*@*library*; ...

  operating-system : **Unix**
  使用不带 LANGUAGE 属性的 EXTERNAL NAME 子句的函数用于定义一个连接本地函数（使用 C 语言等编程语言编写）的接口。该本地函数由数据库服务器装载到其地址空间中。

  library 名可包含文件扩展名，通常在 Windows 中为 *.dll*，在 Unix 中为 *.so*。在没有扩展名的情况下，该软件附加平台特定的缺省库文件扩展名。以下为一个正式示例。

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';
```

  下面是使用特定于平台的缺省值编写上述 EXTERNAL NAME 子句的更简单的方法：

```
CREATE FUNCTION mystring( IN instr LONG VARCHAR )
RETURNS LONG VARCHAR
EXTERNAL NAME 'mystring@mylib';
```

  如果调用，则包含此函数的库将被装载到数据库服务器的地址空间中。本地函数将作为服务器的一部分执行。在这种情况下，如果此函数导致故障，则会使数据库服务器终止。因此，建议在外部环境中装载和执行使用 LANGUAGE 属性的函数。如果函数在外部环境中导致故障，数据库服务器仍可继续运行。

  有关本地库调用的信息，请参见“从过程调用外部库”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *c-call* LANGUAGE {C_ESQL32 | C_ESQL64 | C_ODBC32 | C_ODBC64 } 子句** 要在外部环境而非数据库服务器中调用编译后的本地 C 函数，存储过程或函数应使用后跟 LANGUAGE 属性的 EXTERNAL NAME 子句来定义，其中，LANGUAGE 属性指定 C_ESQL32、C_ESQL64、C_ODBC32 或 C_ODBC64。

  指定 LANGUAGE 属性后，包含此函数的库将由外部进程装载，外部函数将在该外部进程中执行。在这种情况下，如果此函数导致故障，数据库服务器仍可继续运行。

以下是一个示例函数定义。

```
CREATE FUNCTION ODBCinsert(
  IN ProductName CHAR(30),
  IN ProductDescription CHAR(50)
)
RETURNS INT
EXTERNAL NAME 'ODBCexternalInsert@extodbc.dll'
LANGUAGE C_ODBC32;
```

有关详细信息，请参见“ESQL 和 ODBC 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *clr-call* LANGUAGE CLR 子句** 要在外部环境中调用 .NET 函数，函数接口应使用后跟 LANGUAGE CLR 属性的 EXTERNAL NAME 子句来定义。

  CLR 存储过程或函数的行为与 SQL 存储过程或函数的基本相同，只是过程或函数的代码以 C# 或 Visual Basic 等 .NET 语言编写并在数据库服务器外（即在单独的 .NET 可执行文件内）执行。

  以下是一个示例函数定义。

  ```
  CREATE FUNCTION clr_interface(
      IN p1 INT,
      IN p2 UNSIGNED SMALLINT,
      IN p3 LONG VARCHAR)
  RETURNS INT
  EXTERNAL NAME 'CLRlib.dll::CLRproc.Run( int, ushort, string )'
  LANGUAGE CLR;
  ```

  有关详细信息，请参见“CLR 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *perl-call* LANGUAGE PERL 子句** 要在外部环境中调用 Perl 函数，函数接口应使用后跟 LANGUAGE PERL 属性的 EXTERNAL NAME 子句来定义。

  Perl 存储过程或函数的行为与 SQL 存储过程或函数基本相同，只是过程或函数的代码以 Perl 编写并在数据库服务器外（即在 Perl 可执行实例内）执行。

  以下是一个示例函数定义。

  ```
  CREATE FUNCTION PerlWriteToConsole( IN str LONG VARCHAR)
  RETURNS INT
  EXTERNAL NAME '<file=PerlConsoleExample>
      WriteToServerConsole( $sa_perl_arg0 )'
  LANGUAGE PERL;
  ```

  有关详细信息，请参见“PERL 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *php-call* LANGUAGE PHP 子句** 要在外部环境中调用 PHP 函数，函数接口应使用后跟 LANGUAGE PHP 属性的 EXTERNAL NAME 子句来定义。

  PHP 存储过程或函数的行为与 SQL 存储过程或函数基本相同，只是过程或函数的代码以 PHP 编写并且在数据库服务器外（即在 PHP 可执行实例内）执行。

  以下是一个示例函数定义。

  ```
  CREATE FUNCTION PHPPopulateTable()
  RETURNS INT
  EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
  LANGUAGE PHP;
  ```


有关详细信息，请参见“PHP 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *java-call* LANGUAGE JAVA 子句** 要在外部环境中调用 Java 方法，函数接口应使用后跟 LANGUAGE JAVA 属性的 EXTERNAL NAME 子句来定义。

  采用 Java 接口技术的存储过程或函数的行为与 SQL 存储过程或函数的行为基本相同，只是过程或函数的代码以 Java 编写并且在数据库服务器外（即在 Java 虚拟机内）执行。

  以下是一个示例函数定义。

```
CREATE FUNCTION HelloDemo( IN name LONG VARCHAR )
RETURNS INT
EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
LANGUAGE JAVA;
```

  有关详细信息，请参见“Java 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

注释

CREATE FUNCTION 语句在数据库中创建函数。具有 DBA 权限的用户可以通过指定所有者为其他用户创建函数。函数以作为 SQL 表达式一部分的形式被调用。

从多个函数引用临时表时，如果该临时表定义不一致且高速缓存引用该表的语句，则会出现潜在问题。请参见“在过程中引用临时表”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

除非创建临时函数，否则必须具有 RESOURCE 权限。

引用外部函数或者为其他用户创建函数必须有 DBA 权限。

副作用

自动提交。

另请参见

- “ALTER FUNCTION 语句”一节第 354 页
- “CALL 语句”一节第 400 页
- “CREATE FUNCTION 语句”一节第 438 页
- “CREATE FUNCTION 语句（Web 服务）”一节第 445 页
- “CREATE PROCEDURE 语句（外部过程）”一节第 465 页
- “DROP FUNCTION 语句”一节第 546 页
- “GRANT 语句”一节第 595 页
- “外部环境概述”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 持久存储模块特性。Java 结果集的语法扩展如可选的 J621 特性中所指定。

CREATE FUNCTION 语句（Web 服务）

此语句用于在数据库中新建 Web 服务函数。要创建用户定义的 SQL 函数，请参见“CREATE FUNCTION 语句”一节第 438 页。

```
CREATE [ OR REPLACE ] FUNCTION [ owner.]function-name ( [ parameter, ... ] )
RETURNS data-type
URL url-string
[ HEADER header-string ]
[ SOAPHEADER soap-header-string ]
[ TYPE {
  'HTTP[ :{ GET | POST[:MIME-type ] | PUT[:MIME-type ] | DELETE | HEAD } ]' |
  'SOAP[:{ RPC | DOC } ]' } ]
[ NAMESPACE namespace-string ]
[ CERTIFICATE certificate-string ]
[ CLIENTPORT clientport-string ]
[ PROXY proxy-string ]
[ SET protocol-option-string ]

url-string :
' { HTTP | HTTPS | HTTPS_FIPS }://[user:password@]hostname[:port][/path]'

parameter :
   [ IN ] parameter-name data-type [ DEFAULT expression ]
```

参数

- **CREATE FUNCTION** 参数名必须符合数据库标识符规则。它们必须是有效的 SQL 数据类型，且必须以关键字 IN 作为前缀，以表明参数是为函数提供值的表达式。

  执行函数时，不必指定所有参数。如果在 CREATE FUNCTION 语句中提供了缺省值，则会为缺少的参数指派缺省值。如果调用程序未提供参数，且缺省值未设置，则会给出错误。

  指定 OR REPLACE (CREATE OR REPLACE FUNCTION) 将创建一个新函数或替换同名的现有函数。此子句将更改函数的定义，但保留现有权限。不能将 OR REPLACE 子句与临时函数一起使用。

- **RETURNS 子句** 所返回的数据类型可能为 VARCHAR、BINARY、VARBINARY 或 LONG BINARY。数据类型不影响如何处理 HTTP 响应。

- **URL 子句** 仅用于定义 HTTP 或 SOAP Web 服务客户端函数。指定 Web 服务的 URL。其中的用户名和口令参数是可选的，它们提供了一种用于提供 HTTP 基本验证所需的证书的方法。HTTP 基本验证对用户和口令信息进行基于 64 位的编码，并在 HTTP 请求的验证标头中传递它。

  指定 HTTPS_FIPS 会强制系统使用 FIPS 库。如果指定了 HTTPS_FIPS，但不存在 FIPS 库，则会改为使用非 FIPS 库。

- **HEADER 子句** 创建 HTTP Web 服务客户端函数时，此子句用于添加或修改 HTTP 请求标头条目。仅可为 HTTP 标头指定可打印 ASCII 字符，且这些字符不区分大小写。有关如何使用此子句的详细信息，请参见“CREATE PROCEDURE 语句（Web 服务）”一节第 471 页中的 HEADER 子句。

  有关使用 HTTP 标头的详细信息，请参见“处理 HTTP 标头”一节 《SQL Anywhere 服务器 - 编程》。

- **SOAPHEADER 子句** 当将 SOAP Web 服务声明为函数时，此子句用于指定一个或多个 SOAP 请求标头条目。SOAP 标头可声明为静态常量，也可使用参数替代机制动态设置（为参数 hd1、hd2 等声明 IN、OUT 或 INOUT 参数）。Web 服务函数可定义一个或多个 IN 模式替代参数，

但无法定义 INOUT 或 OUT 替代参数。有关如何使用此子句的详细信息，请参见“CREATE PROCEDURE 语句（Web 服务）”一节第 471 页中的 SOAPHEADER 子句。

有关如何使用 SOAP 标头的详细信息，请参见“处理 SOAP 标头”一节 《SQL Anywhere 服务器 - 编程》。

- **TYPE 子句** 用于指定创建 Web 服务请求时使用的格式。如果指定 SOAP 或未包括类型子句，则使用缺省类型 SOAP:RPC。HTTP 隐含 HTTP:POST。由于 SOAP 请求总是作为 XML 文档发送，因此总是使用 HTTP:POST 发送 SOAP 请求。
- **NAMESPACE 子句** 仅适用于 SOAP 客户端函数。此子句标识 SOAP:RPC 和 SOAP:DOC 请求通常都需要的方法命名空间。处理请求的 SOAP 服务器使用此命名空间来解释 SOAP 请求消息主体中的实体的名称。可以通过 Web 服务服务器，从 SOAP 服务的 WSDL（Web Services Description Language，Web 服务描述语言）中获取命名空间。缺省值是函数的 URL，直到（但不包括）可选的路径组成部分。
- **CERTIFICATE 子句** 要创建安全 (HTTPS) 请求，客户端必须有权访问 HTTPS 服务器所用的证书。必要的信息在一个用分号分隔的键/值对字符串中指定。可以将证书放置在一个文件中并使用 file 键提供的文件名，或是将整个证书放置在一个字符串中，但是这两种方法不可同时使用。可以使用以下键：

键	缩写	说明
file		证书的文件名。
certificate	cert	证书本身。
company	co	证书中指定的公司。
unit		证书中指定的公司单元。
name		证书中指定的公用名。

只有被定向到 HTTPS 服务器的请求或可从非安全服务器重定向到安全服务器的请求才需要证书。

- **CLIENTPORT 子句** 标识 HTTP 客户端函数使用 TCP/IP 进行通信的端口号。该子句是为通过防火墙的连接提供的，并建议只用于此类连接，因为防火墙按照 TCP/UDP 端口进行过滤。您可以指定单个端口号、端口号范围或是两者的组合；例如 CLIENTPORT '85,90-97'。请参见“ClientPort 协议选项 [CPORT]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **PROXY 子句** 指定代理服务器的 URI。在客户端必须通过代理访问网络时使用。此子句指示函数将要连接到代理服务器，并通过它将请求发送到 Web 服务。

注释

CREATE FUNCTION 语句在数据库中创建 Web 服务函数。通过指定所有者名称，可以为其他用户创建函数。

SOAP 和 HTTP 函数的返回类型必须是某种字符数据类型（例如 VARCHAR）。返回值是 HTTP 响应的主体。其中不包括 HTTP 标头信息。如果需要详细信息（例如状态信息），请使用过程而非函数。

参数值将作为请求的一部分进行传递。使用的语法取决于请求的类型。对于 HTTP:GET，参数将作为 URL 的一部分进行传递；对于 HTTP:POST 请求，则将值放在请求主体中。SOAP 请求的参数总是被绑定在请求主体中。

权限

RESOURCE 特权。

用于外部函数（包括 Java 函数）的 DBA 权限。

副作用

自动提交。

另请参见

- “ALTER FUNCTION 语句”一节第 354 页
- “CREATE FUNCTION 语句”一节第 438 页
- “CREATE FUNCTION 语句（外部过程）”一节第 441 页
- “CREATE PROCEDURE 语句”一节第 458 页
- “CREATE PROCEDURE 语句（Web 服务）”一节第 471 页
- “DROP FUNCTION 语句”一节第 546 页
- “RETURN 语句”一节第 678 页
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “创建 Web 服务客户端函数和过程”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

以下语句创建名为 **cli_test1** 的函数，该函数用于从 localhost 上运行的 **get_picture** 服务中返回图像：

```
CREATE FUNCTION cli_test1( image LONG VARCHAR )
RETURNS LONG BINARY
URL 'http://localhost/get_picture'
TYPE 'HTTP:GET';
```

以下语句发出 URL 为 *http://localhost/get_picture?image=widget* 的 HTTP 请求：

```
SELECT cli_test1( 'widget' );
```

以下语句使用了替代参数，以允许将请求 URL 作为输入参数传递。SET 子句用于关闭 CHUNK 模式传输编码。

```
CREATE FUNCTION cli_test2( image LONG VARCHAR, myurl LONG VARCHAR )
RETURNS LONG BINARY
URL '!myurl'
TYPE 'HTTP:GET'
SET 'HTTP(CH=OFF)'
HEADER 'ASA-ID';
```

以下语句发出 URL 为 *http://localhost/get_picture?image=widget* 的 HTTP 请求：


```
CREATE VARIABLE a_binary LONG BINARY
a_binary = cli_test2('widget', 'http://localhost/get_picture');
SELECT a_binary;
```

CREATE INDEX 语句

此语句用于在指定表或实例化视图上创建索引。索引可以提高数据库的性能。

语法 1 - 在表上创建索引

CREATE [**VIRTUAL**] [**UNIQUE**] [**CLUSTERED**] **INDEX** *index-name*
ON [*owner.*]*table-name*
(*column-name* [**ASC** | **DESC**], ...
| *function-name* **(** *argument*, ...] **) AS** *column-name* **)**
[{ **IN** | **ON** } *dbspace-name*]
[**FOR OLAP WORKLOAD**]

语法 2 - 在实例化视图上创建索引

CREATE [**VIRTUAL**] [**UNIQUE**] [**CLUSTERED**] **INDEX** *index-name*
ON [*owner.*]*materialized-view-name*
(*column-name* [**ASC** | **DESC**], ...**)**
[{ **IN** | **ON** } *dbspace-name*]
[**FOR OLAP WORKLOAD**]

参数

- **VIRTUAL 子句** VIRTUAL 关键字主要由索引顾问使用。在索引顾问对执行计划进行求值期间以及使用 PLAN 函数时，虚拟索引将模拟真实物理索引的属性。可以将虚拟索引和 PLAN 函数结合使用以充分利用索引在性能方面的影响，从而避免创建真实索引这一通常既耗时又消耗资源的过程。

  虚拟索引对其它连接是不可见的，并且在关闭连接时将被删除。在对查询的实际执行计划进行求值时不使用虚拟索引，所以不会对性能造成影响。

  虚拟索引的列数限制为四列。

  请参见“获取对查询的 [索引顾问] 建议”一节 《SQL Anywhere 服务器 - SQL 的用法》和“索引顾问”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **CLUSTERED 子句** CLUSTERED 属性使行按照与索引对应的大概键顺序来存储。当数据库服务器尝试保留键顺序时，不能保证实施总体聚簇。

  如果存在聚簇索引，LOAD TABLE 语句便按照索引键的顺序插入行，而 INSERT 语句将按照键顺序的定义尝试将新行添加到包含相邻行的同一页上。

  请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **UNIQUE 子句** UNIQUE 属性确保表或实例化视图中不会有两行的值在所有索引列中相同。每个索引键都必须是唯一的，或者至少在一列中包含 NULL。

  唯一约束与唯一索引是有区别的。唯一索引的列可以为 NULL，而唯一约束的列不能为 NULL。外键可以引用主键或者唯一约束，但不能引用唯一索引，因为唯一索引可以包括多个 NULL 实例。

建议不要将近似数据类型（例如 FLOAT 和 DOUBLE）用于主键或具有唯一约束的列。近似数值数据类型在算术运算后容易产生舍入误差。

- **ASC | DESC 子句** 除非显式指定降序 (DESC)，否则列以升序 (increasing) 排序。无论索引是升序排列还是降序排列，在执行升序或降序 ORDER BY 操作时都会使用索引。但是，如果通过混合的升序和降序特性来执行 ORDER BY，则仅当索引是用相同的升序和降序特性创建的时，才会使用索引。

- **function-name** function-name 子句在函数上创建索引。此子句不能用于已声明的临时表或实例化视图。

  CREATE INDEX 语句的这种格式是执行以下操作的便捷方法：

  1. 向表中添加名为 *column-name* 的计算列。该列是通过 COMPUTE 子句（此子句是指定的函数）以及任何指定的参数定义的。有关可指定的函数类型的限制，请参见 CREATE TABLE 语句的 COMPUTE 子句。该列的数据类型依函数的返回类型而定。
  2. 填充表中现有行的计算列。
  3. 创建列的索引。

     删除索引不会导致相关联的计算列被删除。

     有关计算列的详细信息，请参见“使用计算列”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **IN | ON 子句** 缺省情况下，索引与其所在的表或实例化视图放在同一个数据库文件中。通过指定用来存放索引的 dbspace 名称，可以将索引放在单独的数据库文件中。此功能的主要用途是：使大型数据库避开文件大小的限制，或者实现通过使用多个磁盘设备才能达到的性能改善。

  如果新索引可与现有逻辑索引共用物理索引，将忽略 IN 子句。

  有关限制的详细信息，请参见“SQL Anywhere 大小和数量限制”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **FOR OLAP WORKLOAD 子句** 当指定 FOR OLAP WORKLOAD 时，数据库服务器会执行某些优化，并会收集有关该键的统计信息以帮助提高 OLAP 负载的性能。当 optimization_workload 设置为 OLAP 时，性能改善最为显著。请参见“optimization_workload 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  有关 OLAP 的详细信息，请参见“OLAP 支持” 《SQL Anywhere 服务器 - SQL 的用法》。

注释

语法 1 与表一起使用；语法 2 与实例化视图一起使用。

SQL Anywhere 使用物理索引和逻辑索引。物理索引是存储在磁盘上的实际索引结构。逻辑索引是对物理索引的引用。如果所创建索引的物理特性与现有索引相同，则数据库服务器会创建一个共享现有物理索引的逻辑索引。通常，会将您所创建的索引视为逻辑索引。数据库服务器可根据需要创建物理索引以实现逻辑索引，并可在多个逻辑索引之间共享同一物理索引。请参见“使用逻辑索引共享索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

CREATE INDEX 语句在指定表或实例化视图的指定列上创建排序索引。此索引自动用于改进向数据库发出的查询的性能，以及通过 ORDER BY 子句对查询进行排序。索引一旦创建，就不能再在

SQL 语句中加以引用（除非对其进行校验 (VALIDATE INDEX)、变更 (ALTER INDEX)、删除 (DROP INDEX)），而且也不能在优化程序的提示中进行引用。

- **索引所有权** 无法在 CREATE INDEX 语句中指定索引所有者。索引总是由表或实例化视图的所有者拥有。
- **视图索引** 可在实例化视图上创建索引，但无法在非常规视图上创建。
- **索引名称空间** 对于给定表或实例化视图，每个索引的名称必须唯一。
- **独占使用** 只要 CREATE INDEX 语句影响了当前正由其它连接使用的表或实例化视图，就会禁止该语句。CREATE INDEX 可能很耗时，且数据库服务器在处理此语句时不会处理引用同一个表的请求。
- **自动创建的索引** SQL Anywhere 自动为主键、外键和唯一约束创建索引。这些自动创建的索引保存在表所在的同一数据库文件中。

当存在使用 WITH HOLD 子句打开的使用语句快照或事务快照的游标时，不能执行此语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

实例化视图须是表或实例化视图的所有者，或者具有 DBA 权限或 REFERENCES 权限。

副作用

自动提交。在内置函数上创建索引还会生成一个检查点。

更新列统计信息（如果此信息不存在，则创建它）。

另请参见

- “DROP INDEX 语句”一节第 547 页
- “索引”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE STATISTICS 语句”一节第 491 页
- “使用逻辑索引共享索引”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “索引”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在 Employees 表上创建一个两列索引。

```
CREATE INDEX employee_name_index
ON Employees
( Surname, GivenName );
```

在 SalesOrderItems 表上为 ProductID 列创建索引。

```
CREATE INDEX item_prod
ON SalesOrderItems
( ProductID );
```

使用 SORTKEY 函数在 Products 表的 Description 列上创建索引，其排序顺序根据俄语归类而定。该语句有一个副作用，即向表中添加一个计算列 desc_ru。

```
CREATE INDEX ix_desc_ru
ON Products (
 SORTKEY( Description, 'rusdict' )
 AS desc_ru );
```

CREATE LOCAL TEMPORARY TABLE 语句

可在过程内使用此语句来创建局部临时表，而此表将在过程完成后一直保留到它被显式删除或连接终止。

语法

CREATE LOCAL TEMPORARY TABLE *table-name*
({ *column-definition* [*column-constraint* ...] | *table-constraint* | *pctfree* }, ... **)**
[**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS** | **NOT TRANSACTIONAL**]

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

参数

有关 *column-definition*、*column-constraint*、*table-constraint* 和 *pctfree* 的定义，请参见“CREATE TABLE 语句”一节第 497 页。

- **ON COMMIT 子句** 缺省情况下，临时表的行在执行 COMMIT 时删除。可以使用 ON COMMIT 子句在执行 COMMIT 时保留行。
- **NOT TRANSACTIONAL 子句** 在某些情况下，NOT TRANSACTIONAL 子句可以提高性能，因为对非事务性临时表执行的操作不会记入回退日志中。例如，在反复调用使用临时表的过程而不对 COMMIT 或 ROLLBACK 进行干预时，NOT TRANSACTIONAL 可能会非常有用。

注释

在过程中，如果要创建一个在过程完成后仍然保留的表，可使用 CREATE LOCAL TEMPORARY TABLE 语句，而不是 DECLARE LOCAL TEMPORARY TABLE 语句。使用 CREATE LOCAL TEMPORARY TABLE 语句创建的局部临时表会一直保留到它被显式删除或连接终止。

利用使用 CREATE LOCAL TEMPORARY TABLE 的 IF 语句创建的局部临时表，在 IF 语句完成后也会继续保留。

权限

无。

副作用

无。

另请参见

- “CREATE TABLE 语句”一节第 497 页
- “DECLARE LOCAL TEMPORARY TABLE 语句”一节第 529 页
- “使用复合语句”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL/基础特性。

示例

以下示例将创建一个名为 TempTab 的本地临时表：

```
CREATE LOCAL TEMPORARY TABLE TempTab ( number INT )
ON COMMIT PRESERVE ROWS;
```

CREATE LOGIN POLICY 语句

此语句用于创建登录策略。

语法

CREATE LOGIN POLICY *policy-name policy-options*

policy options :
policy-option [*policy-option ...*]

policy-option :
policy-option-name **=** *policy-option-value*

policy-option-value :
{ **UNLIMITED** | *legal-option-value* }

参数

- **policy_name** 登录策略的名称。
- **policy_option_name** 登录策略选项的名称。如果不指定任何选项，则应用从根登录策略获得的值。
- **policy_option_value** 指派给登录策略选项的值。如果指定 UNLIMITED，则不施加任何限制。要查看缺省登录策略值列表，请参见注释。

注释

如果不指定任何策略选项，则采用从根登录策略获得的登录策略值。

所有新数据库都包含根登录策略。可以修改根登录策略的值，但不能删除该策略。下表概述了根登录策略的缺省选项。

Policy_option_name	说明	缺省值	适用于
password_life_time	必须更改口令之前的最大天数。	Unlimited	所有用户，包括具有 DBA 权限的用户
password_grace_time	口令到期之前的天数，这段时间内允许登录但缺省 post_login 过程会发出警告。	0	所有用户，包括具有 DBA 权限的用户
password_expiry_on_next_login	如果此选项的值为 ON，则下次登录后用户口令到期。	OFF	所有用户，包括具有 DBA 权限的用户
locked	如果此选项的值为 ON，则禁止用户建立新连接。	OFF	只针对没有 DBA 权限的用户
max_connections	用户允许的最大并发连接数。	Unlimited	只针对没有 DBA 权限的用户
max_failed_login_attempts	从上次成功尝试开始，帐户锁定之前尝试登录到用户帐户的最多失败次数。	Unlimited	只针对没有 DBA 权限的用户
max_days_since_login	同一用户在两次连续登录之间可以经过的最大天数。	Unlimited	只针对没有 DBA 权限的用户
max_non_dba_connections	没有 DBA 权限的用户可进行的并发连接的最大数量。只在 DEFAULT 登录策略中支持此选项。	Unlimited	只针对没有 DBA 权限的用户。只针对缺省登录策略。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “ALTER LOGIN POLICY 语句”一节第 357 页
- “ALTER USER 语句”一节第 385 页
- “COMMENT 语句”一节第 406 页
- “CREATE USER 语句”一节第 518 页
- “DROP LOGIN POLICY 语句”一节第 548 页
- “DROP USER 语句”一节第 562 页
- “管理登录策略概述”一节 《SQL Anywhere 服务器 - 数据库管理》
- “创建新登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》
- “为现有用户分配登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下例创建了 Test1 登录策略。此示例具有不受限的口令有效期，允许用户在帐户锁定前最多尝试 5 次来输入正确的口令。

```
CREATE LOGIN POLICY Test1
password_life_time=UNLIMITED
max_failed_login_attempts=5;
```

CREATE MATERIALIZED VIEW 语句

此语句用于创建实例化视图。

语法

CREATE MATERIALIZED VIEW
[*owner.*]*materialized-view-name* [**(** *alt-column-names*, ... **)**]
[**IN** *dbspace-name*]
AS *select-statement*
[**CHECK** { **IMMEDIATE** | **MANUAL** } **REFRESH**]

alt-column-names :
(*column-name* [,...] **)**

参数

- **alt-column-names** 此子句用于指定实例化视图中列的替代名称。如果指定替代列名称，则 *alt-column-names* 中列出的列数必须与 *select-statement* 中的列数相匹配。如果不指定替代列名称，则其名称设为 *select-statement* 中的名称。
- **IN 子句** 此子句用于指定要在其中创建实例化视图的 dbspace。如果不指定此子句，则会在 default_dbspace 选项指定的 dbspace 中创建实例化视图。否则，使用系统 dbspace。有关详细信息，请参见“使用附加 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **AS 子句** 此子句采用 SELECT 语句的格式，指定用于填充实例化视图的数据。实例化视图定义只能引用基表，不能引用视图、其它实例化视图或临时表。*select-statement* 必须包含列名或

具有指定的别名。如果指定 *alt-column-names*，则使用这些名称，而不使用 *select-statement* 中指定的别名。

必须显式指定 SELECT 语句中的列名，不能使用 `SELECT *` 结构。例如，不能指定 `CREATE MATERIALIZED VIEW matview AS SELECT * FROM table-name`。另外还应完全限定 *select-statement* 中的对象名称。请参见“实例化视图的限制”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **CHECK 子句** 使用此子句，不实际创建视图即可校验语句。指定 CHECK 子句时：
  - 数据库服务器执行常规语言检查（不使用此子句执行 CREATE MATERIALIZED VIEW 会执行常规语言检查），并按平常的方式返回生成的所有错误。
  - 数据库服务器不实际创建视图。这意味着不会生成某些创建时出现的错误。例如，不会生成表示指定的视图名称已存在这样的错误。这样您就可以使用 CHECK 子句测试对视图定义的预期更改，而不会产生视图命名冲突。
  - 如果使用 CHECK IMMEDIATE REFRESH，则数据库服务器校验语法是否对立即视图有效，并返回所有错误。
  - 数据库不会发生任何更改，事务日志中不记录任何内容。
  - 语句开始执行时有一个隐式提交，结束时有一个回退，释放执行过程中获取的所有锁。

注释

创建实例化视图时，该视图为手动视图且未初始化。即视图具有手动刷新类型，但尚未刷新（用数据填充）。要初始化视图，请执行 REFRESH MATERIALIZED VIEW 语句或使用 sa_refresh_materialized_views 系统过程。请参见“REFRESH MATERIALIZED VIEW 语句”一节 第 665 页和“sa_refresh_materialized_views 系统过程”一节第 878 页。

可以加密实例化视图、更改其 PCTFREE 设置、更改其刷新类型，以及通过优化程序启用或禁用该视图的使用。但是，必须先创建实例化视图，然后才能使用 ALTER MATERIALIZED VIEW 更改这些设置。实例化视图创建时的缺省值为：

- NOT ENCRYPTED
- ENABLE USE IN OPTIMIZATION
- 根据数据库页面大小设置 PCTFREE：页面大小为 4 KB，则设为 200 个字节；页面大小为 2 KB，则设为 100 个字节。
- MANUAL REFRESH

创建实例化视图时，有几个数据库和服务器选项必须有效。请参见“实例化视图的限制”一节 《SQL Anywhere 服务器 - SQL 的用法》。

sa_recompile_views 系统过程不影响实例化视图。

权限

必须具有 RESOURCE 权限和实例化视图定义中的表的 SELECT 权限。要为其他用户创建实例化视图，还必须具有 DBA 权限。

副作用

执行过程中，CREATE MATERIALIZED VIEW 语句会对实例化视图所引用的所有表执行独占锁定，而不会阻塞。如果有一个引用表无法锁定，则语句执行失败并会返回错误。

另请参见

- “使用实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “实例化视图状态和属性”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “ALTER MATERIALIZED VIEW 语句”一节第 358 页
- “DROP MATERIALIZED VIEW 语句”一节第 549 页
- “REFRESH MATERIALIZED VIEW 语句”一节第 665 页
- “CREATE VIEW 语句”一节第 520 页
- “sa_refresh_materialized_views 系统过程”一节第 878 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例会创建实例化视图，该视图包含 SQL Anywhere 示例数据库中关于雇员的保密信息。随后必须执行 REFRESH MATERIALIZED VIEW 语句，才能初始化视图以供使用，如下例所示。

```
CREATE MATERIALIZED VIEW EmployeeConfid2 AS
SELECT EmployeeID, Employees.DepartmentID,
    SocialSecurityNumber, Salary, ManagerID,
    Departments.DepartmentName, Departments.DepartmentHeadID
FROM Employees, Departments
WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid2;
```

CREATE MESSAGE 语句 [T-SQL]

此语句用于将用户定义的消息添加到 ISYSUSERMESSAGE 系统表，以供 PRINT 和 RAISERROR 语句使用。

语法

CREATE MESSAGE *message-number* **AS** *message-text*

message-number : *integer*

message-text : *string*

参数

- **message_number** 要添加的消息的消息号。用户定义消息的消息号必须为 20000 或更大。
- **message_text** 要添加的消息文本。最大长度为 255 个字节。PRINT 和 RAISERROR 识别消息文本中的占位符。一条消息可以按任何顺序包含多达 20 个不同的占位符。将消息文本发送到客户端时，消息后面的任何参数的格式化内容将替换这些占位符。

将消息翻译成一种具有不同语法结构的语言时，会对占位符进行编号以允许对参数进行重新排序。参数的占位符以 "%nn!" 的形式出现：百分号 (%)，后跟一个 1 到 20 之间的整数，然后是一个感叹号 (!)，其中的整数代表参数在参数列表中的位置。"%1!" 是第一个参数，"%2!" 是第二个参数，依此类推。

没有与 sp_addmessage 的 *language* 参数相对应的参数。

注释

CREATE MESSAGE 将消息号与消息字符串关联。在 PRINT 和 RAISERROR 语句中可以使用消息号。

要删除消息，请参见“DROP MESSAGE 语句”一节第 549 页。

权限

必须有 RESOURCE 权限。

副作用

自动提交。

另请参见

- “PRINT 语句 [T-SQL]”一节第 659 页
- “RAISERROR 语句”一节第 661 页
- “ISYSUSERMESSAGE 系统表”一节第 765 页

标准和兼容性

- **SQL/2003** 服务商扩充。

CREATE PROCEDURE 语句

使用此语句在数据库中创建用户定义的 SQL 过程。要创建外部过程接口，请参见“CREATE PROCEDURE 语句（外部过程）”一节第 465 页。要创建 Web 服务过程，请参见“CREATE PROCEDURE 语句（Web 服务）”一节第 471 页。

语法

CREATE [**OR REPLACE** | **TEMPORARY**] **PROCEDURE** [*owner*.]*procedure-name*
([*parameter*, ...] **)**
[**RESULT (** *result-column*, ... **)** | **NO RESULT SET**]
[**SQL SECURITY** { **INVOKER** | **DEFINER** }]
[**ON EXCEPTION RESUME**]
compound-statement | **AT** *location-string*

parameter :
  parameter-mode parameter-name data-type [**DEFAULT** *expression*]
| **SQLCODE**
| **SQLSTATE**

parameter-mode : **IN**
| **OUT**
| **INOUT**

result-column : *column-name data-type*

参数

- **CREATE PROCEDURE** 可以创建永久或临时 (TEMPORARY) 存储过程。可使用 PROC 作为 PROCEDURE 的同义词。

  参数名必须符合其它数据库标识符（如列名）的规则。它们必须是有效的 SQL 数据类型。有关有效数据类型的列表，请参见“SQL 数据类型”第 75 页。

  参数可以使用关键字 IN、OUT 或 INOUT 作为前缀。如果未指定上述任何值，则缺省情况下将使用 INOUT 参数。这些关键字具有以下含义：

  - **IN** 此参数是一个为过程提供值的表达式。
  - **OUT** 此参数是一个可由过程赋值的变量。
  - **INOUT** 此参数是一个为过程提供值的变量，并且可由过程赋值。

  使用 CALL 语句执行过程时，不需要指定所有参数。如果在 CREATE PROCEDURE 语句中提供了缺省值，缺少的参数会被分配缺省值。如果 CALL 语句中既未提供参数也未设置缺省值，则会给出错误。

  SQLSTATE 和 SQLCODE 是特殊 OUT 参数，它们在过程结束时输出 SQLSTATE 或 SQLCODE 值。在过程调用后可立即检查 SQLSTATE 和 SQLCODE 特殊值，以测试过程的返回状态。

  SQLSTATE 和 SQLCODE 特殊值会由下一个 SQL 语句修改。如果将 SQLSTATE 或 SQLCODE 作为过程参数提供，则会允许返回代码存储在变量中。

  指定 CREATE OR REPLACE PROCEDURE 将创建一个新过程或替换同名的现有过程。此子句将更改过程的定义，但保留现有权限。不能将 OR REPLACE 子句与临时过程一起使用。或者，如果正在替换的过程已经使用，则会返回错误。

  如果指定 CREATE TEMPORARY PROCEDURE，则会意味着该存储过程仅对创建它的连接可见，并在删除该连接时随之自动删除。也可以显式删除临时存储过程。您无法在临时存储过程上执行 ALTER、GRANT 或 REVOKE，而且与其它存储过程不同，临时存储过程不会在目录或事务日志中予以记录。

  临时过程使用其创建者（当前用户）或指定所有者的权限执行。可在以下情况下为临时过程指定所有者：

  - 临时过程是在永久存储过程中创建的
  - 临时过程与永久过程的所有者相同

  要删除临时过程的所有者，必须首先删除该临时过程。

  临时存储过程可在连接到只读数据库时加以创建和删除，且不能为外部过程。

  例如，以下临时存储过程会删除名为 CustRank 的表（如果此表存在）。对于此示例，该过程假定表名是唯一的，且可由过程创建者引用而不必指定表所有者：

```
CREATE TEMPORARY PROCEDURE drop_table( IN @TableName char(128) )
BEGIN
    IF EXISTS  ( SELECT * FROM SYS.SYSTAB WHERE table_name = @TableName )
THEN
   EXECUTE IMMEDIATE 'DROP TABLE "' || @TableName || '"';
   MESSAGE 'Table "' || @TableName || '" dropped' to client;
    END IF;
END;
CALL drop_table( 'CustRank' );
```

- **RESULT 子句**　RESULT 子句声明结果集中的列的数量和类型。RESULT 关键字后面括在括号内的列表定义结果的列名和类型。描述 CALL 语句时，嵌入式 SQL DESCRIBE 或 ODBC SQLDescribeCol 会返回此信息。有关有效数据类型的列表，请参见“SQL 数据类型”第 75 页。

  有关从过程返回结果集的详细信息，请参见“从过程返回结果”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  视执行方式而定，某些过程可产生多个结果集，并且列数也不同。例如，以下过程在有些情况下会返回两列，而在有些情况下则会返回一列。

  ```
  CREATE PROCEDURE names( IN formal char(1))
  BEGIN
     IF formal = 'n' THEN
        SELECT GivenName
        FROM Employees
     ELSE
        SELECT Surname, GivenName
        FROM Employees
     END IF
  END;
  ```

  包含可变结果集的过程必须用不带 RESULT 子句的语句编写，或者用 Transact-SQL 编写。它们的使用受以下限制制约：

  - **嵌入式 SQL**　必须在打开用于结果集的游标之后、返回任何行之前对过程调用执行 DESCRIBE，才能获取正确形式的结果集。这需要使用 DESCRIBE 语句的 CURSOR *cursor-name* 子句。
  - **ODBC、OLE DB、ADO.NET**　使用这些接口的应用程序可以使用可变结果集过程。结果集的正确描述由驱动程序或提供程序完成。
  - **Open Client 应用程序**　Open Client 应用程序可以使用可变结果集过程。

  如果过程仅返回一个结果集，则应使用 RESULT 子句。有了这个子句，便可防止 ODBC 和 Open Client 应用程序在游标打开后重新描述结果集。

  要处理多个结果集，ODBC 必须描述当前正在执行的游标，而不是过程的已定义结果集。因此，ODBC 不会始终按过程定义的 RESULT 子句中的定义来描述列名。为避免这种问题，请在生成结果集的 SELECT 语句中使用列的别名。

- **NO RESULT SET 子句**　声明此过程不返回结果集。当外部环境需要知道某个过程不返回结果集时，这将非常有用。
- **SQL SECURITY 子句**　SQL SECURITY 子句定义该过程是作为 INVOKER（调用该过程的用户）执行还是作为 DEFINER（拥有该过程的用户）执行。缺省值为 DEFINER。

指定 SQL SECURITY INVOKER 后，必须对每个调用该过程的用户加以标注，因此会使用更多内存。另外，指定 SQL SECURITY INVOKER 后，也会作为调用者进行名称解析。因此，应注意用适合的所有者限定所有对象名称（表、过程等）。例如，假定 user1 创建了以下过程：

```
CREATE PROCEDURE user1.myProcedure()
   RESULT( columnA INT )
   SQL SECURITY INVOKER
   BEGIN
     SELECT columnA FROM table1;
   END;
```

如果 user2 试图运行此过程，而表 user2.table1 *不*存在，则会产生表查寻错误。另外，如果 user2.table1 *存*在，则使用该表而不使用预定的 user1.table1。为了防止出现这种情况，请在语句中限定表引用（user1.table1，而不只是 table1）。

- **ON EXCEPTION RESUME 子句**　该子句使类似 Transact-SQL 的错误处理能够在 Watcom-SQL 语法过程中使用。

  如果使用 ON EXCEPTION RESUME，则过程会根据 on_tsql_error 选项的设置来执行操作。如果 on_tsql_error 设置为 Conditional（缺省值），则会在下一条语句能处理错误时继续执行该语句；否则，便会退出。

  错误处理语句包括以下这些：

  - IF
  - SELECT @variable =
  - CASE
  - LOOP
  - LEAVE
  - CONTINUE
  - CALL
  - EXECUTE
  - SIGNAL
  - RESIGNAL
  - DECLARE
  - SET VARIABLE

  在 ON EXCEPTION RESUME 子句中不要使用显式错误处理代码。

  请参见“on_tsql_error 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **AT *location-string* 子句**　在当前数据库中为 *location-string* 指定的远程过程创建代理存储过程。AT 子句支持分号 (;) 作为 *location-string* 中的字段分隔符。如果没有分号，则将句号用作字段分隔符。这样一来，便可在数据库和所有者字段中使用文件名和扩展名。

  远程过程接受长度最多 254 个字节的输入参数，并在输出变量中返回最多 254 个字符。

  如果远程过程可以返回结果集，即使并不是在所有情况下都返回，本地过程定义也必须包含 RESULT 子句。

  有关远程服务器的信息，请参见“CREATE SERVER 语句”一节第 481 页。有关使用远程过程的信息，请参见“使用远程过程调用 (RPC)”一节 《SQL Anywhere 服务器 - SQL 的用法》。

注释

CREATE PROCEDURE 语句在数据库中创建过程。具有 DBA 权限的用户可以通过指定所有者为其他用户创建过程。过程可用 CALL 语句进行调用。

如果存储过程返回一个结果集，则它不能同时设置输出参数或返回一个返回值。

从多个过程引用临时表时，如果该临时表定义不一致且高速缓存引用该表的语句，则会出现潜在问题。请参见“在过程中引用临时表”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

除非创建临时过程，否则必须具有 RESOURCE 权限。

引用外部过程或者为其他用户创建过程必须有 DBA 权限。

副作用

自动提交。

另请参见

- “ALTER PROCEDURE 语句”一节第 361 页
- “BEGIN 语句”一节第 395 页
- “CALL 语句”一节第 400 页
- “CREATE FUNCTION 语句”一节第 438 页
- “CREATE PROCEDURE 语句 [T-SQL]”一节第 464 页
- “DROP PROCEDURE 语句”一节第 550 页
- “EXECUTE IMMEDIATE 语句 [SP]”一节第 570 页
- “GRANT 语句”一节第 595 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 持久存储模块特性。Java 结果集的语法扩展如可选的 J621 特性中所指定。

示例

下面的过程使用 case 语句对查询结果归类。

```
CREATE PROCEDURE ProductType (IN product_ID INT, OUT type CHAR(10))
BEGIN
   DECLARE prod_name CHAR(20);
   SELECT name INTO prod_name FROM Products
   WHERE ID = product_ID;
   CASE prod_name
   WHEN 'Tee Shirt' THEN
      SET type = 'Shirt'
   WHEN 'Sweatshirt' THEN
      SET type = 'Shirt'
   WHEN 'Baseball Cap' THEN
      SET type = 'Hat'
   WHEN 'Visor' THEN
      SET type = 'Hat'
   WHEN 'Shorts' THEN
      SET type = 'Shorts'
   ELSE
      SET type = 'UNKNOWN'
```



```
   END CASE;
END;
```

以下示例将替换在上一个示例中创建的 ProductType 过程。在代替该过程后，将更新 Tee Shirt 和 Sweatshirt 的参数：

```
CREATE OR REPLACE PROCEDURE ProductType (IN product_ID INT, OUT type
CHAR(10))
BEGIN
   DECLARE prod_name CHAR(20);
   SELECT name INTO prod_name FROM Products
   WHERE ID = product_ID;
   CASE prod_name
   WHEN 'Tee Shirt' THEN
      SET type = 'T Shirt'
   WHEN 'Sweatshirt' THEN
      SET type = 'Long Sleeve Shirt'
   WHEN 'Baseball Cap' THEN
      SET type = 'Hat'
   WHEN 'Visor' THEN
      SET type = 'Hat'
   WHEN 'Shorts' THEN
      SET type = 'Shorts'
   ELSE
      SET type = 'UNKNOWN'
   END CASE;
END;
```

以下过程使用游标并遍历游标各行，返回单个值。

```
CREATE PROCEDURE TopCustomer (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
   DECLARE err_notfound EXCEPTION
   FOR SQLSTATE '02000';
   DECLARE curThisCust CURSOR FOR
      SELECT CompanyName,
          CAST(SUM(SalesOrderItems.Quantity *
          Products.UnitPrice) AS INTEGER) VALUE
      FROM Customers
      LEFT OUTER JOIN SalesOrders
      LEFT OUTER JOIN SalesOrderItems
      LEFT OUTER JOIN Products
      GROUP BY CompanyName;
   DECLARE ThisValue INT;
   DECLARE ThisCompany CHAR(35);
   SET TopValue = 0;
   OPEN curThisCust;
   CustomerLoop:
   LOOP
      FETCH NEXT curThisCust
      INTO ThisCompany, ThisValue;
      IF SQLSTATE = err_notfound THEN
         LEAVE CustomerLoop;
      END IF;
      IF ThisValue > TopValue THEN
         SET TopValue = ThisValue;
         SET TopCompany = ThisCompany;
         END IF;
   END LOOP CustomerLoop;
   CLOSE curThisCust;
END;
```

CREATE PROCEDURE 语句 [T-SQL]

此语句用于在数据库中创建新过程，采用与 Adaptive Server Enterprise 兼容的方式。

语法 1

SQL Anywhere 支持 Transact-SQL CREATE PROCEDURE 语句的以下子集。

CREATE PROCEDURE [*owner.*]*procedure_name*
[**NO RESULT SET**]
[[**(**] *@parameter_name data-type* [= *default*] [**OUTPUT**], ... [**)**]]
[**WITH RECOMPILE**] **AS** *statement-list*

参数

- **NO RESULT SET 子句** 声明此过程不返回结果集。当外部环境需要知道某个过程不返回结果集时，这将非常有用。

注释

以下列出 Transact-SQL 与 SQL Anywhere 语句 (Watcom-SQL) 之间的差异，以帮助那些使用这两种方言编写代码的人员。

- **带 @ 前缀的变量名** "@"符表示 Transact-SQL 变量名，而 Watcom-SQL 变量可以是任何有效的标识符，也可以带 @ 前缀。
- **输入和输出参数** Watcom-SQL 过程的参数在缺省情况下为 INOUT，也可指定为 IN、OUT 或 INOUT。Transact-SQL 过程的参数在缺省情况下为 INPUT，或者可以指定为 OUTPUT。这些在 SQL Anywhere 中声明为 INOUT 或 OUT 的参数在 Transact-SQL 中应声明为 OUTPUT。
- **参数缺省值** Watcom-SQL 使用关键字 DEFAULT 为过程参数提供缺省值，而 Transact-SQL 使用等号 (=) 提供缺省值。
- **返回结果集** Watcom-SQL 使用 RESULT 子句指定返回的结果集。在 Transact-SQL 过程中，第一个查询的列名或别名将返回到调用环境。

  下面的 Transact-SQL 过程阐释了结果集如何从 Transact-SQL 存储过程返回：

```
CREATE PROCEDURE showdept @deptname varchar(30)
AS
   SELECT Employees.Surname, Employees.GivenName
   FROM Departments, Employees
   WHERE Departments.DepartmentName = @deptname
   AND Departments.DepartmentID = Employees.DepartmentID;
```

  下面是相应的 Watcom-SQL 过程：

```
CREATE PROCEDURE showdept(in deptname
         varchar(30) )
RESULT ( lastname char(20), firstname char(20))
ON EXCEPTION RESUME
BEGIN
   SELECT Employees.Surname, Employees.GivenName
   FROM Departments, Employees
   WHERE Departments.DepartmentName = deptname
   AND Departments.DepartmentID = Employees.DepartmentID
END;
```


- **过程主体** Transact-SQL 过程的主体是一系列以关键字 AS 开始的 Transact-SQL 语句。Watcom-SQL 过程的主体是用关键字 BEGIN 和 END 括起来的复合语句。

权限

必须具有 RESOURCE 权限。

副作用

自动提交。

另请参见

- “CREATE FUNCTION 语句”一节第 438 页
- “CREATE PROCEDURE 语句”一节第 458 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。
- **Sybase** SQL Anywhere 支持 Adaptive Server Enterprise CREATE PROCEDURE 语句语法的子集。

  如果提供了 Transact-SQL WITH RECOMPILE 可选子句，它将被忽略。数据库启动后，当过程第一次执行时，SQL Anywhere 总是重新编译它们，并存储编译的过程直到数据库停止。

  不支持过程组。

CREATE PROCEDURE 语句（外部过程）

使用此语句创建一个连接本地或外部过程的接口。要创建 SQL 过程，请参见“CREATE PROCEDURE 语句”一节第 458 页。

语法

CREATE [**OR REPLACE**] **PROCEDURE** [*owner.*]*procedure-name*
  ([*parameter*, ...] **)**
[**RESULT (** *result-column*, ... **)** | **NO RESULT SET**]
[**DYNAMIC RESULT SETS** *integer-expression*]
[**SQL SECURITY** { **INVOKER** | **DEFINER** }]
EXTERNAL NAME '*external-call***'** [**LANGUAGE** *environment-name*]

parameter :
  [*parameter-mode*] *parameter-name data-type* [**DEFAULT** *expression*]
| **SQLCODE**
| **SQLSTATE**

parameter-mode : **IN**
| **OUT**
| **INOUT**

result-column : *column-name data-type*

environment-name :
C_ESQL32
| **C_ESQL64**
| **C_ODBC32**
| **C_ODBC64**
| **CLR**
| **JAVA**
| **PERL**
| **PHP**

参数

- **CREATE PROCEDURE** 您可以创建用于调用使用各种编程语言编写的外部或本地过程的永久存储过程。可使用 PROC 作为 PROCEDURE 的同义词。

  参数名必须符合其它数据库标识符（如列名）的规则。它们必须是有效的 SQL 数据类型。有关有效数据类型的列表，请参见“SQL 数据类型”第 75 页。

  参数可以使用关键字 IN、OUT 或 INOUT 作为前缀。如果未指定上述任何值，则缺省情况下将使用 INOUT 参数。这些关键字具有以下含义：

  - **IN** 此参数是一个为过程提供值的表达式。
  - **OUT** 此参数是一个可由过程赋值的变量。
  - **INOUT** 此参数是一个为过程提供值的变量，并且可由过程赋值。

  使用 CALL 语句执行过程时，并不需要指定所有参数。如果在 CREATE PROCEDURE 语句中提供了缺省值，则会为缺少的参数指派缺省值。如果 CALL 语句中未提供参数，且缺省值未设置，则会给出错误。

  SQLSTATE 和 SQLCODE 是特殊 OUT 参数，它们在过程结束时输出 SQLSTATE 或 SQLCODE 值。在过程调用后可立即检查 SQLSTATE 和 SQLCODE 特殊值，以测试过程的返回状态。

  SQLSTATE 和 SQLCODE 特殊值会由下一个 SQL 语句修改。如果将 SQLSTATE 或 SQLCODE 作为过程参数提供，则允许返回代码存储在变量中。

  指定 OR REPLACE (CREATE OR REPLACE PROCEDURE) 将创建一个新过程或替换同名的现有过程。此子句将更改过程的定义，但保留现有权限。如果尝试替换已使用的过程，则将返回错误。

  您无法创建 TEMPORARY 外部调用过程。

- **RESULT 子句** RESULT 子句声明结果集中的列的数量和类型。RESULT 关键字后面括在括号内的列表定义结果的列名和类型。描述 CALL 语句时，嵌入式 SQL DESCRIBE 或 ODBC SQLDescribeCol 会返回此信息。有关数据类型的列表，请参见“SQL 数据类型”第 75 页。

  调用嵌入式 SQL（LANGUAGE C_ESQL32、LANGUAGE C_ESQL64）或 ODBC（LANGUAGE C_ODBC32、LANGUAGE C_ODBC64）外部函数的过程可返回 0 或 1 个结果集。

  调用 Perl 或 PHP（LANGUAGE PERL、LANGUAGE PHP）外部函数的过程不能返回结果集。调用由数据库服务器装载的本地函数的过程不能返回结果集。

  调用 CLR 或 Java（LANGUAGE CLR、LANGUAGE JAVA）外部函数的过程可返回 0、1 或更多个结果集。

视执行方式而定，某些过程可产生多个结果集，并且列数也不同。例如，以下过程在有些情况下会返回两列，而在有些情况下则会返回一列。

```
CREATE PROCEDURE names( IN formal char(1))
BEGIN
   IF formal = 'n' THEN
      SELECT GivenName
      FROM Employees
   ELSE
      SELECT Surname, GivenName
      FROM Employees
   END IF
END;
```

包含可变结果集的过程必须用不带 RESULT 子句的语句编写，或者用 Transact-SQL 编写。它们的使用受以下限制制约：

- **嵌入式 SQL** 必须在打开用于结果集的游标之后、返回任何行之前对过程调用执行 DESCRIBE，才能获取正确形式的结果集。这需要使用 DESCRIBE 语句的 CURSOR *cursor-name* 子句。
- **ODBC、OLE DB、ADO.NET** 使用这些接口的应用程序可以使用可变结果集过程。结果集的正确描述由驱动程序或提供程序完成。
- **Open Client 应用程序** Open Client 应用程序可以使用可变结果集过程。

如果过程仅返回一个结果集，则应使用 RESULT 子句。有了这个子句，便可防止 ODBC 和 Open Client 应用程序在游标打开后重新描述结果集。

要处理多个结果集，ODBC 必须描述当前正在执行的游标，而不是过程的已定义结果集。因此，ODBC 并不总是按照过程定义的 RESULT 子句中的定义来描述列名称。为避免这种问题，请在生成结果集的 SELECT 语句中使用列的别名。

有关从过程返回结果集的详细信息，请参见“从过程返回结果”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **NO RESULT SET 子句** 声明此过程不返回结果集。此声明能够促进性能提高。
- **DYNAMIC RESULT SETS 子句** 将此子句与 LANGUAGE CLR 和 LANGUAGE JAVA 调用一起使用。如果不提供 DYNAMIC RESULT SETS 子句，则假定此方法不返回结果集。

  请注意，调用 Perl 或 PHP（LANGUAGE PERL、LANGUAGE PHP）外部函数的过程不能返回结果集。调用由数据库服务器装载的本地函数的过程不能返回结果集。
- **SQL SECURITY 子句** SQL SECURITY 子句定义该过程是作为 INVOKER（调用该过程的用户）执行还是作为 DEFINER（拥有该过程的用户）执行。缺省值为 DEFINER。对于外部调用，此子句用于为外部环境中的非限定对象引用建立所有权上下文。

  指定 SQL SECURITY INVOKER 后，必须对每个调用该过程的用户加以标注，因此会使用更多内存。另外，指定 SQL SECURITY INVOKER 后，也会作为调用者进行名称解析。因此，应注意用适合的所有者限定所有对象名称（表、过程等）。例如，假定 user1 创建了以下过程：

```
CREATE PROCEDURE user1.myProcedure()
   RESULT( columnA INT )
   SQL SECURITY INVOKER
   BEGIN
     SELECT columnA FROM table1;
   END;
```

如果 user2 试图运行此过程，而表 user2.table1 *不*存在，则会产生表查寻错误。另外，如果 user2.table1 *确实*存在，则使用该表而不使用预定的 user1.table1。为了防止出现这种情况，请在语句中限定表引用（user1.table1，而不只是 table1）。

- **EXTERNAL NAME '*native-call*' 子句**

  EXTERNAL NAME '*native-call*'

  native-call :
  [*operating-system*:]*function-name*@*library*; ...

  operating-system : **Unix**
  使用不带 LANGUAGE 属性的 EXTERNAL NAME 子句的过程用于定义一个连接本地函数（使用 C 语言等编程语言编写）的接口。该本地函数由数据库服务器装载到其地址空间中。

  library 名可包含文件扩展名，通常在 Windows 中为 *.dll*，在 Unix 中为 *.so*。在没有扩展名的情况下，该软件附加平台特定的缺省库文件扩展名。以下为一个正式示例。

  ```
  CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
  EXTERNAL NAME 'mystring@mylib.dll;Unix:mystring@mylib.so';
  ```

  下面是使用特定于平台的缺省值编写上述 EXTERNAL NAME 子句的更简单的方法：

  ```
  CREATE PROCEDURE mystring( IN instr LONG VARCHAR )
  EXTERNAL NAME 'mystring@mylib';
  ```

  如果调用，则包含此函数的库将被装载到数据库服务器的地址空间中。本地函数将作为服务器的一部分执行。在这种情况下，如果此函数导致故障，则会使数据库服务器终止。因此，建议在外部环境中装载和执行使用 LANGUAGE 属性的函数。如果函数在外部环境中导致故障，数据库服务器仍可继续运行。

  有关本地库调用的信息，请参见“从过程调用外部库”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME '*c-call*' LANGUAGE {C_ESQL32 | C_ESQL64 | C_ODBC32 | C_ODBC64 } 子句**

  EXTERNAL NAME '*c-call*' **LANGUAGE C_ESQL32**

  EXTERNAL NAME '*c-call*' **LANGUAGE C_ESQL64**

  EXTERNAL NAME '*c-call*' **LANGUAGE C_ODBC32**

  EXTERNAL NAME '*c-call*' **LANGUAGE C_ODBC64**

  c-call :
  [*operating-system*:]*function-name*@*library*; ...

  operating-system : **Unix**
  要在外部环境而非数据库服务器中调用编译后的本地 C 函数，存储过程或函数应使用后跟 LANGUAGE 属性的 EXTERNAL NAME 子句来定义，其中，LANGUAGE 属性指定 C_ESQL32、C_ESQL64、C_ODBC32 或 C_ODBC64。

  指定 LANGUAGE 属性后，包含此函数的库将由外部进程装载，外部函数将在该外部进程中执行。在这种情况下，如果此函数导致故障，数据库服务器仍可继续运行。

  以下是一个示例过程定义。


```
CREATE PROCEDURE ODBCinsert(
  IN ProductName CHAR(30),
  IN ProductDescription CHAR(50)
)
NO RESULT SET
EXTERNAL NAME 'ODBCexternalInsert@extodbc.dll'
LANGUAGE C_ODBC32;
```

有关详细信息，请参见“ESQL 和 ODBC 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *clr-call* LANGUAGE CLR 子句**

  EXTERNAL NAME '*clr-call*' LANGUAGE CLR

  clr-call :
  dll-name::*function-name*(param-type-1, ...)
  要在外部环境中调用 .NET 函数，过程接口应使用后跟 LANGUAGE CLR 属性的 EXTERNAL NAME 子句来定义。

  CLR 存储过程或函数的行为与 SQL 存储过程或函数的基本相同，只是过程或函数的代码以 C# 或 Visual Basic 等 .NET 语言编写并在数据库服务器外（即在单独的 .NET 可执行文件内）执行。

  以下是一个示例过程定义。

```
CREATE PROCEDURE clr_interface(
    IN p1 INT,
    IN p2 UNSIGNED SMALLINT,
    OUT p3 LONG VARCHAR)
NO RESULT SET
EXTERNAL NAME 'CLRlib.dll::CLRproc.Run( int, ushort, out string )'
LANGUAGE CLR;
```

  有关详细信息，请参见“CLR 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *perl-call* LANGUAGE PERL 子句**

  EXTERNAL NAME '*perl-call*' LANGUAGE PERL

  perl-call :
  <file=*perl-file***> $sa_perl_return =** *perl-sub***($sa_perl_arg0**, ... **)**
  要在外部环境中调用 Perl 函数，过程接口应使用后跟 LANGUAGE PERL 属性的 EXTERNAL NAME 子句来定义。

  Perl 存储过程或函数的行为与 SQL 存储过程或函数基本相同，只是过程或函数的代码以 Perl 编写并在数据库服务器外（即在 Perl 可执行实例内）执行。

  以下是一个示例过程定义。

```
CREATE PROCEDURE PerlWriteToConsole( IN str LONG VARCHAR)
NO RESULT SET
EXTERNAL NAME '<file=PerlConsoleExample>
    WriteToServerConsole( $sa_perl_arg0 )'
LANGUAGE PERL;
```

  有关详细信息，请参见“PERL 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *php-call* LANGUAGE PHP 子句**

  EXTERNAL NAME '*php-call*' LANGUAGE PHP

php-call :
<file=*php-file***> print** *php-func***($argv[1]**, ... **)**
要在外部环境中调用 PHP 函数，过程接口应使用后跟 LANGUAGE PHP 属性的 EXTERNAL NAME 子句来定义。

PHP 存储过程或函数的行为与 SQL 存储过程或函数基本相同，只是过程或函数的代码以 PHP 编写并且在数据库服务器外（即在 PHP 可执行实例内）执行。

以下是一个示例过程定义。

```
CREATE PROCEDURE PHPPopulateTable()
NO RESULT SET
EXTERNAL NAME '<file=ServerSidePHPExample> ServerSidePHPSub()'
LANGUAGE PHP;
```

有关详细信息，请参见“PHP 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

- **EXTERNAL NAME *java-call* LANGUAGE JAVA 子句**

  EXTERNAL NAME '*java-call***' LANGUAGE JAVA**

  java-call :
  [*package-name*.]*class-name.method-name method-signature*

  method-signature :
  ([*field-descriptor*, ...] **)** *return-descriptor*

  field-descriptor and *return-descriptor* :
  Z
  | **B**
  | **S**
  | **I**
  | **J**
  | **F**
  | **D**
  | **C**
  | **V**
  | **[***descriptor*
  | **L***class-name***;**
  要在外部环境中调用 Java 方法，过程接口应使用后跟 LANGUAGE JAVA 属性的 EXTERNAL NAME 子句来定义。

  采用 Java 接口技术的存储过程或函数的行为与 SQL 存储过程或函数的行为基本相同，只是过程或函数的代码以 Java 编写并且在数据库服务器外（即在 Java 虚拟机内）执行。

  以下是一个示例过程定义。

  ```
  CREATE PROCEDURE HelloDemo( IN name LONG VARCHAR )
  NO RESULT SET
  EXTERNAL NAME 'Hello.main([Ljava/lang/String;)V'
  LANGUAGE JAVA;
  ```

  有关详细信息，请参见“Java 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

注释

CREATE PROCEDURE 语句在数据库中创建过程。具有 DBA 权限的用户可以通过指定所有者为其他用户创建过程。过程可用 CALL 语句进行调用。

如果存储过程返回一个结果集，则它不能同时设置输出参数或返回一个返回值。

从多个过程引用临时表时，如果该临时表定义不一致且高速缓存引用该表的语句，则会出现潜在问题。请参见“在过程中引用临时表”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

除非创建临时过程，否则必须具有 RESOURCE 权限。

引用外部过程或者为其他用户创建过程必须有 DBA 权限。

副作用

自动提交。

另请参见

- “ALTER PROCEDURE 语句”一节第 361 页
- “CALL 语句”一节第 400 页
- “CREATE FUNCTION 语句”一节第 438 页
- “CREATE FUNCTION 语句（外部过程）”一节第 441 页
- “CREATE PROCEDURE 语句”一节第 458 页
- “DROP PROCEDURE 语句”一节第 550 页
- “GRANT 语句”一节第 595 页
- “外部环境概述”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 持久存储模块特性。Java 结果集的语法扩展如可选的 J621 特性中所指定。

CREATE PROCEDURE 语句（Web 服务）

此语句用于创建发出 HTTP 或 SOAP 请求的 Web 服务客户端过程。要创建 SQL 过程，请参见“CREATE PROCEDURE 语句”一节第 458 页。

CREATE [**OR REPLACE**] **PROCEDURE** [*owner*.]*procedure-name* **(** [*parameter*, ...] **)**
URL *url-string*
[**TYPE** { *http-type-spec-string* | *soap-type-spec-string* }]
[**HEADER** *header-string*]
[**CERTIFICATE** *certificate-string*]
[**CLIENTPORT** *clientport-string*]
[**PROXY** *proxy-string*]
[**SET** *protocol-option-string*]
[**SOAPHEADER** *soap-header-string*]
[**NAMESPACE** *namespace-string*]

http-type-spec-string :
HTTP[**:** { **GET**
| **POST**[**:***MIME-type*]
| **PUT**[**:***MIME-type*]
| **DELETE**
| **HEAD** }]

soap-type-spec-string :
SOAP[:{ **RPC** | **DOC** }

parameter :
parameter-mode parameter-name data-type [**DEFAULT** *expression*]

parameter-mode :
IN
| **OUT**
| **INOUT**

url-string :
{ **HTTP** | **HTTPS** | **HTTPS_FIPS** }**://**[*user*:*password***@**]*hostname*[**:***port*][**/***path*]

protocol-option-string
[*http-option-list*]
[**,** *soap-option-list*]

http-option-list :
HTTP(
[**CH**[**UNK**]={ **ON** | **OFF** | **AUTO** }]
[**;** **VER**[**SION**]={ **1.0** | **1.1** }]
)

soap-option-list:
SOAP(OP[**ERATION**]=*soap-operation-name***)**

参数

- **CREATE PROCEDURE** 您可创建或替换 Web 服务客户端过程。可使用 PROC 作为 PROCEDURE 的同义词。

  对于 SOAP 请求，缺省情况下，将过程名用作 SOAP 操作名。有关详细信息，请参见下面的 SET 子句。

  参数名必须符合其它数据库标识符（如列名）的规则。它们必须是有效的 SQL 数据类型。有关有效数据类型的列表，请参见“SQL 数据类型”第 75 页。仅 SOAP 请求支持传输划分了类型的数据，如 FLOAT、INT 等等。HTTP 请求仅支持传输字符串，因此您只能使用 CHAR 类型。有关支持的 SOAP 类型的详细信息，请参见“使用数据类型”一节 《SQL Anywhere 服务器 - 编程》和“使用结构化数据类型”一节 《SQL Anywhere 服务器 - 编程》。

  参数可以使用关键字 IN、OUT 或 INOUT 作为前缀。如果未指定上述任何值，则缺省情况下将使用 INOUT 参数。这些关键字具有以下含义：

  - **IN** 此参数是一个为过程提供值的表达式。
  - **OUT** 此参数是一个可由过程赋值的变量。
  - **INOUT** 此参数是一个为过程提供值的变量，并且可由过程赋值。

  使用 CALL 语句执行过程时，并不需要指定所有参数。如果在 CREATE PROCEDURE 语句中提供了缺省值，则会为缺少的参数指派缺省值。如果 CALL 语句中未提供参数，且缺省值未设置，则会给出错误。

指定 OR REPLACE (CREATE OR REPLACE PROCEDURE) 将创建一个新过程或替换同名的现有过程。此子句将更改过程的定义，但保留现有权限。如果尝试替换已使用的过程，则将返回错误。

您无法创建 TEMPORARY Web 服务过程。

- **URL 子句** 指定 Web 服务的 URI。其中的用户名和口令参数是可选的，它们提供了一种用于提供 HTTP 基本验证所需的证书的方法。HTTP 基本验证对用户和口令信息进行基于 64 位的编码，并在 HTTP 请求的验证标头中传递它。当以这种方式进行指定时，会以未加密形式将用户名和口令作为 URL 的一部分进行传递。

  如果指定 HTTPS 作为 URI 方案，将配置用于通过安全套接字层安全通信的过程。此类 URI 要求使用相应的 CERTIFICATE 子句来验证服务器并建立安全数据通道。

  指定 HTTPS_FIPS 会强制系统使用 FIPS 库。如果指定了 HTTPS_FIPS，但不存在 FIPS 库，则会改为使用非 FIPS 库。

  当以这种方式进行指定时，会以未加密形式将用户名和口令作为 URL 的一部分进行传递。

- **TYPE 子句** 用于指定创建 Web 服务请求时使用的格式。如果指定 SOAP 或未包括 TYPE 子句，则使用缺省类型 SOAP:RPC。HTTP 隐含 HTTP:POST。由于 SOAP 请求总是作为 XML 文档发送，因此总是使用 HTTP:POST 发送 SOAP 请求。请参见“创建 Web 服务客户端函数和过程”一节 《SQL Anywhere 服务器 - 编程》。

- **HEADER 子句** 创建 HTTP Web 服务客户端过程时，此子句用于添加、修改或删除 HTTP 请求标头条目。标头说明与 RFC2616 超文本传输协议—HTTP/1.1 中指定的格式，以及用于 ARPA Internet 文本消息的 RFC822 标准极其相似，其中包括只能为 HTTP 标头指定可打印的 ASCII 字符，且这些字符不区分大小写这一事实。

  有关使用 HTTP 标头的详细信息，请参见“处理 HTTP 标头”一节 《SQL Anywhere 服务器 - 编程》。

- **CERTIFICATE 子句** 要创建安全 (HTTPS) 请求，客户端必须有权访问 HTTPS 服务器所用的证书。必要的信息在一个用分号分隔的键/值对字符串中指定。可以将证书放置在一个文件中并使用 file 关键字提供文件名，或是将整个证书放置在一个字符串中，但是这两种方法不可同时使用。可以使用以下键：

按键	缩写	说明
file		证书的文件名。
certificate	cert	证书本身。
company	co	证书中指定的公司。
unit		证书中指定的公司单位。
name		证书中指定的公用名。

只有被定向到 HTTPS 服务器的请求或可从非安全服务器重定向到安全服务器的请求才需要证书。

- **CLIENTPORT 子句** 标识 HTTP 客户端过程使用 TCP/IP 进行通信的端口号。该子句是为通过防火墙的连接提供的，并建议只用于此类连接，因为防火墙过滤 "外发" TCP/IP 连接。您可以指定单个端口号、端口号范围或是两者的组合；例如 CLIENTPORT '85,90-97'。请参见“ClientPort 协议选项 [CPORT]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **PROXY 子句** 指定代理服务器的 URI。在客户端必须通过代理访问网络时使用。指示过程将要连接到代理服务器，并通过它将请求发送到 Web 服务。
- **SET 子句** 指定 HTTP 和 SOAP 的协议特定行为选项。以下列表介绍了受支持的 SET 选项。CHUNK 和 VERSION 适用于 HTTP 协议，而 OPERATION 适用于 SOAP 协议。此子句支持参数替代。
  - **'HTTP(CH[UNK]=option)'** （HTTP 或 SOAP）此选项可用于指定是否使用分块。分块可将 HTTP 消息拆分为几个部分。可能的值是 ON（始终分块）、OFF（从不分块） 和 AUTO（仅在内容超过 8196 字节时分块，自动生成的标记除外）。例如，以下 SET 子句可启用分块：

    ```
    SET 'HTTP(CHUNK=ON)'
    ```

    如果未指定 CHUNK 选项，缺省行为是 AUTO。如果分块请求在 AUTO 模式中失败，状态为 505（**HTTP 版本不受支持**）、501（**未实现**）或 411（**必需的长度**），则客户端将重试该请求但不使用分块传输编码。

    如果 HTTP 服务器不支持分块传输编码请求，将 CHUNK 选项设置为 OFF（从不分块）。

    由于 CHUNK 模式从 HTTP 1.1 版开始支持传输编码，将 CHUNK 设置为 ON 需要将版本 (VER) 设置为 1.1，或根本不设置，在后一种情况下，1.1 将作为缺省版本使用。
  - **' HTTP(VER[SION]=ver)'** （HTTP 或 SOAP）此选项可指定用于 HTTP 消息格式的 HTTP 协议的版本。例如，以下 SET 子句将 HTTP 版本设置为 1.1：

    ```
    SET 'HTTP(VERSION=1.1)'
    ```

    可能的值是 1.0 和 1.1。如果未指定 VERSION：
    - 如果将 CHUNK 设置为 ON，1.1 将作为 HTTP 版本使用
    - 如果将 CHUNK 设置为 OFF，1.0 将作为 HTTP 版本使用
    - 如果将 CHUNK 设置为 AUTO，会使用 1.0 或 1.1，这取决于客户端是否在 CHUNK 模式下发送
  - **' SOAP(OP[ERATION]=soap-operation-name)** （仅 SOAP）如果 SOAP 操作的名称与您正在创建的过程的名称不同，此选项可用于指定 SOAP 操作的名称。OPERATION 的值类似于远程过程调用的名称。例如，如果想要创建过程 accounts_login，此过程调用名为 login 的 SOAP 操作，则您需要按如下方式指定一些东西：

    ```
    CREATE PROCEDURE accounts_login(
          name LONG VARCHAR,
          pwd LONG VARCHAR )
       SET 'SOAP(OPERATION=login)'
    ```

    如果未指定 OPERATION 选项，SOAP 操作的名称必须与正在创建的过程的名称匹配。

  以下语句显示了几个 *protocol-option* 设置在同一 SET 子句中是如何组合的：


```
CREATE PROCEDURE accounts_login(
      name LONG VARCHAR,
      pwd LONG VARCHAR )
   SET 'HTTP ( CHUNK=ON; VERSION=1.1 ), SOAP( OPERATION=login )'
   ...
```

- **SOAPHEADER 子句** （仅 SOAP 格式）当将 SOAP Web 服务声明为过程时，此子句用于指定一个或多个 SOAP 请求标头条目。SOAP 标头可声明为静态常量，也可使用参数替代机制动态设置（为参数 hd1、hd2 等声明 IN、OUT 或 INOUT 参数）。Web 服务过程可定义一个或多个 IN 模式替代参数，以及单个 INOUT 或 OUT 替代参数。

  以下示例说明客户端如何指定发送多个带有参数的标头条目和接收响应 SOAP 标头数据：

  ```
  CREATE PROCEDURE soap_client
    (INOUT hd1 LONG VARCHAR, IN hd2 LONG VARCHAR, IN hd3 LONG VARCHAR)
    URL 'localhost/some_endpoint'
    SOAPHEADER '!hd1!hd2!hd3';
  ```

  有关如何使用 SOAP 标头的详细信息，请参见“处理 SOAP 标头”一节 《SQL Anywhere 服务器 - 编程》。

- **NAMESPACE 子句** （仅 SOAP 格式）此子句标识 SOAP:RPC 和 SOAP:DOC 请求通常都需要的方法命名空间。处理请求的 SOAP 服务器使用此命名空间来解释 SOAP 请求消息主体中的实体名称。可以通过 Web 服务服务器，从 SOAP 服务的 WSDL（Web Services Description Language，简称 Web 服务描述语言）中获取命名空间。缺省值是过程的 URL，但是不包括可选的路径组件。有关如何使用 SOAP 命名空间的详细信息，请参见“使用结构化数据类型”一节 《SQL Anywhere 服务器 - 编程》。

有关如何创建 Web 服务的详细信息，包括示例，请参见“SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》。

注释

参数值将作为请求的一部分进行传递。使用的语法取决于请求的类型。对于 HTTP:GET，参数将作为 URL 的一部分进行传递；对于 HTTP:POST 请求，则将值放在请求主体中。SOAP 请求的参数总是被绑定在请求主体中。

权限

必须具有 RESOURCE 权限。

必须有 DBA 权限才能为其他用户创建过程。

副作用

自动提交。

另请参见

- “ALTER PROCEDURE 语句”一节第 361 页
- “CALL 语句”一节第 400 页
- “CREATE FUNCTION 语句”一节第 438 页
- “CREATE FUNCTION 语句（Web 服务）”一节第 445 页
- “CREATE PROCEDURE 语句”一节第 458 页
- “DROP PROCEDURE 语句”一节第 550 页
- “GRANT 语句”一节第 595 页
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “创建 Web 服务客户端函数和过程”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 持久存储模块特性。Java 结果集的语法扩展如可选的 J621 特性中所指定。

示例

以下示例创建一个名为 FtoC 的 Web 服务客户端过程。

```
CREATE PROCEDURE FtoC( IN temperature FLOAT,
    INOUT inoutheader LONG VARCHAR,
    IN inheader LONG VARCHAR )
  URL 'http://localhost:8082/FtoCService'
  TYPE 'SOAP:DOC'
  SOAPHEADER '!inoutheader!inheader';
```

CREATE PUBLICATION 语句 [MobiLink] [SQL Remote]

此语句用于创建发布。在 MobiLink 中，发布可以标识 SQL Anywhere 远程数据库中的同步数据。在 SQL Remote 中，发布可以标识统一数据库和远程数据库中的复制数据。

语法 1（MobiLink 通用）

CREATE PUBLICATION [*owner.*]*publication-name*
(*article-definition*, ... **)**

article-definition :
  TABLE *table-name* [**(** *column-name*, ... **)**]
[**WHERE** *search-condition*]

语法 2（MobiLink 脚本式上载）

CREATE PUBLICATION [*owner.*]*publication-name*
WITH SCRIPTED UPLOAD
(*article-definition*, ... **)**

article-definition :
  TABLE *table-name* [**(** *column-name*, ... **)**]
[**USING (** [**PROCEDURE**] [*owner.*][*procedure-name*]
  FOR UPLOAD { **INSERT** | **DELETE** | **UPDATE** }, ... **)**]

语法 3（MobiLink 仅下载发布）

CREATE PUBLICATION [*owner*.]*publication-name*
FOR DOWNLOAD ONLY
(*article-definition*, ... **)**

article-definition : **TABLE** *table-name* [**(** *column-name*, ... **)**]

语法 4 (SQL Remote)

CREATE PUBLICATION [*owner*.]*publication-name*
(*article-definition*, ... **)**

article-definition :
TABLE *table-name* [**(** *column-name*, ... **)**]
[**WHERE** *search-condition*]
[**SUBSCRIBE BY** *expression*]

参数

- **article-definition** 发布是由项目构建而成。若要包含多个项目，请使用逗号分隔各项目定义。每个项目都是一个表或表的一部分。项目可以是表的垂直分区（表列的子集）、水平分区（基于 WHERE 子句的表行的子集）或垂直水平分区。

  在语法 2（用于执行脚本式上载的发布）中，项目说明还会注册用于定义上载的脚本。请参见“创建脚本式上载发布”一节 《MobiLink - 客户端管理》。

  在语法 3（用于仅下载发布）中，项目仅指定要下载的表和列。

- **WHERE 子句** WHERE 子句定义项目中包括的表行子集的一种方法。

  在 MobiLink 应用程序中，WHERE 子句会影响上载中所包含的行。（下载由 download_cursor 脚本定义。）在 MobiLink SQL Anywhere 远程数据库中，WHERE 子句只能引用包含在项目中的列，而不能包含子查询、变量或非确定性函数。

- **SUBSCRIBE BY 子句** 在 SQL Remote 中，定义要包含在项目中的表行子集的一种方法是使用 SUBSCRIBE BY 子句。此子句允许多个不同的预订者在单个发布定义中接收来自某表的不同行。

注释

CREATE PUBLICATION 语句可在数据库中创建一个发布。可以通过指定所有者名称为另一个用户创建发布。

在 MobiLink 中，发布在 SQL Anywhere 远程数据库中是必需的，而在 UltraLite 数据库中却是可选的。这些发布以及对它们的预订决定了将哪些数据上载到 MobiLink 服务器。

应使用 CREATE SYNCHRONIZATION SUBSCRIPTION 语句或 ALTER SYNCHRONIZATION SUBSCRIPTION 语句中的 ADD OPTION 子句为 MobiLink 发布设置选项。

语法 2 创建用于脚本式上载的发布。USING 子句用于注册想要用于定义上载的存储过程。对于每个表，最多可使用三个存储过程：分别用于插入、删除和更新。

语法 3 创建可在没有日志文件的情况下进行同步的仅下载发布。当同步仅下载发布时，下载行可能会覆盖对远程数据库中的那些行所做的更改。

在 SQL Remote 中，发布是一种双向操作，因为在统一数据库和远程数据库中都可以输入数据。在 SQL Remote 安装中，任何统一数据库和所有远程数据库都必须定义相同的发布。从统一数据库运行 SQL Remote 抽取实用程序时，将在远程数据库中自动执行正确的 CREATE PUBLICATION 语句。

权限

必须具有 DBA 权限。要求可以对语句中引用的所有表进行独占访问。

副作用

自动提交。

另请参见

- “ALTER PUBLICATION 语句 [MobiLink] [SQL Remote]”一节第 362 页
- “DROP PUBLICATION 语句 [MobiLink] [SQL Remote]”一节第 551 页
- “CREATE SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节第 494 页
- “ALTER SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节第 370 页
- SQL Anywhere MobiLink 客户端： “发布数据”一节 《MobiLink - 客户端管理》
- UltraLite MobiLink 客户端： “UltraLite CREATE PUBLICATION 语句”一节 《UltraLite - 数据库管理和参考》
- SQL Remote： “发布和项目”一节 《SQL Remote》
- “脚本式上载” 《MobiLink - 客户端管理》
- “仅下载发布”一节 《MobiLink - 客户端管理》
- “ISYSSYNC 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句发布两个表中的所有列和行。

```
CREATE PUBLICATION pub_contact (
   TABLE Contacts,
   TABLE Company
);
```

下列语句仅发布一个表中的某些列。

```
CREATE PUBLICATION pub_customer (
   TABLE Customers ( ID, CompanyName, City )
);
```

下列语句包含一个测试 Customers 表 Status 列的 WHERE 子句，因此只发布活动的客户行。

```
CREATE PUBLICATION pub_customer (
   TABLE Customers ( ID, CompanyName, City, State, Status )
   WHERE Status = 'active'
);
```

下列语句提供预订者值，因此仅发布某些行。此方法只能在 SQL Remote 中使用。

```
CREATE PUBLICATION pub_customer (
   TABLE Customers ( ID, CompanyName, City, State )
```



```
   SUBSCRIBE BY State
);
```

创建 SQL Remote 预订时，按如下所示使用预订者值。

```
CREATE SUBSCRIPTION TO pub_customer ( 'NY' )
   FOR jsmith;
```

下例创建使用脚本式上载的 MobiLink 发布：

```
CREATE PUBLICATION pub WITH SCRIPTED UPLOAD (
      TABLE t1 (a, b, c) USING (
         PROCEDURE my.t1_ui FOR UPLOAD INSERT,
         PROCEDURE my.t1_ud FOR UPLOAD DELETE,
         PROCEDURE my.t1_uu FOR UPLOAD UPDATE
      ),
      TABLE t2 AS my_t2 USING (
         PROCEDURE my.t2_ui FOR UPLOAD INSERT
      )
   );
```

以下示例创建仅下载发布：

```
CREATE PUBLICATION p1 FOR DOWNLOAD ONLY (
   TABLE t1
);
```

CREATE REMOTE MESSAGE TYPE 语句 [SQL Remote]

此语句用于标识消息链接和从数据库返回外发消息的地址。

语法

CREATE REMOTE MESSAGE TYPE *message-system*
[**ADDRESS** *address-string*]

message-system:
FILE
| **FTP**
| **SMTP**

参数

- **message-system**　支持的消息系统之一。
- **address-string**　指定消息系统的地址。

注释

消息代理使用受支持的消息链接之一从数据库发送外发消息。只要远程数据库是用抽取实用程序创建的，为使用指定链接的用户返回的消息都将发送到指定的地址。消息代理仅在有远程用户使用这些链接时才会启动这些链接。

地址是指定的消息系统中发布者的地址。如果是电子邮件系统，则地址字符串必须是有效的电子邮件地址。如果是文件共享系统，地址字符串是 SQLREMOTE 环境变量中设置的目录的子目录，或者是当前目录的子目录（如果未设置目录）。在远程数据库中，可以对 GRANT CONSOLIDATE 语句替换此设置。

要移除地址，请执行不带 ADDRESS 子句的 CREATE REMOTE MESSAGE TYPE 语句。

初始化实用程序 (dbinit) 会自动创建不带地址的消息类型。与其它 CREATE 语句不同，如果已有该类型，CREATE REMOTE MESSAGE TYPE 语句不会返回错误，而是会变更类型。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “GRANT PUBLISH 语句 [SQL Remote]”一节第 601 页
- “GRANT REMOTE 语句 [SQL Remote]”一节第 602 页
- “GRANT CONSOLIDATE 语句 [SQL Remote]”一节第 599 页
- “DROP REMOTE MESSAGE TYPE 语句 [SQL Remote]”一节第 552 页
- “SQL Remote 消息系统”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

使用抽取实用程序抽取远程数据库时，下列语句设置文件消息系统消息的所有接收者，以将消息发送回 *company* 子目录。

此语句还指示 dbremote 查看 *company* 子目录是否有进来的消息。

```
CREATE REMOTE MESSAGE TYPE file
ADDRESS 'company';
```

CREATE SCHEMA 语句

此语句用于创建数据库用户的表、视图和权限的集合。

语法

CREATE SCHEMA AUTHORIZATION *userid*
[
create-table-statement
| *create-view-statement*
| *grant-statement*
] ... ;

注释

CREATE SCHEMA 语句用于创建模式。模式是表、视图及其关联权限的集合。

userid 必须是当前连接的用户 ID。不可以为其他用户创建模式。

如果 CREATE SCHEMA 语句中包含的任何语句失败，则回退整个 CREATE SCHEMA 语句。

CREATE SCHEMA 语句只是一种将各个 CREATE 语句和 GRANT 语句合成为一个操作的方法。数据库中没有创建任何 SCHEMA 数据库对象，要删除对象，必须使用各 DROP TABLE 或 DROP VIEW 语句。要撤消权限，必须对授予的每个权限使用 REVOKE 语句。

各个 CREATE 语句或 GRANT 语句不用语句分隔符分隔。语句分隔符用于标记 CREATE SCHEMA 语句其本身的结尾。

必须排序各 CREATE 或 GRANT 语句，以便在授予对象权限之前创建它们。

尽管目前可以为用户创建多个模式，但不建议这样做，将来的版本可能也不支持这样做。

权限

必须具有 RESOURCE 权限。

副作用

自动提交。

另请参见

- “CREATE TABLE 语句”一节第 497 页
- “CREATE VIEW 语句”一节第 520 页
- “GRANT 语句”一节第 595 页

标准和兼容性

- **SQL/2003** 核心特性。
- **Sybase** SQL Anywhere 不支持在 CREATE SCHEMA 语句中使用 REVOKE 语句，也不允许在 Transact-SQL 批处理或过程中使用它。

示例

下面的 CREATE SCHEMA 语句将创建一个由两个表组成的模式。该语句必须由具有 RESOURCE 权限的用户 ID sample_user 执行。如果创建表 t2 的语句失败，则两个表都不创建。

```
CREATE SCHEMA AUTHORIZATION sample_user
CREATE TABLE t1 ( id1 INT PRIMARY KEY )
CREATE TABLE t2 ( id2 INT PRIMARY KEY );
```

下列 CREATE SCHEMA 语句中的语句分隔符放置在第一个 CREATE TABLE 语句之后。由于语句分隔符标记 CREATE SCHEMA 语句的结束，因此数据库服务器将该示例解释为两个批处理语句。如果创建表 t2 的语句失败，仍会创建表 t1。

```
CREATE SCHEMA AUTHORIZATION sample_user
CREATE TABLE t1 ( id1 INT PRIMARY KEY );
CREATE TABLE t2 ( id2 INT PRIMARY KEY );
```

CREATE SERVER 语句

此语句用于创建远程服务器。

语法 1

CREATE SERVER *server-name*
CLASS *server-class-string*
USING *connection-info-string*
[**READ ONLY**]

server-class-string :
'SAODBC'
| **'ASEODBC'**
| **'DB2ODBC'**
| **'MSSODBC'**
| **'ORAODBC'**
| **'MSACCESSODBC'**
| **'MYSQLODBC'**
| **'ULODBC'**
| **'ADSODBC'**
| **'ODBC'**
| **'SAJDBC'**
| **'ASEJDBC'**

connection-info-string :
{ *host-name*:*port-number* [*/dbname*] | *data-source-name* | *sqlanywhere-connection-string* }

语法 2

CREATE SERVER *server-name*
CLASS 'DIRECTORY'
USING *using-string*

using-string :
'ROOT = *path*
[**;SUBDIRS =** *n*]
[**;READONLY =** { **YES** | **NO** }]**'**

参数

- **CLASS 子句** 指定要用于远程连接的服务器类。服务器类包含详细的服务器功能信息。语法 2 中使用 DIRECTORY 类来访问本地计算机上的目录。
- **USING 子句** 在语法 1 中，USING 子句为数据库服务器提供了一个连接字符串。适用的连接字符串取决于所使用的驱动程序，而驱动程序又取决于 *server-class-string*。

  如果使用基于 ODBC 的服务器类，则 USING 子句为 *data-source-name*。*data-source-name* 是 ODBC 数据源名称。

  对于 SQL Anywhere 远程服务器（SAODBC 服务器类），*connection-info-string* 参数可以是任何有效的 SQL Anywhere 连接字符串。可以使用任何 SQL Anywhere 连接参数。例如，如果存在连接问题，则可以包含一个 LOG 连接参数以对连接尝试进行故障排除。

  有关 SQL Anywhere 连接字符串的详细信息，请参见“连接参数”一节 《SQL Anywhere 服务器 - 数据库管理》。

  在 Unix 平台上，您还需要引用 ODBC 驱动程序管理器。例如，使用所提供的 iAnywhere Solutions ODBC 驱动程序时，语法如下：

```
USING 'driver=SQL Anywhere 11;dsn=my_dsn'
```


如果使用基于 JDBC 的服务器类，则 USING 子句的形式为 *host-name:port-number* [*/dbname*]，其中：

- **host-name** 运行远程服务器的计算机。
- **port-number** 远程服务器所监听的 TCP/IP 端口号。SQL Anywhere 的缺省端口号是 2638。
- **dbname** 对于 SQL Anywhere 远程服务器，如果未指定 *dbname*，则使用缺省数据库。对于 Adaptive Server Enterprise，缺省值为 master 数据库，除使用 *dbname* 之外，另一个选择是通过其它某些方法（例如，在 FORWARD TO 语句中），使用另一个数据库。

在语法 2 中，USING 子句为本地目录指定以下值：

- **ROOT 子句** 即代表目录访问类根目录的路径（相对于数据库服务器）。当使用目录访问服务器名创建代理表时，代理表相对于此根路径。
- **SUBDIRS 子句** 介于 0 与 10 之间的数字，用于表示根目录内可供数据库服务器访问的目录层级数。如果省略 SUBDIRS 或将其设置为 0，则通过目录访问服务器仅可以访问根目录中的文件。可通过目录访问服务器创建任何可用目录或子目录的代理表。
- **READONLY 子句** 指定是否可修改由目录访问服务器访问的文件。缺省情况下，此值设置为 NO。
- **CREATEDIRS 子句** 指定是否可使用目录访问服务器创建目录。缺省情况下，此值设置为 NO。

注释

当创建远程服务器时，会将它添加到 ISYSSERVER 系统表中。

语法 1 CREATE SERVER 语句用于定义远程服务器。

有关服务器的类及如何配置服务器的详细信息，请参见“用于远程数据访问的服务器类”《SQL Anywhere 服务器 - SQL 的用法》。

语法 2 CREATE SERVER 语句允许创建一个访问运行数据库服务器的计算机上的本地目录结构的目录访问服务器。必须为需要使用目录访问服务器的每个数据库用户创建一个外部登录。在 Unix 上，数据库服务器以特定用户身份运行，因此文件权限基于授予给数据库服务器用户的权限而定。

有关目录访问服务器的详细信息，请参见“使用目录访问服务器”一节《SQL Anywhere 服务器 - SQL 的用法》。

权限

要执行此命令，必须具有 DBA 权限。

Windows Mobile 上不支持。

副作用

自动提交。

另请参见

- “ALTER SERVER 语句”一节第 364 页
- “DROP SERVER 语句”一节第 552 页
- “用于远程数据访问的服务器类” 《SQL Anywhere 服务器 - SQL 的用法》
- “ISYSSERVER 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例将创建一个名为 testsa 的 SQL Anywhere 远程服务器，它位于名为 apple 的计算机上，并监听端口号 2638：

```
CREATE SERVER testsa
CLASS 'SAJDBC'
USING 'apple:2638';
```

下例将为名为 ase_prod 的基于 JDBC 的 Adaptive Server 创建一个远程服务器。它的计算机名是 banana，端口号是 3025。

```
CREATE SERVER ase_prod
CLASS 'asejdbc'
USING 'banana:3025';
```

下例为名为 oracle723 的 Oracle 服务器创建一个远程服务器。它的 ODBC 数据源名称是 oracle723。

```
CREATE SERVER oracle723
CLASS 'oraodbc'
USING 'oracle723';
```

以下示例创建一个仅能访问 *c:\temp* 目录内文件的目录访问服务器：

```
CREATE SERVER diskserver0
CLASS 'directory'
USING 'root=c:\temp';
CREATE EXTERNLOGIN DBA TO diskserver0;
CREATE EXISTING TABLE diskdir0 AT 'diskserver0;;;.';

-- Get a list of those files.
SELECT permissions, file_name, size FROM diskdir0;
```

以下示例创建一个可访问九个目录层级的目录访问服务器：

```
-- Create a directory server that sees 9 levels of directories.
CREATE SERVER diskserver9
CLASS 'directory'
USING 'ROOT=c:\temp;SUBDIRS=9';
CREATE EXTERNLOGIN DBA TO diskserver9;
CREATE EXISTING TABLE diskdir9 AT 'diskserver9;;;.';
```

CREATE SERVICE 语句

使用此语句，允许数据库服务器充当 Web 服务器。

语法 1 - DISH 服务

CREATE SERVICE *service-name*
TYPE 'DISH'
[**GROUP** { *group-name* | **NULL** }]
[**FORMAT** { **'DNET'** | **'CONCRETE'** [**EXPLICIT** { **ON** | **OFF** }] | **'XML'** | **NULL** }]
[*common-attributes*]

语法 2 - SOAP 服务

CREATE SERVICE *service-name*
TYPE 'SOAP'
[**DATATYPE** { **ON** | **OFF** | **IN** | **OUT** }]
[**FORMAT** { **'DNET'** | **'CONCRETE'** [**EXPLICIT** { **ON** | **OFF** }] | **'XML'** | **NULL** }]
[*common-attributes*]
AS *statement*

语法 3 - 杂项服务

CREATE SERVICE *service-name*
TYPE { **'RAW'** | **'HTML'** | **'JSON'** | **'XML'** }
[**URL** [**PATH**] { **ON** | **OFF** | **ELEMENTS** }]
[*common-attributes*]
[**AS** { *statement* | **NULL** }]

common-attributes:
[**AUTHORIZATION** { **ON** | **OFF** }]
[**ENABLE** | **DISABLE**]
[**METHODS** '*method,...*']
[**SECURE** { **ON** | **OFF** }]
[**USER** { *user-name* | **NULL** }]

method:
DEFAULT
| **POST**
| **GET**
| **HEAD**
| **PUT**
| **DELETE**
| **NONE**
| *

参数

- **service-name** Web 服务名可以是由字母数字字符，或 /、-、_、.、!、~、*、'、(或) 组成的任意序列，但第一个字符不得以斜杠 (/) 开头，并且名称中不得包含两个或两个以上连续的斜杠字符。

  与其它服务不同，您不能在 DISH 服务名称中指定正斜线 (/)。

- **TYPE 子句** 通过返回的结果集标识服务类型。该类型必须是列出的服务类型之一。无缺省值。

  - **'SOAP'** 结果集作为 SOAP 响应返回。数据的格式由 FORMAT 子句确定。对 SOAP 服务的请求必须是有效 SOAP 请求，而不仅是简单的 HTTP 请求。有关 SOAP 标准的详细信息，请参见 http://www.w3.org/TR/2000/NOTE-SOAP-20000508/。

  - **'DISH'** DISH 服务 (Determine SOAP Handler) 充当 GROUP 子句标识的 SOAP 服务的代理，根据请求，为其中各 SOAP 服务生成 WSDL（Web 服务描述语言）。

- **'RAW'** 不对结果集另外进行任何格式设置就将其发送到客户端。可以通过在过程中显式生成所需的标记来生成格式化文档。如果使用 TYPE 'RAW' 创建服务，应使用 sa_set_http_header 设置 HTTP 标头 Content-Type。否则，有些浏览器会将内容显示为纯文本。请参见“使用提供 HTML 文档的过程”一节 《SQL Anywhere 服务器 - 编程》和“sa_set_http_header 系统过程”一节第 894 页。
- **'HTML'** 将语句或过程的结果集自动设置为包含表的 HTML 文档格式。
- **'JSON'** 结果集以 JavaScript Object Notation (JSON) 格式返回。JSON Web 服务与 XML Web 服务相似，它们都接受 "通用" HTTP 请求（即请求不必包含特殊格式的主体，如 SOAP Web 服务）。有关 JSON 的详细信息，请访问 http://www.json.org。
- **'XML'** 结果集以 XML 格式返回。如果结果集已是 XML 格式，则不应用其它任何格式设置。如果它还不是 XML 格式，则将其格式自动设置为 XML。其作用与在 SELECT 语句中使用 FOR XML RAW 子句的作用类似。

- **GROUP 子句** 仅适用于 DISH 服务。指定一个用于控制 DISH 服务公开哪些 SOAP 服务的公用前缀。例如，指定 GROUP *xyz* 则仅公开 SOAP 服务 *xyz/aaaa*、*xyz/bbbb* 或 *xyz/cccc*，而不公开 *abc/aaaa* 或 *xyzaaaa*。如果没有指定组名，则 DISH 服务公开数据库中的所有 SOAP 服务。SOAP 服务可由多个 DISH 服务公开。组名和服务名中允许使用相同的字符。

  创建服务时，GROUP NULL 与不指定 GROUP 子句效果相同。但是，如果在 ALTER SERVICE 语句中使用 GROUP NULL，GROUP NULL 将删除任何现有分组，而如果不使用 GROUP 子句则不会出现这种情况。

- **DATATYPE 子句** 仅适用于 SOAP 服务。控制是否对于所有 SOAP 服务格式都支持参数输入和/或结果集输出（响应）的数据类型设置。如果支持，则数据类型设置允许 SOAP 工具箱分析数据并将其归到相应类型。参数数据类型在 DISH 服务生成的 WSDL 的模式部分中公开。输出数据类型则表示为每列数据的 XML 模式类型属性。

  DATATYPE 子句允许使用以下值：

  - **ON** 为输入参数和结果集响应生成数据类型设置。
  - **OFF** 不生成输入参数和结果集响应的数据类型设置（缺省值）。
  - **IN** 仅生成输入参数的数据类型设置。
  - **OUT** 仅生成结果集响应的数据类型设置。

  有关 SOAP 服务的详细信息，请参见“使用 SOAP 服务”一节 《SQL Anywhere 服务器 - 编程》。

  有关将 XMLSchema 类型映射为 SQL 数据类型的详细信息，请参见“使用数据类型”一节 《SQL Anywhere 服务器 - 编程》。

- **URL 子句** 确定是否接受 URL 路径以及在接受的情况下如何处理 URL 路径。
  - **OFF** OFF 表示 URL 请求中的服务名后不能跟有任何内容。如果不允许使用 URL 路径的剩余部分，或者如果服务名称以正斜线 (/) 结尾，则指定 OFF。例如，如果选择 OFF 并且有 URL 路径 *http://<主机名>/<服务名>/aaa/bbb/ccc*，则只允许使用 *http://<主机名>/<服务名>*。不允许 URL 路径的其余部分 */<aaa/bbb/ccc*。

- **ON** ON 表示 URL 的余下部分将解释为变量 URL 的值。如果允许 URL 路径的其余部分并将其设置为单个参数，则指定 ON。例如，在 URL 路径 *http://<主机名>/<服务名>/aaa/bbb/ccc* 中，参数值为 *aaa/bbb/ccc*。
- **ELEMENTS** ELEMENTS 表示按照斜线字符将 URL 路径的余下部分拆分为数量最多为 10 个的一组元素，将其设为多个参数。例如，在 URL 路径 *http://<主机名>/<服务名>/aaa/bbb/ccc* 中，路径的每个元素都按单独的参数处理。例如 url1=aaa、url2=bbb、url3=ccc 等等。将这些值指派给名称由 url 加上一个 1 到 10 之间的数字作后缀组成的变量。如果提供的值少于 10 个，则将其余变量均设置为 NULL。

如果服务名以正斜线字符 / 结尾，则必须将 url 设置为 OFF。缺省值为 OFF。

有关 URL 的详细信息，请参见“了解如何解释 URL”一节 《SQL Anywhere 服务器 - 编程》和“处理变量”一节 《SQL Anywhere 服务器 - 编程》。

- **FORMAT 子句** 仅适用于 DISH 和 SOAP 服务。生成与各种类型 SOAP 客户端（例如 .NET 或 Java JAX-RPC）兼容的输出格式。如果未指定 SOAP 服务的格式，则从该服务的 DISH 服务声明继承格式。如果 DISH 服务也没有声明格式，则格式缺省设置为与 .NET 客户端兼容的 DNET。可以通过定义多个 DISH 服务（每个都使用一个不同的 FORMAT 类型）将没有声明格式的 SOAP 服务与不同类型的 SOAP 客户端同用。

  支持以下格式：

  - **'DNET'** 适于和 .NET SOAP 客户端一同使用的 Microsoft 数据集格式。DNET 是缺省的 FORMAT 值，并且是 SQL Anywhere 9.0.2 版之前唯一可用的格式。
  - **'CONCRETE'** 一种与平台无关的数据集格式，适于和 JAX-WS 之类的客户端，或根据返回数据结构的格式自动生成接口的客户端一同使用。指定此格式类型会在 WSDL 中提供显式 dataset 元素或 SimpleDataset 元素。

    指定 **EXPLICIT ON** 后（缺省值），当满足以下条件时，WSDL 描述显式 DataSet 元素：

    - CREATE SERVICE 语句调用存储过程
    - 在存储过程中指定 RESULT 语句

    如果服务语句或存储过程未正确定义，则由 DISH 服务生成的结果集映射将恢复为 SimpleDataset 元素。

    指定 **EXPLICIT OFF** 后，WSDL 描述 SimpleDataset 元素。SimpleDataset 元素将结果集描述为由一组行（其中每个行都包含一组列元素）组成的行集的包容层次。显式 dataset 元素通过包含各列的实际名称和类型，对其进行扩展。
  - **'XML'** 一种简单的 XML 字符串格式。将数据集作为可以传递到 XML 分析程序的字符串返回。此格式是 SOAP 客户端之间最便于移植的格式。
  - **NULL** SOAP 服务的格式将从 DISH 服务格式（如果存在）继承或缺省设置为 DNET。DISH 服务的格式将缺省设置为 DNET。创建新服务时，指定 FORMAT NULL 与不指定任何格式效果相同。但是，如果在 ALTER SERVICE 语句中使用 FORMAT NULL，FORMAT NULL 将替换任何原有格式，而如果不使用 FORMAT 子句则不替换原有格式。
- ***语句*** 如果语句为 NULL，则 URL 必须指定要执行的语句。否则，指定的 SQL 语句是唯一可通过该服务执行的语句。SOAP 服务必须含有语句；而 DISH 服务不能含有语句。缺省值为 NULL。

强烈建议在生产系统中使用的所有服务都定义一个语句。只有在启用了授权的情况下，该语句才能为 NULL。

- **AUTHORIZATION 子句** 确定在连接服务时用户是否必须通过基本 HTTP 授权指定用户名和口令。缺省值为 ON。如果 AUTHORIZATION 为 OFF，则需要使用 AS 子句，并且必须用 USER 子句标识单个用户。所有请求都使用该用户的帐户和权限运行。如果 AUTHORIZATION 为 ON，则所有用户都必须提供用户名和口令。也可通过使用 USER 子句提供用户名或用户组名来限定允许使用服务的用户。如果用户名为 NULL，则所有已知用户都可以访问该服务。建议在启用授权的情况下运行生产系统，并通过向组中添加用户来授予使用该服务的权限。

  当授权的值为 ON 时，与 Web 服务相连的 HTTP 客户端使用通过 64 位编码模糊处理用户和口令信息的基本授权 (RFC 2617)。建议使用 HTTPS 协议以提高安全性。

- **ENABLE 和 DISABLE 子句** 确定是否该服务是否可用。缺省情况下，创建服务时，即启用该服务。创建或变更服务时，可以包含 ENABLE 或 DISABLE 子句。禁用服务可以使该服务离线。之后可以使用 ALTER SERVICE 及 ENABLE 子句启用该服务。

- **METHODS 子句** 指示服务支持哪类请求。有效的请求类型有 DEFAULT、POST、GET、HEAD、PUT、DELETE 和 NONE。可将星号 (*) 用作简写形式，以表示 POST、GET 和 HEAD 属性，这些属性是 RAW、HTML 和 XML 服务类型的缺省请求类型。SOAP 服务的缺省请求类型是 POST 和 HEAD。DISH 服务的缺省请求类型是 GET 和 HEAD。并非所有的请求类型都对所有服务类型有效。下表总结了每种服务类型的有效请求类型。

请求类型	适用的服务	说明
DEFAULT	所有服务	DEFAULT 用于将请求类型设置重置为给定服务类型的缺省设置。该请求类型不能包含在具有其它请求类型的列表中。
POST	SOAP、DISH、RAW、HTML、XML	SOAP、RAW、HTML 和 XML 缺省情况下启用此请求类型。
GET	DISH、RAW、HTML、XML	DISH、RAW、HTML 和 XML 缺省情况下启用此请求类型。
HEAD	SOAP、DISH、RAW、HTML、XML	SOAP、DISH、RAW、HTML 和 XML 缺省情况下启用此请求类型。
PUT	RAW、HTML、XML	缺省情况下不启用。

请求类型	适用的服务	说明
DELETE	RAW、HTML、XML	缺省情况下不启用。
NONE	所有服务	使用 NONE 来禁止对服务的访问。如果应用于 SOAP 服务，则 SOAP 请求无法直接访问该服务。 始终将 DISH 服务作为 SOAP 端点用于一个或多个 SOAP 服务。要强行通过 DISH 服务端点独占访问 SOAP 操作，请为每个 SOAP 服务指定 **NONE**。
*	RAW、HTML、XML	与指定 **'POST,GET,HEAD'** 效果相同。

例如，要为 RAW 服务类型指定请求类型的完整列表，可以使用以下任意一个子句：

```
METHODS '*,PUT,DELETE'
METHODS 'POST,GET,HEAD,PUT,DELETE'
```

要将任意服务类型的请求类型列表重置为缺省值，可以使用下列子句：

```
METHODS 'DEFAULT'
```

- **SECURE 子句** 指明是否接受不安全的连接。ON 表示只接受 HTTPS 连接。HTTP 端口上接收的服务请求将自动重定向到 HTTPS 端口。如果设置为 OFF，则 HTTP 和 HTTPS 两种连接都接受。缺省值为 OFF。
- **USER 子句** 如果禁用授权，则此参数变为必需参数，并指定用于执行所有服务请求的用户 ID。如果启用授权（缺省设置），则此可选子句标识允许访问该服务的用户或用户组。缺省值为 NULL，即向所有用户授予访问权限。

注释

CREATE SERVICE 语句将使数据库服务器充当 Web 服务器。在 ISYSWEBSERVICE 系统表中创建一个新条目。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “ALTER SERVICE 语句”一节第 366 页
- “DROP SERVICE 语句”一节第 553 页
- “ISYSWEBSERVICE 系统表”一节第 766 页
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

若要快速设置 Web 服务器，则使用 -xs 选项（例如 -xs http）启动数据库服务器，然后执行下面的语句：

```
CREATE SERVICE tables TYPE 'HTML'
   AUTHORIZATION OFF
   USER DBA
   AS SELECT *
      FROM SYS.SYSTAB;
```

执行完此语句之后，使用任意 Web 浏览器打开 URL http://localhost/tables。

以下示例演示了如何编写一个 Hello World 程序。

```
CREATE PROCEDURE hello_world_proc( )
RESULT (html_doc long varchar)
BEGIN
   CALL dbo.sa_set_http_header( 'Content-Type', 'text/html' );
   SELECT '<html>\n'
       || '<head><title>Hello World</title></head>\n'
       || '<body>\n'
       || '<h1>Hello World!</h1>\n'
       || '</body>\n'
       || '</html>\n';
END;

CREATE SERVICE hello_world TYPE 'RAW'
AUTHORIZATION OFF
USER DBA
AS CALL hello_world_proc;
```

执行完此语句之后，使用任意 Web 浏览器打开 URL *http://localhost/hello_world*。

以下示例演示了如何创建 JSON 服务。

```
CREATE PROCEDURE ListEmployees()
RESULT (
 EmployeeID            integer,
 Surname               person_name_t,
 GivenName             person_name_t,
 StartDate             date,
 TerminationDate       date )
BEGIN
  SELECT EmployeeID, Surname, GivenName,
         StartDate, TerminationDate
  FROM Employees
END;

CREATE SERVICE "JSON/EmployeeList"
TYPE 'JSON'
AUTHORIZATION OFF
SECURE OFF
USER DBA
AS CALL ListEmployees();
```

执行完此语句之后，使用任意 Web 浏览器打开 URL *http://localhost/JSON/EmployeeList*，并将 JSON 响应保存到一个文件中，这样就可以使用文本编辑器浏览该服务。

CREATE STATISTICS 语句

重新创建优化程序所使用的列统计信息，并将它们存储到 ISYSCOLSTAT 系统表中。

语法

CREATE STATISTICS *object-name* [**(** *column-list* **)**]

object-name :
table-name | *materialized-view-name* | *temp-table-name*

注释

CREATE STATISTICS 语句重新创建列统计信息，SQL Anywhere 使用此统计信息优化数据库查询，可在基表、实例化视图、局部临时表和全局临时表上执行此语句。但不可以在代理表上创建统计信息。列统计信息包括直方图，它反映数据库中指定列的数据分布情况。缺省情况下，会自动为含有五个或五个以上数据行的表创建列统计信息。

在极少数情况下，数据库查询非常多变，且数据分布不均或数据频繁变化，这时可通过针对某表或某列运行 CREATE STATISTICS 语句来提高性能。

执行过程中，无论表大小如何，CREATE STATISTICS 语句都会更新现有列统计信息，除非表为空（在这种情况下不执行任何操作）。如果空表存在列统计信息，则 CREATE STATISTICS 语句不会更改这些信息。若要删除空表的列统计信息，则执行 DROP STATISTICS 语句。

运行 CREATE STATISTICS 的过程中会对表执行全面扫描。因此，请在经过慎重考虑后再执行 CREATE STATISTICS 语句。

如果删除了统计信息，建议您使用 CREATE STATISTICS 语句重新创建此统计信息。没有统计信息，优化程序可能会生成低效数据访问计划，从而导致数据库性能降低。

权限

必须具有 DBA 权限。

副作用

执行计划可能发生变化。

另请参见

- “优化程序估计值和列统计信息”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “DROP STATISTICS 语句”一节第 554 页
- “LOAD TABLE 语句”一节第 626 页
- “ISYSCOLSTAT 系统表”一节第 757 页
- “直方图实用程序 (dbhist)”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_get_histogram 系统过程”一节第 830 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下列语句更新 SalesOrderItems 表中 ProductID 列的列统计信息：

```
CREATE STATISTICS SalesOrderItems ( ProductID );
```

CREATE SUBSCRIPTION 语句 [SQL Remote]

此语句用于为用户创建发布预订。

语法

CREATE SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*

publication-name: *identifier*

subscription-value : *string*

subscriber-id: *string*

参数

- **publication-name** 用户所预订的发布的名称。这可以包括发布的所有者。
- **subscription-value** 与发布的预订表达式进行比较的字符串。预订者接收预订表达式与预订值相匹配的所有行。
- **subscriber-id** 发布预订者的用户 ID。此用户必须已被授予 REMOTE 权限。

注释

在 SQL Remote 安装中，会将数据安排到各发布中以备复制。若要接收 SQL Remote 消息，必须为具有 REMOTE 权限的用户 ID 创建一个预订。

如果预订中提供了字符串，则将该字符串与发布中的每个 SUBSCRIBE BY 表达式进行匹配。预订者将接收表达式的值等于所提供的字符串的所有行。

在 SQL Remote 中，发布和预订是一种双向关系。如果在统一数据库中为远程用户创建发布预订，则在远程数据库中也应为统一数据库创建预订。抽取实用程序会自动执行此过程。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “DROP SUBSCRIPTION 语句 [SQL Remote]”一节第 555 页
- “GRANT REMOTE 语句 [SQL Remote]”一节第 602 页
- “SYNCHRONIZE SUBSCRIPTION 语句 [SQL Remote]”一节第 724 页
- “START SUBSCRIPTION 语句 [SQL Remote]”一节第 716 页
- “ISYSSUBSCRIPTION 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句将为用户 p_chin 创建一个对发布 pub_sales 的预订。预订者将接收预订表达式的值为 Eastern 的所有行。

```
CREATE SUBSCRIPTION
TO pub_sales ( 'Eastern' )
FOR p_chin;
```

CREATE SYNCHRONIZATION PROFILE 语句 [MobiLink]

此语句用于创建 SQL Anywhere 同步配置文件。同步配置文件定义 SQL Anywhere 数据库如何与 MobiLink 服务器同步。

语法

CREATE SYNCHRONIZATION PROFILE *name string*

参数

- **name** 指定要创建的同步配置文件的名称。每个配置文件必须有唯一的名称。
- **string** 指定有效选项字符串，如下所述。选项字符串指定为用分号分隔的格式为 <选项名称>=<选项值> 的元素列表。例如 *publication=p1;verbosity=high*。

注释

有关 dbmlsync 所支持的同步配置文件选项的列表，请参见“MobiLink 同步配置文件”一节《MobiLink - 客户端管理》。

对于采用 Boolean 值的选项，将值设为 TRUE 等同于在命令行指定相应的选项。

以下值可用于指定 TRUE：TRUE、ON、1、YES。

以下值可用于指定 FALSE：FALSE、OFF、0、NO。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “ALTER SYNCHRONIZATION PROFILE 语句 [MobiLink]”一节第 369 页
- “DROP SYNCHRONIZATION PROFILE 语句 [MobiLink]”一节第 556 页

CREATE SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]

此语句用于在 SQL Anywhere 远程数据库中创建 MobiLink 用户与发布间的预订。

语法

CREATE SYNCHRONIZATION SUBSCRIPTION
TO *publication-name*
[**FOR** *ml_username*, ...]
[**TYPE** *network-protocol*]
[**ADDRESS** *protocol-options*]
[**OPTION** *option*=*value*, ...]

ml_username: *identifier*

network-protocol: **http** | **https** | **tls** | **tcpip**

protocol-options: *string*

value: *string* | *integer*

参数

- **TO 子句** 指定发布名称。
- **FOR 子句** 指定一个或多个 MobiLink 用户名。如果指定多个用户名，则为每个用户都创建一个单独预订。

  ml_username 是已被授权与 MobiLink 服务器同步的用户。

  有关同步用户名的详细信息，请参见“MobiLink 用户简介”一节 《MobiLink - 客户端管理》。

  忽略 FOR 子句以设置用于发布的协议类型、协议选项以及扩展选项。

  有关 dbmlsync 如何处理在不同位置指定的选项的信息，请参见“优先级顺序”一节 《MobiLink - 客户端管理》。
- **TYPE 子句** 此子句指定同步中使用的网络协议。缺省协议为 tcpip。

  有关网络协议的详细信息，请参见“CommunicationType (ctp) 扩展选项”一节 《MobiLink - 客户端管理》。
- **ADDRESS 子句** 此子句指定网络协议选项，例如 MobiLink 服务器的位置。多个选项之间必须用分号隔开。

  有关协议选项的完整列表，请参见“MobiLink 客户端网络协议选项汇总”一节 《MobiLink - 客户端管理》。
- **OPTION 子句** 此子句可用于为预订设置扩展选项。如果未提供 FOR 子句，则由扩展选项充当发布的缺省设置。

  有关 dbmlsync 如何处理在不同位置指定的选项的信息，请参见“优先级顺序”一节 《MobiLink - 客户端管理》。

有关选项的完整列表，请参见“MobiLink SQL Anywhere 客户端扩展选项” 《MobiLink - 客户端管理》。

注释

network-protocol、*protocol-options* 和 *options* 可在多处设置。

有关 dbmlsync 如何处理在不同位置指定的选项的信息，请参见“优先级顺序”一节 《MobiLink - 客户端管理》。

此语句会使各选项和其它信息存储在 SQL Anywhere ISYSSYNC 系统表中。具有数据库 DBA 权限的任何人都可查看这些信息，其中包括口令和加密证书。为避免发生这种潜在安全问题，可在 dbmlsync 命令行上指定信息。

请参见“dbmlsync 语法”一节 《MobiLink - 客户端管理》。

权限

必须具有 DBA 权限。要求对发布中引用的所有表具有独占访问权限。

副作用

自动提交。

另请参见

- “ALTER SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节第 370 页
- “DROP SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节第 557 页
- SQL Anywhere MobiLink 客户端： “创建同步预订”一节 《MobiLink - 客户端管理》
- UltraLite MobiLink 客户端： “设计 UltraLite 中的同步”一节 《UltraLite - 数据库管理和参考》
- “ISYSSYNC 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例将在 MobiLink 用户 ml_user1 与名为 sales_publication 的发布之间创建一个预订，并将内存设置为 3 MB：

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION memory='3m';
```

以下示例省略了 FOR 子句，这样就会存储名为 sales_publication 的发布的设置：

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   ADDRESS 'host=test.internal;port=2439;
      security=ecc_tls'
   OPTION memory='2m';
```

CREATE SYNCHRONIZATION USER 语句 [MobiLink]

此语句用于在 SQL Anywhere 远程数据库中创建 MobiLink 用户。

语法

CREATE SYNCHRONIZATION USER *ml_username*
[**TYPE** *network-protocol*]
[**ADDRESS** *protocol-options*]
[**OPTION** *option*=*value*, ...]

ml_username: *identifier*

network-protocol :
tcpip
| **http**
| **https**
| **tls**

protocol-options : *string*

value: *string* | *integer*

参数

- **ml_username** 标识 MobiLink 用户的名称。

  有关 MobiLink 用户的详细信息，请参见“MobiLink 用户简介”一节 《MobiLink - 客户端管理》。

- **TYPE 子句** 此子句指定同步中使用的网络协议。缺省协议为 tcpip。

  有关通信协议的详细信息，请参见“CommunicationType (ctp) 扩展选项”一节 《MobiLink - 客户端管理》。

- **ADDRESS 子句** 此子句指定 *protocol-options*，格式为 *keyword*=*value*，用分号分隔。您提供的设置取决于当前使用的通信协议（TCPIP、TLS、HTTP 或 HTTPS）。

  有关协议选项的完整列表，请参见“MobiLink 客户端网络协议选项汇总”一节 《MobiLink - 客户端管理》。

- **OPTION 子句** OPTION 子句可用于使用 *option*=*value* 以逗号分隔的列表形式设置扩展选项。

  每个选项的值不能包含等号或分号。数据库服务器接受您输入的任何选项而检查其是否有效。因此，如果选项的拼写不正确或输入的值无效，则直到您运行 dbmlsync 命令执行同步时，才会出现错误消息。

  在各个预订或在 dbmlsync 命令行上可以覆盖为同步用户设置的选项。

  有关扩展选项的信息，请参见“MobiLink SQL Anywhere 客户端扩展选项” 《MobiLink - 客户端管理》。

  network-protocol、*protocol-options* 和 *options* 可在多处设置。

  有关 dbmlsync 如何处理在不同位置指定的选项的信息，请参见“优先级顺序”一节 《MobiLink - 客户端管理》。

此语句会使各选项和其它信息存储在 SQL Anywhere ISYSSYNC 系统表中。具有数据库 DBA 权限的任何人都可查看这些信息，其中包括口令和加密证书。为避免发生这种潜在安全问题，可指定有关 dbmlsync 命令行的信息。

请参见“dbmlsync 语法”一节 《MobiLink - 客户端管理》。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “ALTER SYNCHRONIZATION USER 语句 [MobiLink]”一节第 372 页
- “DROP SYNCHRONIZATION USER 语句 [MobiLink]”一节第 558 页
- “加密 MobiLink 客户端/服务器通信”一节 《SQL Anywhere 服务器 - 数据库管理》
- “ISYSSYNC 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例将创建一个名为 SSinger 的 MobiLink 用户，此用户使用口令 Sam 通过 TCP/IP 协议与名为 mlserver.mycompany.com 的服务器计算机同步。在用户定义中使用口令是*不*安全的。

```
CREATE SYNCHRONIZATION USER SSinger
TYPE http
ADDRESS 'host=mlserver.mycompany.com'
OPTION MobiLinkPwd='Sam';
```

CREATE TABLE 语句

此语句用于在数据库中创建新表，也可以在远程服务器中创建表。

语法

CREATE [**GLOBAL TEMPORARY**] **TABLE** [**IF NOT EXISTS**] [*owner.*]*table-name*
({ *column-definition* | *table-constraint* | *pctfree* }, ... **)**
[{ **IN** | **ON** } *dbspace-name*]
[**ENCRYPTED**]
[**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS**
  | **NOT TRANSACTIONAL**]
[**AT** *location-string*]
[**SHARE BY ALL**]

column-definition :
column-name data-type
[**COMPRESSED**]
[**INLINE** { *inline-length* | **USE DEFAULT** }]
[**PREFIX** { *prefix-length* | **USE DEFAULT** }]

[[**NO**] **INDEX**]
[[**NOT**] **NULL**]
[**DEFAULT** *default-value* | **IDENTITY**]
[*column-constraint* ...]

default-value :
 special-value
| *string*
| *global variable*
| [**-**] *number*
| **(** *constant-expression* **)**
| *built-in-function***(** *constant-expression* **)**
| **AUTOINCREMENT**
| **CURRENT DATABASE**
| **CURRENT REMOTE USER**
| **CURRENT UTC TIMESTAMP**
| **GLOBAL AUTOINCREMENT** [**(** *partition-size* **)**]
| **NULL**
| **TIMESTAMP**
| **UTC TIMESTAMP**
| **LAST USER**

special-value:
CURRENT {
  DATE
  | **TIME**
  | **TIMESTAMP**
  | **UTC TIMESTAMP**
  | **USER**
  | **PUBLISHER**
}
| **USER**

column-constraint :
[**CONSTRAINT** *constraint-name*] {
  UNIQUE [**CLUSTERED**]
 | **PRIMARY KEY** [**CLUSTERED**] [**ASC** | **DESC**]
 | **REFERENCES** *table-name* [**(** *column-name* **)**]
    [**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]
    [*action-list*] [**CLUSTERED**]
  }
| [**CONSTRAINT** *constraint-name*] **CHECK (** *condition* **)**
| **COMPUTE (** *expression* **)**

table-constraint :
[**CONSTRAINT** *constraint-name*] {
    UNIQUE [**CLUSTERED**] **(** *column-name* [**ASC** | **DESC**], ... **)**
 | **PRIMARY KEY** [**CLUSTERED**] **(** *column-name* [**ASC** | **DESC**], ... **)**
 | **CHECK (** *condition* **)**
 | *foreign-key-constraint*
}

foreign-key-constraint :
[**NOT NULL**] **FOREIGN KEY** [*role-name*]
  [**(** *column-name* [**ASC** | **DESC**], ... **)**]
  REFERENCES *table-name*
  [**(** *column-name*, ... **)**]

[**MATCH** [**UNIQUE**] { **SIMPLE** | **FULL** }]
[*action-list*] [**CHECK ON COMMIT**] [**CLUSTERED**] [**FOR OLAP WORKLOAD**]

action-list :
[**ON UPDATE** *action*]
[**ON DELETE** *action*]

action :
CASCADE
| **SET NULL**
| **SET DEFAULT**
| **RESTRICT**

location-string :
remote-server-name.[*db-name*].[*owner*].*object-name*
| *remote-server-name*;[*db-name*];[*owner*];*object-name*

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

参数

- **IN 子句** 此子句用于指定基表所处的 dbspace。如果未指定此子句，则会在 default_dbspace 选项指定的 dbspace 中创建基表。

  临时表只能在临时 dbspace 中创建。如果创建 GLOBAL TEMPORARY 表，并指定 IN，则会在临时 dbspace 中创建该表。如果指定用户定义的 dbspace，则会返回错误。

  有关 dbspace 的详细信息，请参见：

  - “CREATE DBSPACE 语句”一节第 420 页
  - “使用附加 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “default_dbspace 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

- **ENCRYPTED 子句** ENCRYPTED 子句指定应对表加密。如果想要加密表，必须在创建数据库时启用表加密。将使用在数据库创建时间所指定的加密密钥和算法来加密表。请参见“启用数据库中的表加密”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **ON COMMIT 子句** ON COMMIT 子句仅适用于临时表。缺省情况下，临时表的行将在 COMMIT（提交）时被删除。如果指定了 SHARE BY ALL 子句，则必须指定 ON COMMIT PRESERVE ROWS 或 NOT TRANSACTIONAL。

- **NOT TRANSACTIONAL 子句** 创建全局临时表时允许使用 NOT TRANSACTIONAL 子句。使用 NOT TRANSACTIONAL 创建的表不受 COMMIT 或 ROLLBACK 中的任何一个的影响。如果指定了 SHARE BY ALL 子句，则必须指定 ON COMMIT PRESERVE ROWS 或 NOT TRANSACTIONAL。有关 NOT TRANSACTIONAL 子句用处的信息，请参见“使用临时表”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **AT 子句** 在 *location-string* 所指定的另一个服务器上创建一个远程表，同时在当前数据库中创建一个映射到该远程表的代理表。AT 子句支持分号 (;) 作为 *location-string* 中的字段分隔符。如果没有分号，则使用句号作为字段分隔符。利用此语法，便可在数据库和所有者字段中使用文件名和扩展名。

  例如，下列语句将表 a1 映射到 Microsoft Access 文件 *mydbfile.mdb*：

```
CREATE TABLE a1
AT 'access;d:\mydbfile.mdb;;a1';
```

有关远程服务器的信息，请参见“CREATE SERVER 语句”一节第 481 页。有关代理表的信息，请参见“CREATE EXISTING TABLE 语句”一节第 435 页和“指定代理表位置”一节《SQL Anywhere 服务器 - SQL 的用法》。

Windows Mobile 不支持 AT 子句。

在远程表上忽略外键定义。引用远程表的本地表的外键定义也被忽略。如果数据库服务器支持主键，则将主键定义发送到远程服务器。

- **SHARE BY ALL 子句** 此子句仅在创建全局临时表时使用，以允许服务器的所有连接都共享相应表。如果指定了 SHARE BY ALL 子句，则必须指定 ON COMMIT PRESERVE ROWS 或 NOT TRANSACTIONAL。

  有关临时表特性的信息，请参见“使用临时表”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **IF NOT EXISTS 子句** 此子句用于创建永久表、全局临时表和局部临时表。如果指定的表已存在，则不会进行任何更改，也不会返回错误。

  有关临时表特性的信息，请参见“使用临时表”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- ***column-definition*** 定义表中的列。以下是部分列定义。

  - ***column-name*** 列名是一个标识符。同一表中的两列不能同名。请参见“标识符”一节第 8 页。

  - ***data-type*** 存储在列中的数据类型。请参见“SQL 数据类型”第 75 页。

  - **COMPRESSED** 压缩列。例如，下面的语句将创建表 t，其中含有两个列：filename 和 contents。contents 列为 LONG BINARY 且已压缩：

    ```
    CREATE TABLE t (
      filename VARCHAR(255),
      contents LONG BINARY COMPRESSED
    );
    ```

- **INLINE 和 PREFIX 子句** INLINE 子句用于指定要存储在行中的 BLOB 大小上限（以字节为单位）。小于等于 INLINE 子句所指定值的 BLOB 存储在行内部。超出 INLINE 子句所指定值的 BLOB 将存储在行外部的表扩展页中。另外，当 BLOB 大于 INLINE 值时，BLOB 开头部分某些字节的副本会保留在行中。PREFIX 子句用于指定要保留在行中的字节数。对于需要利用 BLOB 的前缀字节数来确定是接受还是拒绝某行的请求来说，PREFIX 子句可提高这些请求的性能。

  压缩列的前缀数据未经压缩存储，因此如果满足请求所需的所有数据都存储在前缀中，则不必进行解压缩。

  如果 INLINE 和 PREFIX 都未指定，或指定了 USE DEFAULT，则如下所述应用缺省值：

  - 对于字符数据类型列（例如 CHAR、NCHAR 和 LONG VARCHAR），INLINE 的缺省值为 256，PREFIX 的缺省值为 8。

  - 对于二进制数据类型列（例如 BINARY、LONG BINARY、VARBINARY、BIT、VARBIT、LONG VARBIT、BIT VARYING 和 UUID），INLINE 的缺省值为 256，PREFIX 的缺省值为 0。

注意
强烈建议使用缺省值，除非出现需要另一种设置的特定情况。选择缺省值是为了在性能和磁盘空间要求间取得均衡。例如，如果将 INLINE 设置为较大值，并且所有 BLOB 都被内置存储，则行处理性能可能会下降。如果将 PREFIX 设置得过高，则要增加存储 BLOB 所需的磁盘空间量，因为前缀数据是由 BLOB 的一部分复制而来。

如果仅指定其中一个值，则另一个值会自动设置为不与指定值冲突的最大量。INLINE 和 PREFIX 值均不能超出数据库页面大小。此外，在表页中还要保留少量不可以用于存储行数据的开销。因此，如果指定的 INLINE 值近似于数据库页面大小，可导致要内置存储的字节数稍有减少。

- **INDEX 和 NO INDEX 子句** 存储 BLOB（仅字符或二进制类型）时，指定 INDEX 可对超过内部 BLOB 大小阈值（约八个数据库页）的插入值创建 BLOB 索引。这是缺省行为。

  当需要在 BLOB 内进行随机访问搜索时，BLOB 索引可提高性能。但对于某些类型的 BLOB 值（例如永远也不需要随机访问的映像和多媒体文件），关闭 BLOB 索引会提高性能。要关闭某列的 BLOB 索引，请指定 NO INDEX。

  注意
  BLOB 索引与表索引不同。创建表索引是为了给一个或多个列中的值建立索引。

- **NULL 和 NOT NULL 子句** 如果指定 NULL，则列中允许 NULL 值。这是缺省行为。

  如果指定 NOT NULL，则不允许 NULL 值。

  如果该列是 UNIQUE 或 PRIMARY KEY 约束的一部分，则即使指定 NULL，该列也不能包含 NULL。

- **DEFAULT 子句** 有关 *special-value* 的详细信息，请参见“特殊值”一节第 56 页。

  如果指定了 DEFAULT 值，则它将用作未指定列值的任何 INSERT 语句中的列值。如果未指定任何 DEFAULT 值，则它相当于 DEFAULT NULL。

  以下列出了 DEFAULT 的可能值：

  - **常量表达式** 在 DEFAULT 子句中允许使用不引用数据库对象的常量表达式，因此可以使用 GETDATE 或 DATEADD 之类的函数。如果表达式不是函数或简单值，则必须用圆括号将其括起来。
  - **CURRENT REMOTE USER** 当 CURRENT REMOTE USER 执行未提供文档的 REMOTE USER 语句时，由 SQL Remote 消息代理 (dbremote) 来设置其缺省值。除非从 DBREMOTE 进行连接，否则该值将为 NULL。请参见“消息代理 (dbremote)”一节 《SQL Remote》。
  - **AUTOINCREMENT** 使用 AUTOINCREMENT 时，列必须是整型数据类型之一或是精确的数值类型。

    插入到表中时，如果没有指定 AUTOINCREMENT 列的值，则生成一个比列中的任何其它值都大的唯一值。如果 INSERT 指定的列值大于列的当前最大值，则插入该值并将其用作后续插入的起点值。

删除行不会相应递减 AUTOINCREMENT 计数器的值。由于删除行而产生的间隙只能由使用插入时的显式赋值填充。显式插入小于该列最大值的列值后，后面没有显式赋值的行仍自动递增为比上一个最大值大 1 的值。

通过检查 @@identity 全局变量，可以找到最近一次插入的列值。请参见“@@identity 全局变量”一节第 68 页。

AUTOINCREMENT 值会以有符号的 64 位整数形式保留，对应于 SYSTABCOL 系统视图中 max_identity 列的数据类型。当要生成的下一个值超过可以存储在指派了 AUTOINCREMENT 的列中的最大值时，将返回 NULL。如果此列已声明不允许 NULL（与主键列相同），将生成 SQL 错误。

有关重建使用 AUTOINCREMENT 的数据库的信息，请参见“重装带有自动增量列的表”一节 《SQL Anywhere 11 - 更改和升级》。

- **IDENTITY** IDENTITY 缺省值是使用 AUTOINCREMENT 缺省值的替代方法，该方法与 Transact-SQL 兼容。在 SQL Anywhere 中，定义为 IDENTITY 的列按 AUTOINCREMENT 执行。请参见“特殊 IDENTITY 列”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **GLOBAL AUTOINCREMENT** 此缺省值专用于在 MobiLink 同步环境或 SQL Remote 复制中使用多个数据库的情况。

  此选项类似于 AUTOINCREMENT，只不过要对域进行分区。每个分区都包含相同数目的值。为每个数据库副本指定一个唯一全局数据库标识号。SQL Anywhere 只从用数据库编号唯一标识的分区中提供数据库中的缺省值。

  可以在紧跟 AUTOINCREMENT 关键字之后的圆括号内指定分区大小。分区大小可以为任意正整数，但分区大小的选择通常要保证任何一个分区内的编号资源尽量不被用尽（如果曾有过这种情况）。

  对于 BIGINT 或 UNSIGNED BIGINT 类型的列，缺省分区大小是 $2^{32} = 4294967296$；对于所有其它类型的列，缺省分区大小是 $2^{16} = 65536$。由于这些缺省值可能不合适（尤其当列不是 INT 或 BIGINT 类型时），因此最好显式指定分区大小。

  使用此缺省值时，每个数据库中的公共选项 global_database_id 的值必须设置为唯一的非负整数。该值唯一标识数据库，并指示从哪个分区分配缺省值。允许的值范围是 $np + 1$ 到 $p(n + 1)$，其中 n 是公共选项 global_database_id 的值，p 是分区大小。例如，如果将分区大小定义为 1000 并将 global_database_id 设置为 3，则该范围将是 3001 到 4000。

  如果上一个值小于 $p(n + 1)$，则下一个缺省值比列中的上一个最大值大 1。如果列不包含任何值，则第一个缺省值为 $np + 1$。缺省列值不受当前分区之外的列值的影响，即不受小于 $np + 1$ 或大于 $p(n + 1)$ 的数的影响。如果通过 MobiLink 或 SQL Remote 从另一个数据库复制了这种值，就可能存在这种值。

  通过检查 @@identity 全局变量，可以找到最近一次插入的列值。

  GLOBAL AUTOINCREMENT 值会以有符号的 64 位整数形式保留，对应于 SYSTABCOL 系统视图中 max_identity 列的数据类型。当分区中可以提供的值用完时，将返回 NULL。如果此列已声明不允许 NULL（与主键列相同），将生成 SQL 错误。在这种情况下，应为数据库指派一个新的 global_database_id 值，以便可以从另一个分区中选择缺省值。若要检测提供的未用值是否过小并处理此情况，请创建一个 GlobalAutoincrement 类型的事件。请参见“了解事件”一节 《SQL Anywhere 服务器 - 数据库管理》。

由于不能将公共选项 global_database_id 设置为负值，因此所选值始终是正数。最大标识号仅受列数据类型和分区大小的限制。

如果将公共选项 global_database_id 设置为缺省值 2147483647，则在列中插入 NULL 值。如果不允许使用 NULL 值，则尝试插入行时将出错。

- **TIMESTAMP 子句** 用于指明表中每行的上次修改时间。当用 DEFAULT TIMESTAMP 声明列时，会提供一个缺省的插入值，每当更新行时，该值都用当前日期和时间更新。

  要提供插入时的缺省值，而在每次更新时不改变，请使用 DEFAULT CURRENT TIMESTAMP 而不是 DEFAULT TIMESTAMP。

  有关 timestamp 列的详细信息，请参见“特殊的 Transact-SQL 时间戳列和数据类型”一节《SQL Anywhere 服务器 - SQL 的用法》。

  以 DEFAULT TIMESTAMP 声明的列包含唯一值，以便应用程序能检测到同一行几乎同时发生的更新。如果当前时间戳的值与上一个值相同，它将按 default_timestamp_increment 选项的值相应递增。请参见“default_timestamp_increment 选项 [数据库] [MobiLink 客户端]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  在 SQL Anywhere 中，您可以根据 default_timestamp_increment 选项自动截断时间戳的值。这对于同其它记录精度较低的时间戳值的数据库软件保持兼容很有用。请参见“default_timestamp_increment 选项 [数据库] [MobiLink 客户端]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  全局变量 @@dbts 返回一个 TIMESTAMP 值，该值表示上次使用 DEFAULT TIMESTAMP 为列生成的值。请参见“全局变量”一节第 65 页。

- **UTC TIMESTAMP 子句** 除了 UTC TIMESTAMP 值以协调通用时间 (UTC) 计算外，UTC TIMESTAMP 的行为与 TIMESTAMP 相同。

- **string** 请参见“字符串”一节第 9 页。

- **global-variable** 请参见“全局变量”一节第 65 页。

- **column-constraint 和 table-constraint 子句** 列约束和表约束有助于确保数据库中数据的完整性。如果语句会导致违反约束，则该语句的执行不会完成。该语句在检测到错误前所做的任何更改都被撤消并报告错误。可以创建两类约束：**检查约束**和**参照完整性（referential integrity，简称 RI）约束**。检查约束用于指定要放入数据库中的列值所必须满足的条件。RI 约束将在不同表中的数据间建立一种关系，除了为数据指定唯一性要求之外，还必须维护这种关系。

  有三种类型的 RI 约束：主键约束、外键约束和唯一约束。当创建 RI 约束（主键约束、外键约束或唯一约束）时，数据库服务器会通过在组成约束键的各列上隐式创建一个索引来强制实施该约束。该索引将按照指定方式在约束的键上创建。键由一组有序列以及每列的一序列值 (ASC/DESC) 组成。

  可以在列或表上指定约束。一般来说，列约束是引用表中某一列的约束，而表约束可以引用表中一个或多个列。

  - **PRIMARY KEY 约束** 主键用于唯一定义表中的每个行。主键由一个列或多个列组成。一个表只能有一个主键。在 *column-constraint* 子句中，指定 PRIMARY KEY 则表示该列是表的主键。在 *table-constraint* 中，可使用 PRIMARY KEY 子句指定在以指定顺序合并时组成表的主键的一个或多个列。

主键中列的排序不必与各个列的相应序号匹配。也就是说，列在主键中的物理顺序不必与在行中的物理顺序相同。此外，不能指定重复的列名。

当创建主键时，会自动创建主键的索引。可通过为每个列指定 ASC（升序）或 DESC（降序）来指定索引中各值的排序方式。还可以使用 CLUSTERED 关键字指定是否聚簇索引。有关 CLUSTERED 选项和聚簇索引的详细信息，请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

主键包括的列不能为 NULL。表的每一行均有一个唯一主键值。

建议不要将近似数据类型（例如 FLOAT 和 DOUBLE）用于主键。近似数值数据类型在算术运算后容易产生舍入误差。

- **Foreign key** 外键可将某组列的值限于与主键中的值匹配，或者与另一个表（主表）的唯一约束值匹配。例如，外键约束可用于确保 invoice 表中的客户号与 Customers 表中的客户号相对应。

  有关数据库服务器如何自动为外键选择列的信息，请参见“创建外键时忽略列名 (SQL)”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  外键列顺序不必反映出表中的列顺序。

  外键说明中不允许使用重复的列名。

  如果没有为 UPDATE 或 DELETE 操作指定任何操作，缺省 *action* 为 RESTRICT。

  当创建外键时，会自动创建外键的索引。可通过为每个列指定 ASC（升序）或 DESC（降序）来指定索引中各值的排序方式。还可以使用 CLUSTERED 关键字指定是否聚簇索引。有关 CLUSTERED 选项和聚簇索引的详细信息，请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  临时表不能有引用基表的外键，而基表不能有引用临时表的外键。

  - **NOT NULL 选项** 外键列中不允许使用 NULL。外键中的 NULL 表示主表中没有任何行与外表中的此行相对应。
  - **role-name 子句** 角色名是外键的名称。角色名的主要作用是区分同一表的两个外键。如果不指定角色名，则按如下方式分配角色名：

    1. 如果没有与表名同名的角色名外键，则将表名指派为角色名。
    2. 如果表名已被使用，则角色名为表名加上表的唯一 3 位零填充数字。

  - **REFERENCES 子句** 外键约束可使用 REFERENCES 列约束（仅限单个列）或 FOREIGN KEY 表约束来实现，使用后者时，约束可指定一个或多个列。如果在 REFERENCES 列约束中指定 *column-name*，则此列必须在主表中，必须受唯一约束或主键约束的制约，而且该约束必须仅包含这一列。如果未指定 *column-name*，则外键列引用主表的单个主键列。
  - **MATCH 子句** MATCH 子句可用于在使用多列外键时控制将何种情况视为匹配。它还允许您指定键的唯一性，从而免除了另外单独声明唯一性的必要。以下列出了可指定的匹配类型：

    - **UNIQUE 选项** 引用表对于非 NULL 键值只能有一项匹配（至少有一个非 NULL 列值的键具有隐式唯一性）。

- **SIMPLE 选项** 如果键中至少有一列为 NULL，或者所有列值都与被引用表某行中的对应列值匹配，则引用表中发生行匹配。
- **FULL 选项** 如果键中所有列值都为 NULL，或者所有列值都与被引用表某行中的值相匹配，则引用表中发生行匹配。
- **SIMPLE UNIQUE 选项** 如果同时满足 SIMPLE 和 UNIQUE 的条件，则发生匹配。
- **FULL UNIQUE 选项** 如果同时满足 FULL 和 UNIQUE 的条件，则发生匹配。

- **UNIQUE 约束** 在 *column-constraint* 子句中，UNIQUE 约束指定列中的值必须唯一。在 *table-constraint* 子句中，UNIQUE 约束标识一个或多个唯一标识表中每行的列。表中任何两行的值在所有指定的列中不能相同。一个表可以有一个以上的 UNIQUE 约束。

  UNIQUE 约束与唯一索引不同。唯一索引的列可以为 NULL，而 UNIQUE 约束中的列不能为 NULL。此外，外键可以引用主键或者 UNIQUE 约束，但不能引用唯一索引，因为唯一索引可以包含多个 NULL 实例。

  可以按任意顺序指定 UNIQUE 约束中的列。此外，可以通过为每列指定 ASC（升序）或 DESC（降序）来指定自动创建的相应索引中各值的排序方式。但不能指定重复的列名。

  建议不要将近似数据类型（例如 FLOAT 和 DOUBLE）用于具有唯一约束的列。近似数值数据类型在算术运算后容易产生舍入误差。

  还可以使用 CLUSTERED 关键字指定是否聚簇约束。有关 CLUSTERED 选项的详细信息，请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  有关唯一索引的信息，请参见“CREATE INDEX 语句”一节第 449 页。

- **CHECK 约束** 此约束允许对任意条件进行校验。例如，CHECK 约束可用于确保名为 Sex 的列只包含值 M 或 F。

  表中的任何行都不能违反 CHECK 约束。如果 INSERT 或 UPDATE 语句会导致行违反约束，则不允许执行相应操作并且撤消语句的作用。只有在 CHECK 约束条件求值结果为 FALSE 时才拒绝更改，如果 CHECK 约束条件求值结果为 TRUE 或 UNKNOWN，则允许更改。

  有关 TRUE、FALSE 和 UNKNOWN 条件的详细信息，请参见“NULL 值”一节第 71 页和 “搜索条件”一节第 33 页。

- **COMPUTE 子句** COMPUTE 子句仅供在 *column-constraint* 子句中使用。在使用 COMPUTE 子句创建列时，该列在任何行中的值都是所提供表达式的值。使用此约束创建的列对于应用程序为只读列：只要对行进行了修改，数据库服务器就会更改该值。COMPUTE 表达式不应返回非确定性值。例如，它不应包含特殊值（如 CURRENT TIMESTAMP）或非确定性函数。如果 COMPUTE 表达式返回一个非确定性值，则不能使用它来匹配查询中的表达式。请参见“使用计算列”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  忽略远程表的 COMPUTE 子句。

  任何尝试更改计算列的值的 UPDATE 语句都会触发与该列关联的所有触发器。

- **CHECK ON COMMIT 选项** CHECK ON COMMIT 选项将覆盖 wait_for_commit 数据库选项，使数据库服务器等到执行 COMMIT 之后再检查对外键的 RESTRICT 操作。CHECK ON COMMIT 选项不会延迟 CASCADE、SET NULL 或 SET DEFAULT 操作。

如果使用 CHECK ON COMMIT 但未指定任何操作，则将 RESTRICT 暗指为 UPDATE 和 DELETE 的操作。

- **FOR OLAP WORKLOAD 选项** 当在外键定义的 REFERENCES 子句中指定 FOR OLAP WORKLOAD 时，数据库服务器会执行某些优化并收集有关键的统计信息以帮助提高 OLAP 负载的性能，特别是当 optimization_workload 设置为 OLAP 时。请参见“optimization_workload 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  有关详细信息，请参见“OLAP 支持” 《SQL Anywhere 服务器 - SQL 的用法》。

- **PCTFREE 子句** 指定要为每个表页保留的可用空间百分比。如果数据更新时行大小增加，将占用可用空间。如果表页中没有可用空间，则该页上的行大小每次增加时，行都需要在多个表页中拆分，从而导致行碎片并可能引起性能下降。

  值 *percent-free-space* 是一个介于 0 和 100 之间的整数。0 表示每页上没有任何可用空间—每页均完全填满。如果值很高，会使每行单独插入到页中。如果未设置 PCTFREE，或稍后将其删除，则会根据数据库页面大小应用缺省的 PCTFREE 值（4 KB（及以上）的页面大小应用 200 字节）。PCTFREE 的值存储在系统表 ISYSTAB 中。

注释

CREATE TABLE 语句用于创建新表。通过指定所有者名称，可为其他用户创建表。如果指定 GLOBAL TEMPORARY，则表为临时表。否则，该表为基表。

通过在 CREATE TABLE 语句中的表名前加上井号 (#) 创建的表是已声明的临时表，这些表仅可用于当前连接。通过井号 (#) 创建的临时表与通过 ON COMMIT PRESERVE ROWS 子句创建的临时表相同。请参见“DECLARE LOCAL TEMPORARY TABLE 语句”一节第 529 页。

缺省情况下，SQL Anywhere 中的列允许使用 NULL 值。可使用 allow_nulls_by_default 数据库选项控制此设置。请参见“allow_nulls_by_default 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

必须具有 RESOURCE 权限。

必须有 DBA 权限才能为其他用户创建表。

副作用

自动提交。

另请参见

- “CREATE LOCAL TEMPORARY TABLE 语句”一节第 452 页
- “ALTER TABLE 语句”一节第 373 页
- “CREATE DBSPACE 语句”一节第 420 页
- “CREATE EXISTING TABLE 语句”一节第 435 页
- “DECLARE LOCAL TEMPORARY TABLE 语句”一节第 529 页
- “DROP TABLE 语句”一节第 558 页
- “特殊值”一节第 56 页
- “SQL 数据类型”第 75 页
- “创建表”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “allow_nulls_by_default 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “使用临时表”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心特性。

  以下为服务商扩充：

  - { IN | ON } *dbspace-name* 子句。
  - ON COMMIT 子句。
  - 一些缺省值。

示例

下面的示例为图书馆数据库创建用于保存图书信息的表。

```
CREATE TABLE library_books (
   -- NOT NULL is assumed for primary key columns
   isbn CHAR(20) PRIMARY KEY,
   copyright_date DATE,
   title CHAR(100),
   author CHAR(50),
   -- column(s) corresponding to primary key of room
   -- are created automatically
   FOREIGN KEY location REFERENCES room
);
```

以下示例为图书馆数据库创建用于保存借出图书信息的表。date_borrowed 的缺省值指示在创建条目的当天借出图书。date_returned 列在归还图书前一直为 NULL。

```
CREATE TABLE borrowed_book (
   date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
   date_returned DATE,
   book CHAR(20)
   REFERENCES library_books (isbn),
   -- The check condition is UNKNOWN until
   -- the book is returned, which is allowed
CHECK( date_returned >= date_borrowed )
);
```

以下示例将为某销售数据库创建保存订单和订单项信息的表。

```
CREATE TABLE Orders (
   order_num INTEGER NOT NULL PRIMARY KEY,
```

```
    date_ordered DATE,
    name CHAR(80)
);
CREATE TABLE Order_item (
    order_num INTEGER NOT NULL,
    item_num SMALLINT NOT NULL,
    PRIMARY KEY ( order_num, item_num ),
    -- When an order is deleted, delete all of its
    -- items.
    FOREIGN KEY ( order_num )
    REFERENCES Orders ( order_num )
    ON DELETE CASCADE
);
```

以下示例将在远程服务器 SERVER_A 上创建一个名为 t1 的表，并创建一个映射到该远程表的名为 t1 的代理表。

```
CREATE TABLE t1
( a INT,
  b CHAR(10) )
AT 'SERVER_A.db1.joe.t1';
```

CREATE TEXT CONFIGURATION 语句

创建文本配置对象。

语法

CREATE TEXT CONFIGURATION [*owner.*]*new-config-name*
FROM [*owner.*]*existing-config-name*

参数

- **FROM 子句** 指定文本配置对象的名称用作创建新文本配置对象的模板。缺省文本配置对象的名称是 default_char 和 default_nchar。请参见“缺省文本配置对象”一节 《SQL Anywhere 服务器 - SQL 的用法》。

注释

可以将另一文本配置对象用作模板来创建文本配置对象，然后使用 ALTER TEXT CONFIGURATION 语句根据需要更改选项。

要查看数据库中所有文本配置对象及其设置的列表，请查询 SYSTEXTCONFIG 系统视图。请参见“SYSTEXTCONFIG 系统视图”一节第 979 页。

权限

必须具有 DBA 或 RESOURCE 权限。

所有文本配置对象都具有 PUBLIC 访问权限。任何具有创建文本索引权限的用户都能够使用任意文本配置对象。

副作用

自动提交

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本配置对象设置”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “教程：对 GENERIC 文本索引执行全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “教程：执行模糊全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “ALTER TEXT CONFIGURATION 语句”一节第 381 页
- “DROP TEXT CONFIGURATION 语句”一节第 559 页
- “sa_refresh_text_indexes 系统过程”一节第 877 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下 CREATE TEXT CONFIGURATION 语句使用 default_char 文本配置对象创建文本配置对象 max_term_sixteen。后续 ALTER TEXT CONFIGURATION 语句将 max_term_sixteen 的最大术语长度更改为 16。

```
CREATE TEXT CONFIGURATION max_term_sixteen FROM default_char;
ALTER TEXT CONFIGURATION max_term_sixteen
   MAXIMUM TERM LENGTH 16;
```

CREATE TEXT INDEX 语句

创建文本索引。

语法

CREATE TEXT INDEX *text-index-name*
ON [*owner.*]*table-name***(** *column-name*, ... **)**
[**IN** *dbspace-name*]
[**CONFIGURATION** [*owner.*]*text-configuration-name*]
[{ **IMMEDIATE REFRESH**
  | **MANUAL REFRESH**
  | **AUTO REFRESH** [**EVERY** *integer* { **MINUTES** | **HOURS** }]]
  }

参数

- **ON 子句** 此子句用于指定在其上构建文本索引的表和列。
- **IN 子句** 此子句用于指定文本索引所处的 dbspace。如果未指定此子句，则会在 default_dbspace 选项指定的 dbspace 中创建文本索引。
- **CONFIGURATION 子句** 此子句用于指定创建文本索引时要使用的文本配置文件。在未指定此子句的情况下，如果索引中的任意列为 NCHAR，则使用 default_nchar 文本配置对象；否则，使用 default_char 文本配置对象。
- **REFRESH 子句** 此子句用于指定文本索引的刷新类型。如果未指定 REFRESH 子句，IMMEDIATE REFRESH 语句则作为缺省设置使用。以下是可指定的刷新类型的列表：

- **IMMEDIATE REFRESH** 指定 IMMEDIATE REFRESH 以便在每次基础表中的变化影响到文本索引中的数据时刷新文本索引。
- **AUTO REFRESH** 此子句用于通过内部服务器事件自动刷新实例化视图。使用 EVERY 次级子句来指定以分钟或小时为单位的刷新间隔。如果在未提供间隔信息的情况下指定 AUTO REFRESH，数据库服务器将每 60 分钟刷新一次文本索引。如果上次刷新时，sa_text_index_stats 系统过程返回的 pending_size 值超过文本索引大小的 20%，文本索引可能早于 AUTO REFRESH 子句指定的时间刷新。某个内部事件每分钟执行一次来针对所有 AUTO REFRESH 文本索引检查此条件。
- **MANUAL REFRESH** 此子句用于指定手动刷新文本索引。

有关刷新类型的详细信息，请参见“文本索引刷新类型”一节 《SQL Anywhere 服务器 - SQL 的用法》。

注释

不能在视图、实例化视图或临时表上创建文本索引。

IMMEDIATE REFRESH 文本索引会在创建时填充，并在此初始刷新过程中保存该表的独占锁。IMMEDIATE REFRESH 文本索引为使用快照隔离的查询提供完全支持。

MANUAL 和 AUTO REFRESH 文本索引必须在创建后进行初始化（刷新）。

AUTO REFRESH 文本索引的刷新使用隔离级别 0 扫描表。请参见“isolation_level 选项 [数据库] [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

文本索引创建后，不能将其更改到定义为 IMMEDIATE REFRESH，也不能将其从定义为 IMMEDIATE REFRESH 更改为其它值。如果要进行任何一种更改，必须先删除文本索引然后再重新创建。

可以选择使用 REFRESH TEXT INDEX 语句手动刷新 AUTO REFRESH 文本索引。请参见“REFRESH TEXT INDEX 语句”一节第 668 页。

要查看文本索引及其引用的文本配置对象，请参见“查看数据库中的文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

必须是基础表的所有者，或者具有 DBA 权限，或者具有 REFERENCES 权限。

当存在使用 WITH HOLD 子句打开的使用语句快照或事务快照的游标时，不能执行此语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

副作用

自动提交

另请参见

- “教程：对 GENERIC 文本索引执行全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “教程：执行模糊全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “SYSTEXTCONFIG 系统视图”一节第 979 页
- “CREATE TEXT INDEX 语句”一节第 509 页
- “ALTER TEXT INDEX 语句”一节第 383 页
- “DROP TEXT INDEX 语句”一节第 560 页
- “REFRESH TEXT INDEX 语句”一节第 668 页
- “TRUNCATE TEXT INDEX 语句”一节第 728 页
- “sa_char_terms 系统过程”一节第 798 页
- “sa_nchar_terms 系统过程”一节第 866 页
- “sa_refresh_text_indexes 系统过程”一节第 877 页
- “sa_text_index_stats 系统过程”一节第 906 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例在 demo 数据库中的 MarketingInformation 表的 Description 列上创建文本索引 myTxtIdx：使用 MarketingTextConfig 文本配置对象，并将刷新间隔设置为每 24 小时一次。

```
CREATE TEXT INDEX myTxtIdx ON MarketingInformation ( Description )
   CONFIGURATION MarketingTextConfig
   AUTO REFRESH EVERY 24 HOURS;
```

CREATE TRIGGER 语句

此语句用于在表中创建触发器。

语法

CREATE [**OR REPLACE**] **TRIGGER** *trigger-name trigger-type*
{ *trigger-event-list* | **UPDATE OF** *column-list* }
[**ORDER** *integer*] **ON** *table-name*
[**REFERENCING** [**OLD AS** *old-name*]
  [**NEW AS** *new-name*]
  [**REMOTE AS** *remote-name*]]
[**FOR EACH** { **ROW** | **STATEMENT** }]
[**WHEN (** *search-condition* **)**]
trigger-body

column-list : *column-name*[, ...]

trigger-type :
BEFORE
| **AFTER**
| **INSTEAD OF**
| **RESOLVE**

trigger-event-list : *trigger-event*[, ...]

trigger-event :
DELETE
| **INSERT**
| **UPDATE**

trigger-body :BEGIN 语句。请参见“BEGIN 语句”一节第 395 页。

参数

- **OR REPLACE 子句** 指定 OR REPLACE (CREATE OR REPLACE TRIGGER) 将创建一个新触发器或替换同名的现有触发器。
- ***触发器事件*** 触发器可以由以下事件触发。您可以为 DELETE、INSERT 或 UPDATE 事件定义多个触发器，或为 UPDATE OF *column-list* 事件定义一个触发器：
  - **DELETE** 每当删除关联表中的行时激活。
  - **INSERT** 每当有新行插入到与触发器关联的表中时激活。
  - **UPDATE** 每当更新关联表中的行时激活。
  - **UPDATE OF *column-list*** 每当更新关联表中的行和修改 *column-list* 中的列时就被调用。这种类型的触发器事件无法在 *trigger-event-list* 中使用；它必须是为触发器定义的唯一的触发器事件。此子句不能在 INSTEAD OF 触发器中使用。

    您可以为需要处理的每个事件分别编写触发器，如果您有一些共享操作和一些取决于事件的操作，也可以为所有事件创建一个触发器并使用 IF 语句辨别所发生的操作。有关触发操作的详细信息，请参见“触发器操作条件”一节第 52 页。
- ***trigger-type*** 可以将行级触发器定义为在插入、更新或删除操作之前 (BEFORE)、之后 (AFTER) 执行，或替代 (INSTEAD OF) 其中一个操作执行。可以将语句级触发器定义为替代 (INSTEAD OF) 语句执行或在语句之后 (AFTER) 执行。

  只要行上发生更新 (UPDATE)，BEFORE UPDATE 触发器就会触发，而无论新值是否与旧值不同。也就是说，如果已为 BEFORE UPDATE 触发器指定了 *column-list*，则 *column-list* 中的任何列出现在 UPDATE 语句的 SET 子句中都会导致该触发器触发。如果已为 AFTER UPDATE 触发器指定了 *column-list*，则只有在 *column-list* 中任意列的值被 UPDATE 语句*更改* 时触发器才触发。

  INSTEAD OF 触发器是可在常规视图上定义的唯一触发器形式。INSTEAD OF 触发器会用其它操作替代触发操作。当 INSTEAD OF 触发器触发时，会跳过触发操作而执行指定的操作。INSTEAD OF 触发器可在行级或语句级定义。语句级 INSTEAD OF 触发器会替换整个语句，包括全部行级操作。如果语句级 INSTEAD OF 触发器触发，该语句不会导致任何行级触发器的触发。但是，语句级触发器的主体可能会执行其它操作，从而导致其它行级触发器的触发。

  如果正在定义 INSTEAD OF 触发器，就无法使用 UPDATE OF *column-list* 子句、ORDER 子句或 WHEN 子句。

  有关 INSTEAD OF 触发器的功能和限制的详细信息，请参见“INSTEAD OF 触发器”一节《SQL Anywhere 服务器 - SQL 的用法》。

  RESOLVE 触发器类型适于和 SQL Remote 一同使用：它仅在行级 UPDATE 或 UPDATE OF *column-list* 前触发。

- **FOR EACH 子句** 要将触发器声明为行级触发器，请使用 FOR EACH ROW 子句。要将触发器声明为语句级触发器，可以使用 FOR EACH STATEMENT 子句或忽略 FOR EACH 子句。为保证清晰，建议在声明语句级触发器时指定 FOR EACH STATEMENT 子句。

- **ORDER 子句** 定义同时触发（之前、之后或解析）的相同类型的附加触发器（插入、更新或删除）时，必须指定一个 ORDER 子句来告知数据库服务器触发器触发的顺序。配置为在同时触发的相同类型触发器之间，顺序编号必须是唯一的。如果指定的顺序编号不是唯一的，则会返回错误。顺序编号不需要连续（例如可以指定 1、12、30）。数据库服务器从最低编号开始触发触发器。

  如果忽略 ORDER 子句或指定 0，数据库服务器会指派顺序 1。但是，如果另一相同类型触发器已设置为 1，则会返回错误。

  添加附加触发器时，可能需要修改事件的现有相同类型触发器，这取决于触发器的操作是否进行交互。如果它们不进行交互，新触发器的 ORDER 值必须高于现有触发器。如果它们进行交互，则需要考虑其它触发器，可能需要更改触发器触发的顺序。

  ORDER 子句不能在 INSTEAD OF 触发器中使用，因为在表或视图上只能定义每种类型（插入、更新或删除）的一个 INSTEAD OF 触发器。

- **REFERENCING 子句** REFERENCING OLD 和 REFERENCING NEW 子句允许您引用已插入、已删除或已更新的行。对于此子句而言，UPDATE 被视为删除后插入。

  INSERT 会使用 REFERENCING NEW 子句，该子句代表已插入的行。但不使用 REFERENCING OLD 子句。

  DELETE 会使用 REFERENCING OLD 子句，该子句代表已删除的行。但不使用 REFERENCING NEW 子句。

  UPDATE 用到 REFERENCING OLD 子句，代表更新前的行；并用到 REFERENCING NEW 子句，代表更新后的行。

  REFERENCING OLD 和 REFERENCING NEW 的含义因时而异，具体取决于触发器是行级还是语句级触发器。对于行级触发器，REFERENCING OLD 子句允许引用更新或删除之前行中的值，REFERENCING NEW 子句允许引用已插入或更新的值。在 BEFORE 和 AFTER 触发器中可以引用 OLD 和 NEW 行。REFERENCING NEW 子句允许在插入或更新操作发生之前在 BEFORE 触发器中修改新行。

  对于语句级触发器，REFERENCING OLD 和 REFERENCING NEW 子句引用保存行的新旧值的已声明临时表。这些表的缺省名是 deleted 和 inserted。

  REFERENCING REMOTE 子句适于和 SQL Remote 一同使用。它允许引用 UPDATE 语句的 VERIFY 子句中的值。它只能和 RESOLVE UPDATE 或 RESOLVE UPDATE OF 列列表触发器一同使用。

- **WHEN 子句** 仅对搜索条件求值结果为 true 的行触发触发器。WHEN 子句仅能用于行级触发器。此子句不能在 INSTEAD OF 触发器中使用。

- **触发器主体** 触发器主体包含触发操作发生时要执行的操作，由一个 BEGIN 子句组成。请参见“BEGIN 语句”一节第 395 页。

  可以在 BEGIN 语句中包含触发器操作条件。触发器操作条件将根据导致触发器触发的触发器事件来完成操作。例如，如果触发器定义为针对更新和删除都触发，则可以为这两种情况指定不

同的操作。有关触发器操作条件（包括示例）的详细信息，请参见“触发器操作条件”一节第 52 页。

注释

CREATE TRIGGER 语句创建与数据库中的表关联的触发器，并在数据库中存储触发器。

您不能在实例化视图上定义触发器。如果这样做，将返回 SQLE_INVALID_TRIGGER_MATVIEW 错误。

触发器声明为行级触发器（此情况下，它在每行修改之前或之后执行）或语句级触发器（此情况下，它在整个触发器语句完成后执行）。

权限

必须具有 RESOURCE 权限以及对表的 ALTER 权限，或者必须是表的所有者或具有 DBA 权限。CREATE TRIGGER 将对表进行表锁定，这样就需要表的独占使用权。

副作用

自动提交。

另请参见

- “BEGIN 语句”一节第 395 页
- “CREATE PROCEDURE 语句（Web 服务）”一节第 471 页
- “CREATE TRIGGER 语句 [T-SQL]”一节第 516 页
- “DROP TRIGGER 语句”一节第 561 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》
- “UPDATE 语句”一节第 735 页

标准和兼容性

- **SQL/2003** 持久存储模块特性。一些子句是服务商扩充。

示例

此示例将创建一个语句级触发器。您必须先创建一个表：

```
CREATE TABLE t0
( id integer NOT NULL,
 times timestamp NULL DEFAULT current timestamp,
 remarks text NULL,
 PRIMARY KEY ( id )
);
```

接下来，为此表创建一个语句级触发器：

```
CREATE TRIGGER myTrig AFTER INSERT ORDER 4 ON t0
REFERENCING NEW AS new_name
FOR EACH STATEMENT
BEGIN
  DECLARE @id1 INTEGER;
  DECLARE @times1 TIMESTAMP;
  DECLARE @remarks1 LONG VARCHAR;
  DECLARE @err_notfound EXCEPTION FOR SQLSTATE VALUE '02000';
//declare a cursor for table new_name
  DECLARE new1 CURSOR FOR
```



```
    SELECT id, times, remarks FROM new_name;
   OPEN new1;
  //Open the cursor, and get the value
   LoopGetRow:
   LOOP
       FETCH NEXT new1 INTO @id1, @times1,@remarks1;
       IF SQLSTATE = @err_notfound THEN
    LEAVE LoopGetRow
       END IF;
       //print the value or for other use
       PRINT (@remarks1);
   END LOOP LoopGetRow;
   CLOSE new1
END;
```

以下示例将替换在上一个示例中创建的 myTrig 触发器。

```
CREATE OR REPLACE TRIGGER myTrig AFTER INSERT ORDER 4 ON t0
REFERENCING NEW AS new_name
FOR EACH STATEMENT
BEGIN
FOR L1 AS new1 CURSOR FOR
   SELECT id, times, remarks FROM new_name
DO
     //print the value or for other use
     PRINT (@remarks1);
END FOR;
END;
```

下一个示例说明如何在 BEFORE UPDATE 触发器中使用 REFERENCING NEW。此示例确保新 Employees 表中的邮政编码均为大写形式：

```
CREATE TRIGGER emp_upper_postal_code
BEFORE UPDATE OF PostalCode
ON Employees
REFERENCING NEW AS new_emp
FOR EACH ROW
WHEN ( ISNUMERIC( new_emp.PostalCode ) = 0 )
BEGIN
   -- Ensure postal code is uppercase (employee might be
   -- in Canada where postal codes contain letters)
   SET new_emp.PostalCode = UPPER(new_emp.PostalCode)
END;

UPDATE Employees SET state='ON', PostalCode='n2x 4y7' WHERE EmployeeID=191;
SELECT PostalCode FROM Employees WHERE EmployeeID = 191;
```

下一个示例说明如何在 BEFORE DELETE 触发器中使用 REFERENCING OLD。此示例可防止从 Employees 表中删除尚未离职的雇员。

```
CREATE TRIGGER TR_check_delete_employee
BEFORE DELETE
ON Employees
REFERENCING OLD AS current_employees
FOR EACH ROW /* WHEN( search_condition ) */
BEGIN
   IF current_employees.TerminationDate IS NULL THEN
    RAISERROR 30001 'You cannot delete an employee who has not been fired';
    END IF;
END;
```

下一个示例说明如何在 BEFORE UPDATE 触发器中使用 REFERENCING NEW 和 REFERENCING OLD。此示例可防止降低雇员薪水。

```
CREATE TRIGGER TR_check_salary_decrease
    BEFORE UPDATE
      ON Employees
    REFERENCING OLD AS before_update
    NEW AS after_update
FOR EACH ROW
BEGIN
   IF after_update.salary < before_update.salary THEN
    RAISERROR 30002 'You cannot decrease a salary';
    END IF;
END;
```

下一个示例说明如何在 BEFORE UPDATE 触发器中使用 REFERENCING NEW 和 REFERENCING OLD。此示例也可以禁止降低雇员薪水，但是此触发器更有效，因为它只在更新薪水列时才触发。

```
CREATE TRIGGER TR_check_salary_decrease_column
    BEFORE UPDATE OF Salary
      ON Employees
    REFERENCING OLD AS before_update
    NEW AS after_update
FOR EACH ROW /* WHEN( search_condition ) */
BEGIN
   IF after_update.salary < before_update.salary THEN
    RAISERROR 30002 'You cannot decrease a salary';
End IF;
END;
```

下一个示例说明如何在 BEFORE INSERT 和 UPDATE 触发器中使用 REFERENCING NEW。以下示例会创建一个触发器，此触发器将在 SalesOrderItems 表中的行插入或更新前触发。

```
CREATE TRIGGER TR_update_date
   BEFORE INSERT, UPDATE
     ON SalesOrderItems
   REFERENCING NEW AS new_row
FOR EACH ROW
BEGIN
   SET new_row.ShipDate = CURRENT TIMESTAMP;
END;
```

CREATE TRIGGER 语句 [T-SQL]

此语句用于采用与 Adaptive Server Enterprise 兼容的方式，在数据库中创建新触发器。

语法 1

CREATE TRIGGER [*owner.*]*trigger_name*
ON [*owner.*]*table_name*
FOR { **INSERT**, **UPDATE**, **DELETE** }
AS *statement-list*

语法 2

CREATE TRIGGER [*owner.*]*trigger_name*
ON [*owner.*]*table_name*
FOR {**INSERT**, **UPDATE**}

AS
[**IF UPDATE (** *column_name* **)**
[{ **AND** | **OR** } **UPDATE (** *column_name* **)**] ...]
statement-list
[**IF UPDATE (** *column_name* **)**
[{ **AND** | **OR**} **UPDATE (** *column_name* **)**] ...]
statement-list

注释

删除或插入的行保存在两个临时表中。在 Transact-SQL 形式的触发器中，可以使用表名 deleted 和 inserted 访问它们，这与在 Adaptive Server Enterprise 中一样。在 Watcom-SQL CREATE TRIGGER 语句中，使用 REFERENCING 子句来访问这些行。

触发器名称在数据库中必须是唯一的。

Transact-SQL 触发器在触发器语句后 (AFTER) 执行。

权限

必须有 RESOURCE 权限和表的 ALTER 权限，或者必须有 DBA 权限。

CREATE TRIGGER 锁定表中的所有行，因此要求独占使用表。

副作用

自动提交。

另请参见

- “CREATE TRIGGER 语句”一节第 511 页

标准和兼容性

- **SQL/2003** 服务商扩充。

CREATE VARIABLE 语句

此语句用于创建 SQL 变量。

语法

CREATE VARIABLE *identifier data-type*

注释

CREATE VARIABLE 语句用于创建指定数据类型的新变量。此变量在被 SET 语句赋予另一个值之前包含的是 NULL 值。

在 SQL 表达式中，只要是允许使用列名的地方，就可以使用变量。名称解析按如下方式执行：

1. 与查询的 SELECT 列表中指定的任何别名进行匹配。
2. 与任何被引用表的列名进行匹配。
3. 假定名称为变量。

变量与当前连接关联，当与数据库断开连接或使用 DROP VARIABLE 语句时，变量就会消失。变量对其它连接不可见。变量不受 COMMIT 或 ROLLBACK 语句的影响。

在从嵌入式 SQL 程序为 INSERT 或 UPDATE 语句创建较大文本或二进制对象时，变量十分有用。

过程和触发器中的局部变量在复合语句中声明（请参见“使用复合语句”一节 《SQL Anywhere 服务器 - SQL 的用法》）。

权限

无。

副作用

无。

另请参见

- “BEGIN 语句”一节第 395 页
- “SQL 数据类型”第 75 页
- “DROP VARIABLE 语句”一节第 563 页
- “SET 语句”一节第 696 页
- “VAREXISTS 函数 [Miscellaneous]”一节第 326 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

此示例将创建一个名为 first_name 的变量，数据类型为 VARCHAR(50)。

```
CREATE VARIABLE first_name VARCHAR(50);
```

此示例将创建一个名为 'birthday' 的变量，数据类型为 DATE。

```
CREATE VARIABLE birthday DATE;
```

CREATE USER 语句

此语句用于创建用户。

语法

CREATE USER *user-name* [**IDENTIFIED BY** *password*]
[**LOGIN POLICY** *policy-name*]
[**FORCE PASSWORD CHANGE** { **ON** | **OFF** }]

参数

- **user-name** 所创建的用户的名称。
- **IDENTIFIED BY 子句** 所创建的用户的口令。没有口令的用户不能连接到数据库。

- **policy-name** 指派给用户的登录策略的名称。如果未指定登录策略，则应用 DEFAULT 登录策略。
- **FORCE PASSWORD CHANGE 子句** 控制用户登录时是否必须指定新口令。此设置将覆盖他们策略中的 password_expiry_on_next_login 选项设置。

注释

不必为用户指定口令。没有口令的用户不能连接到数据库。如果要创建组，但不希望任何人使用组用户 ID 连接到数据库，则这很有用。用户 ID 必须是有效的标识符。

用户 ID 和口令不能：

- 以空格、单引号或双引号开头
- 以空格结尾
- 含有分号

口令必须是有效的标识符，或者是用单引号引上的字符串（最大 255 字节）。口令区分大小写。建议采用由 7 位 ASCII 字符组成的口令，因为如果数据库服务器不能将客户端的字符集转换为 UTF-8，其它字符可能无法正常工作。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “CREATE LOGIN POLICY 语句”一节第 453 页
- “ALTER LOGIN POLICY 语句”一节第 357 页
- “ALTER USER 语句”一节第 385 页
- “COMMENT 语句”一节第 406 页
- “DROP LOGIN POLICY 语句”一节第 548 页
- “DROP USER 语句”一节第 562 页
- “管理登录策略概述”一节 《SQL Anywhere 服务器 - 数据库管理》
- “创建用户并为其分配登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》
- “GRANT 语句”一节第 595 页
- “min_password_length 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例创建一个名为 SQLTester 的用户，口令为 welcome。为 SQLTester 用户指派 Test1 登录策略，而且下次登录时口令到期。

```
CREATE USER SQLTester IDENTIFIED BY welcome
LOGIN POLICY Test1
FORCE PASSWORD CHANGE ON;
```

以下示例创建一个名为 MyGroup 的组

```
CREATE USER MyGroup;
GRANT GROUP TO MyGroup;
```

CREATE VIEW 语句

此语句用于在数据库上创建视图。视图用于以一种不同的视角展现数据，即使数据不是以这种方式存储。

语法

CREATE [**OR REPLACE**] **VIEW**
[*owner*.]*view-name* [**(** *column-name*, ... **)**]
AS *select-statement*
[**WITH CHECK OPTION**]

参数

- **OR REPLACE 子句** 指定 OR REPLACE (CREATE OR REPLACE VIEW) 将创建一个新视图或替换同名的现有视图。如果使用 OR REPLACE 子句，将保留现有权限。
- **AS 子句** 视图所基于的 SELECT 语句。SELECT 语句不能引用局部临时表。另外，SELECT 语句可带 ORDER BY 或 GROUP BY 子句，并且可以是 UNION。但是，在某些情况下（特别是当与 FIRST 或 TOP 子句结合使用时），使用带 ORDER BY 子句的 SELECT 的确会影响视图定义的结果。
- **WITH CHECK OPTION 子句** WITH CHECK OPTION 子句拒绝不满足 SELECT 语句定义的视图条件的任何视图更新和插入。

注释

CREATE VIEW 语句用于以给定名称创建视图。可以通过指定所有者来创建归另一个用户所有的视图。必须具有 DBA 权限才能为另一个用户创建视图。

在 SELECT、DELETE、UPDATE 和 INSERT 语句中，可以使用视图名取代表名。但在数据库中，视图实际上不是以表的形式存在。每次使用时它们就会派生。视图作为 CREATE VIEW 语句中指定的 SELECT 语句的结果派生。视图中使用的表名应该由表所有者的用户 ID 限定。否则，另一个用户 ID 可能无法找到该表或可能得到错误的表。

只要定义视图的 SELECT 语句不包含 GROUP BY 子句、集合函数或不涉及 UNION 子句，就可以更新视图。视图的更新会导致基础表更新。

视图中列的名称在 *column-name* 列表中指定。如果没有指定列名列表，则视图列名来自选择列表项。选择列表中的所有项都必须具有唯一名称。若要使用选择列表项中的名称，每项必须是一个简单列名或指定了别名。请参见“SELECT 语句”一节第 689 页。

通常，视图会引用在目录中定义的表和视图（及其各自的属性）。但视图还可以引用 SQL 变量。在这种情况下，当执行引用视图的查询时，将使用 SQL 变量的值。引用 SQL 变量的视图称为**参数化视图**，因为这些变量将充当视图执行过程中的参数。

除了在查询中嵌入对等的 SELECT 块主体以作为查询的 FROM 子句中的派生表之外，参数化视图提供了另一种替代方法。参数化视图对于存储过程中内嵌的查询尤为有用，其中，视图中所引用的 SQL 变量即是过程的输入参数。

执行 CREATE VIEW 语句时，不需要 SQL 变量存在。但如果在执行引用该视图的查询时未定义 SQL 变量，则会返回 [**未找到列**] 错误。

权限

必须有 RESOURCE 权限和视图定义中的表的 SELECT 权限。

副作用

自动提交。

另请参见

- “CREATE VIEW 语句”一节第 520 页
- “CREATE TABLE 语句”一节第 497 页
- “CREATE MATERIALIZED VIEW 语句”一节第 455 页

标准和兼容性

- **SQL/2003** 核心特性。但参数化视图是服务商扩充。

示例

下面的示例将创建一个仅显示男雇员信息的视图。此视图与基表采用相同的列名。

```
CREATE VIEW MaleEmployees
AS SELECT *
FROM Employees
WHERE Sex = 'M';
```

以下示例将创建一个显示雇员及其所属部门的视图。

```
CREATE VIEW EmployeesAndDepartments
  AS SELECT Surname, GivenName, DepartmentName
  FROM Employees JOIN Departments
  ON Employees.DepartmentID = Departments.DepartmentID;
```

以下示例将替换在上一个示例中创建的 EmployeesAndDepartments 视图。在替换视图后，视图中将显示每位员工所在的城市、州和国家（地区）位置：

```
CREATE OR REPLACE VIEW EmployeesAndDepartments
  AS SELECT Surname, GivenName, City, State, Country
  FROM Employees JOIN Departments
  ON Employees.DepartmentID = Departments.DepartmentID;
```

以下示例将基于变量 var1 和 var2 创建一个参数化视图，这两个变量既不是 Employees 表也不是 Departments 表的属性：

```
CREATE VIEW EmployeesByState
  AS SELECT Surname, GivenName, DepartmentName
  FROM Employees JOIN Departments
  ON Employees.DepartmentID = Departments.DepartmentID
  WHERE Employees.State = var1 and Employees.Status = var2;
```

在变量为允许的表达式的上下文中，变量可以出现在视图的 SELECT 语句中。例如，下面的参数化视图将参数 var1 用作 LIKE 谓语的模式：

```
CREATE VIEW ProductsByDescription
  AS SELECT *
  FROM Products
  WHERE Products.Description LIKE var1;
```

若要使用此视图，则必须在执行引用视图的查询前定义变量 var1。例如，可以将以下内容放置在过程、函数或批处理语句中：

```
BEGIN
DECLARE var1 CHAR(20);
SET var1 = '%cap%';
SELECT * FROM ProductsByDescription
END
```

DEALLOCATE 语句

此语句在 SQL Anywhere 中无效，并且将被忽略。提供此选项是为了与 Adaptive Server Enterprise 和 Microsoft SQL Server 兼容。有关此语句的详细信息，请参见 Adaptive Server Enterprise 或 Microsoft SQL Server 的相关文档。

标准和兼容性

- **SQL/2003** 服务商扩充。

DEALLOCATE DESCRIPTOR 语句 [ESQL]

此语句用于释放与 SQL 描述符区关联的内存。

语法

DEALLOCATE DESCRIPTOR *descriptor-name*

descriptor-name : *identifier*

注释

释放与描述符区关联的所有内存，包括数据项、指示符变量和结构本身。

权限

无。

副作用

无。

另请参见

- “ALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 343 页
- “SQL 描述符区域 (SQLDA)”一节 《SQL Anywhere 服务器 - 编程》
- “SET DESCRIPTOR 语句 [ESQL]”一节第 701 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

有关示例内容，请参见“ALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 343 页。

声明部分 [ESQL]

此语句用于在嵌入式 SQL 程序中声明主机变量。主机变量用于与数据库交换数据。

语法

EXEC SQL BEGIN DECLARE SECTION;
C declarations
EXEC SQL END DECLARE SECTION;

注释

声明部分只不过是由 BEGIN DECLARE SECTION 和 END DECLARE SECTION 语句括起来的 C 变量声明部分。声明部分使 SQL 处理器可以识别用作主机变量的 C 变量。并非所有的 C 声明在声明部分内都有效。请参见“使用主机变量”一节 《SQL Anywhere 服务器 - 编程》。

权限

无。

另请参见

- “BEGIN 语句”一节第 395 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

```
EXEC SQL BEGIN DECLARE SECTION;
char *surname, initials[5];
int dept;
EXEC SQL END DECLARE SECTION;
```

DECLARE 语句

此语句用于在复合语句 (BEGIN ... END) 中声明 SQL 变量。

语法

DECLARE *variable-name data-type*

注释

在过程、触发器或批处理主体中使用的变量可以使用 DECLARE 语句声明。该变量在声明它的复合语句的持续时间内会一直保留。

Watcom-SQL 过程或触发器的主体是复合语句，并且必须使用紧跟 BEGIN 关键字之后的其它声明（例如游标声明 (DECLARE CURSOR)）来声明变量。在 Transact-SQL 过程或触发器中没有这种限制。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “DECLARE CURSOR 语句 [T-SQL]”一节第 528 页

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

下面的批处理语句说明了 DECLARE 语句的用法，并在数据库服务器消息窗口中输出一条消息：

```
BEGIN
  DECLARE varname CHAR(61);
  SET varname = 'Test name';
  MESSAGE varname;
END
```

DECLARE CURSOR 语句 [ESQL] [SP]

此语句用于声明游标。游标是处理查询结果的主要方法。

语法 1 [ESQL]

DECLARE *cursor-name*
[**UNIQUE**]
[**NO SCROLL**
| **DYNAMIC SCROLL**
| **SCROLL**
| **INSENSITIVE**
| **SENSITIVE**
]
CURSOR FOR
{ *select-statement*
| *statement-name* [**FOR** { **UPDATE** [*cursor-concurrency*] | **READ ONLY** }]
| *call-statement* }

语法 2 [SP]

DECLARE *cursor-name*
[**NO SCROLL**
| **DYNAMIC SCROLL**


```
  | SCROLL
  | INSENSITIVE
  | SENSITIVE
]
CURSOR
{ FOR  select-statement [ FOR { UPDATE [ cursor-concurrency ] | READ ONLY } ]
| FOR call-statement
| USING variable-name }

cursor-name :  identifier

statement-name :  identifier | hostvar

variable-name :  identifier

cursor-concurrency :
BY { VALUES | TIMESTAMP | LOCK }
```

参数

- **UNIQUE 子句**　当游标被声明为 UNIQUE 时，强制查询返回唯一标识每行所需的所有列。这通常意味着确保返回主键或唯一性表约束中的所有列。任何需要的但未在查询中指定的列都将添加到结果集中。

  对 UNIQUE 游标执行的 DESCRIBE 会在指示符变量中设置以下附加选项：

  - **DT_KEY_COLUMN**　该列是行键的一部分
  - **DT_HIDDEN_COLUMN**　该列被添加到查询中，因为唯一标识行时需要使用它

- **NO SCROLL 子句**　声明为 NO SCROLL 的游标仅限于使用 FETCH NEXT 和 FETCH RELATIVE 0 查找操作在结果集中前进。

  由于一旦游标离开行，行就不能返回，所以对游标没有敏感性限制。当请求 NO SCROLL 游标时，SQL Anywhere 提供效率最高的游标，即敏感性未定型游标。请参见“敏感性未定型游标”一节 《SQL Anywhere 服务器 - 编程》。

- **DYNAMIC SCROLL 子句**　DYNAMIC SCROLL 是缺省游标类型。DYNAMIC SCROLL 游标可以使用 FETCH 语句的所有形式。

  当请求 DYNAMIC SCROLL 游标时，SQL Anywhere 提供敏感性未定型游标。当使用游标时，总是存在效率与一致性的平衡问题。敏感性未定型游标以一致性为代价提供高效性能。请参见“敏感性未定型游标”一节 《SQL Anywhere 服务器 - 编程》。

- **SCROLL 子句**　声明了 SCROLL 的游标可以使用 FETCH 语句的所有形式。当请求 SCROLL 游标时，SQL Anywhere 提供对值敏感的游标。请参见“对值敏感的游标”一节 《SQL Anywhere 服务器 - 编程》。

  SQL Anywhere 必须以保证结果集成员资格的方式执行对值敏感的游标。DYNAMIC SCROLL 游标更高效，应加以使用，除非要求 SCROLL 游标的一致行为。

- **INSENSITIVE 子句**　声明为 INSENSITIVE 的游标在打开时修复其成员资格；系统会使用所有原始行的副本创建一个临时表。如果对 INSENSITIVE 游标执行 FETCHING 操作，则看不到其它任何 INSERT、UPDATE 或 DELETE 语句的效果，也看不到另一个游标上的其它任何 PUT、UPDATE WHERE CURRENT、DELETE WHERE CURRENT 操作的效果。它看不到对同一游标

执行 PUT、UPDATE WHERE CURRENT、DELETE WHERE CURRENT 操作的效果。请参见“不敏感游标”一节 《SQL Anywhere 服务器 - 编程》。

- **SENSITIVE 子句** 声明为 SENSITIVE 的游标对结果集的成员资格或值的更改敏感。请参见“敏感游标”一节 《SQL Anywhere 服务器 - 编程》。
- **FOR statement-name** 使用 PREPARE 语句为各语句命名。只能为预准备 SELECT 或 CALL 声明游标。
- **FOR UPDATE | READ ONLY** 声明为 FOR READ ONLY 的游标不能在 UPDATE（定位）或 DELETE（定位）操作中使用。FOR UPDATE 是缺省设置。

  对于不带 ORDER BY 子句的单表查询，或者在 ansi_update_constraints 选项设置为 Off 的情况下，游标的缺省设置为 FOR UPDATE。如果 ansi_update_constraints 选项设置为 Cursors 或 Strict，则停留在包含 ORDER BY 子句的查询上的游标的缺省设置为 READ ONLY。但可使用 FOR UPDATE 子句将游标显式标记为可更新。由于允许通过 ORDER BY 子句或连接来更新游标的开销大，因此，停留在包含两个或更多表连接的查询上的游标为 READ ONLY，并且无法设置为可更新。

  为响应指定 FOR UPDATE 的游标的任何请求，SQL Anywhere 提供了对值敏感的游标或敏感性未定型的游标。不敏感的游标或者敏感性未定型游标不能更新。

- **USING variable-name** 仅限于在存储过程中使用。此变量是包含游标 SELECT 语句的字符串。当处理 DECLARE 时，此变量必须可用，因此必须是以下项之一：

  过程的参数。例如，

```
CREATE FUNCTION GetRowCount( IN qry LONG VARCHAR)
RETURNS INT
BEGIN
  DECLARE crsr CURSOR USING qry;
  DECLARE rowcnt INT;

  SET rowcnt = 0;
  OPEN crsr;
  lp: LOOP
    fetch crsr;
    IF SQLCODE <> 0 THEN LEAVE lp END IF;
    SET rowcnt = rowcnt + 1;
  END LOOP;
  RETURN rowcnt;
END;
```

  变量被赋值后，嵌套在另一个 BEGIN...END 内。例如，

```
CREATE PROCEDURE get_table_name(
  IN id_value INT, OUT tabname CHAR(128)
)
BEGIN
  DECLARE qry LONG VARCHAR;

  SET qry = 'SELECT table_name FROM SYS.SYSTAB ' ||
            'WHERE table_id=' || string(id_value);
  BEGIN
    DECLARE crsr CURSOR USING qry;
    OPEN crsr;
    FETCH crsr INTO tabname;
    CLOSE crsr;
```



```
    END
  END;
```

- **BY VALUES | TIMESTAMP | LOCK 子句** 在嵌入式 SQL 中，可通过在 SELECT 语句本身或游标声明中包括语法来设置并发说明。通过带有 DECLARE CURSOR 或 FOR 语句的选项，或特定编程接口中的并发设置 API，可以在游标级选择保守或优化的并发。如果语句可更新，并且游标未指定特定并发控制机制，则将使用语句的说明。语法如下：
- **FOR UPDATE BY LOCK 子句** 数据库服务器在结果集的 FETCHed（已读取）行上获得意图行锁。这些锁是长期锁，会一直保留到事务 COMMIT 或 ROLLBACK 执行。
- **FOR UPDATE BY { VALUES | TIMESTAMP }** 数据库服务器利用了由键集决定的游标，这样，如果在滚动结果集过程中修改或删除了行，则会通过该游标通知应用程序。

注释

DECLARE CURSOR 语句用 SELECT 语句或 CALL 语句的指定名称声明游标。在 Watcom-SQL 过程、触发器或批处理中，DECLARE CURSOR 语句必须与其它声明一同出现，且紧跟在 BEGIN 关键字之后。此外，游标名必须唯一。

如果某游标是在复合语句内声明的，则它仅在该复合语句的持续时间内存在（无论它是在 Watcom-SQL 还是在 Transact-SQL 复合语句中声明）。

权限

无。

副作用

无。

另请参见

- “PREPARE 语句 [ESQL]”一节第 657 页
- “OPEN 语句 [ESQL] [SP]”一节第 647 页
- “EXPLAIN 语句 [ESQL]”一节第 573 页
- “SELECT 语句”一节第 689 页
- “CALL 语句”一节第 400 页
- “FOR 语句”一节第 577 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

下面的示例说明了如何在嵌入式 SQL 中声明滚动游标：

```
EXEC SQL DECLARE cur_employee SCROLL CURSOR
FOR SELECT * FROM Employees;
```

以下示例说明了如何在嵌入式 SQL 中为预准备语句声明游标：

```
EXEC SQL PREPARE employee_statement
FROM 'SELECT Surname FROM Employees';
EXEC SQL DECLARE cur_employee CURSOR
FOR employee_statement;
```


以下示例说明了如何在存储过程中使用游标：

```
BEGIN
  DECLARE cur_employee CURSOR FOR
      SELECT Surname
      FROM Employees;
  DECLARE name CHAR(40);
  OPEN cur_employee;
  lp: LOOP
    FETCH NEXT cur_employee INTO name;
    IF SQLCODE <> 0 THEN LEAVE lp END IF;
    ...
  END LOOP;
  CLOSE cur_employee;
END
```

DECLARE CURSOR 语句 [T-SQL]

此语句用于以与 Adaptive Server Enterprise 兼容的方式声明游标。

语法

DECLARE *cursor-name*
CURSOR FOR *select-statement*
[**FOR** { **READ ONLY** | **UPDATE** }]

cursor-name : *identifier*

select-statement : *string*

注释

Transact-SQL 过程中的 DECLARE CURSOR 语句被视为可执行语句，且可以在过程中的任何位置出现。游标声明在执行该语句时生效，有效性一直保持到执行 DEALLOCATE CURSOR 语句或过程完成时为止。

在 SQL Anywhere 中，如果游标是在复合语句内声明的，则它仅在该复合语句的持续时间内存在（无论它是在 Watcom-SQ 还是在 Transact-SQL 复合语句中声明）。

在 Transact-SQL 过程、触发器或批处理中，DECLARE CURSOR 语句可以出现在其它可执行语句之后。

权限

无。

副作用

无。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页

标准和兼容性

- **SQL/2003** 核心特性。FOR UPDATE 和 FOR READ ONLY 选项是 Transact-SQL 扩展。

DECLARE LOCAL TEMPORARY TABLE 语句

此语句用于声明局部临时表。

语法

DECLARE LOCAL TEMPORARY TABLE *table-name*
({ *column-definition* [*column-constraint* ...] | *table-constraint* | *pctfree* }, ... **)**
[**ON COMMIT** { **DELETE** | **PRESERVE** } **ROWS**
| **NOT TRANSACTIONAL**]

pctfree : **PCTFREE** *percent-free-space*

percent-free-space : *integer*

参数

有关 *column-definition*、*column-constraint*、*table-constraint* 和 *pctfree* 的定义，请参见“CREATE TABLE 语句”一节第 497 页。

- **ON COMMIT 子句** 缺省情况下，临时表的行在执行 COMMIT 时删除。可以使用 ON COMMIT 子句在执行 COMMIT 时保留行。
- **NOT TRANSACTIONAL 子句** 使用此子句创建的表不受 COMMIT 或 ROLLBACK 两者中任一个影响。在某些情况下，NOT TRANSACTIONAL 子句可以提高性能，因为对非事务性临时表执行的操作不会导致在回退日志中生成条目。例如，在反复调用使用临时表的过程而不对 COMMIT 或 ROLLBACK 进行干预时，NOT TRANSACTIONAL 可能会非常有用。

注释

DECLARE LOCAL TEMPORARY TABLE 语句用于声明临时表。

当声明的临时表被显式删除或超出范围时，该表的行将被删除。也可以使用 TRUNCATE 或 DELETE 显式删除行。

复合语句中声明的局部临时表存在于复合语句中。（请参见“使用复合语句”一节 《SQL Anywhere 服务器 - SQL 的用法》）。否则，在连接断开前，声明的局部临时表会一直存在。

如果想要过程创建一个在过程完成后仍然存在的局部临时表，则改为使用 CREATE LOCAL TEMPORARY TABLE 语句。（请参见“CREATE LOCAL TEMPORARY TABLE 语句”一节第 452 页）。

权限

无。

副作用

无。

另请参见

- “CREATE TABLE 语句”一节第 497 页
- “CREATE LOCAL TEMPORARY TABLE 语句”一节第 452 页
- “使用复合语句”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL/基础特性。

示例

下面的示例阐释如何在存储过程中声明临时表：

```
BEGIN
  DECLARE LOCAL TEMPORARY TABLE TempTab ( number INT );
  ...
END
```

DELETE 语句

此语句用于从数据库中删除行。

语法

DELETE [*row-limitation*]
[**FROM**] [*owner.*]*table-expression*
[**FROM** *table-list* [,...]]
[**WHERE** *search-condition*]
[**ORDER BY** { *expression* | *integer* } [**ASC** | **DESC**], ...]
[**OPTION(** *query-hint*, ... **)**]

table-list :
table-name [,...]

table-name :
[*owner.*]*base-table-name* [[**AS**] *correlation-name*]
| [*owner.*]*view-name* [[**AS**] *correlation-name*]
| *derived-table*

derived-table :
(*select-statement* **)**
[**AS**] *correlation-name* [**(** *column-name* [,...] **)**]

row-limitation :
FIRST | **TOP** *n* [**START AT** *m*]

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| **FORCE NO OPTIMIZATION**
| *option-name* = *option-value*

table-expression：完整的表表达式可包含连接。请参见“FROM 子句”一节第 582 页。

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

参数

- **row-limitation 子句** 行限制子句允许您只返回满足 WHERE 子句的行的子集。TOP 和 START AT 值可以是主机变量、整型常量或整型变量。TOP 值必须大于等于 0。START AT 值必须大于 0。通常，在指定这些子句时也指定 ORDER BY 子句，这样可以用一种有意义的方式对这些行进行排序。请参见“显式限制查询返回的行数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **FROM 子句** FROM 子句指明要从中删除行的表。DELETE 语句中的第二个 FROM 子句将基于连接限定要从指定表中删除的行。如果存在第二个 FROM 子句，则 WHERE 子句限定这第二个 FROM 子句的行。

  FROM *table-expression* 子句允许根据连接来进行删除。*table-expression* 可以包含任意复杂表的表达式，如 KEY 和 NATURAL 连接。有关 FROM 子句和连接的完整说明，请参见“FROM 子句”一节第 582 页。

  下面的语句阐释 DELETE 语句（有两个使用相关名的 FROM 子句）的表名中潜在的不明确性：

  ```
  DELETE
  FROM table_1
  FROM table_1 AS alias_1, table_2 AS alias_2
  WHERE ...
  ```

  表 table_1 在第一个 FROM 子句中没有相关名标识，而在第二个 FROM 子句中具有相关名标识。在本例中，第一个子句中的 table_1 在第二个子句中用 alias_1 标识—此语句中只有一个 table_1 实例。这是一般规则的特例，即一个表在同一语句中的两个实例，一次用相关名标识另一次没有相关名。

  但以下示例中，第二个 FROM 子句中有两个 table_1 的实例。此语句因语法错误而失败，因为不明确第二个 FROM 子句中的哪个 table_1 实例与第一个 FROM 子句中的第一个 table_1 实例匹配。

  ```
  DELETE
  FROM table_1
  FROM table_1 AS alias_1, table_1 AS alias_2
  WHERE ...
  ```

- **WHERE 子句** DELETE 语句将删除满足 WHERE 子句中的条件的所有行。如果没有指定 WHERE 子句，则删除指定表中的所有行。如果存在第二个 FROM 子句，则 WHERE 子句可限定第二个 FROM 子句的各行。

- **ORDER BY 子句** 指定行的排序顺序。通常情况下，行以什么顺序更新并不重要。但与 FIRST 或 TOP 子句联用时，顺序就非常重要。

  不能在 ORDER BY 子句中使用列序号。

  ORDER BY 列表中的每一项均可标记为 ASC 以按升序排序（缺省值），或标记为 DESC 以按降序排序。

- **OPTION 子句** 此子句用于指定执行语句时的提示。支持以下提示：
  - MATERIALIZED VIEW OPTIMIZATION *option-value*
  - FORCE OPTIMIZATION
  - FORCE NO OPTIMIZATION
  - *option-name* = *option-value*

  有关这些选项的说明，请参见“SELECT 语句”一节第 689 页的 OPTIONS 子句。

注释

使用 DELETE 语句删除大量数据会导致对列统计信息的更新。

如果定义视图的 SELECT 语句在 FROM 子句中只有一个表并且不包含 GROUP BY 子句、集合函数或不涉及 UNION 子句，则可以在视图上使用 DELETE 语句。

如果定义视图的查询说明是可更新的，可对视图执行 DELETE。有关识别固有*不可更新的*视图的详细信息，请参见“使用常规视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

必须有表的 DELETE 权限。

副作用

无。

另请参见

- “TRUNCATE 语句”一节第 727 页
- “INSERT 语句”一节第 616 页
- “INPUT 语句 [Interactive SQL]”一节第 610 页
- “FROM 子句”一节第 582 页
- “删除过程中的锁定”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心特性。在 FROM 子句中使用一个以上的表是服务商扩充。

示例

从 FinancialData 表中删除 2000 年之前的所有数据。

```
DELETE
FROM FinancialData
WHERE Year < 2000;
```

从 SalesOrderItems 表中删除发货日期早于 2001 年 1 月 1 日且其地区为 Central 的前 10 张订单。

```
DELETE TOP 10
FROM SalesOrderItems
FROM SalesOrders
WHERE SalesOrderItems.ID = SalesOrders.ID
  and ShipDate < '2001-01-01' and Region ='Central'
ORDER BY ShipDate ASC;
```

以隔离级别 3 执行此语句，将部门 600 从数据库中删除。


```
DELETE FROM Departments
WHERE DepartmentID = 600
OPTION( isolation_level = 3 );
```

DELETE（定位）语句 [ESQL] [SP]

此语句用于从游标的当前位置删除数据。

语法

DELETE [**FROM** *table-spec*] **WHERE CURRENT OF** *cursor-name*

cursor-name : *identifier* | *hostvar*

table-spec : [*owner*.]*correlation-name*

owner : *identifier*

注释

这种形式的 DELETE 语句用于删除指定游标的当前行。当前行被定义为从游标读取的最后一行。

从中删除行的表由以下确定：

- 如果不包括 FROM 子句，则游标必须仅在一个表中。
- 如果游标用于连接式查询（包括使用含有连接的视图），则必须使用 FROM 子句。只会删除指定表的当前行，连接中涉及的其它表不会受到影响。
- 如果包含 FROM 子句但未指定表所有者，则 *table-spec* 首先与任何相关名进行匹配。
  - 如果已有相关名，则 *table-spec* 以该相关名标识。
  - 如果相关名不存在，则 *table-spec* 必须可明确标识为游标中的表名。
- 如果包含 FROM 子句并且指定了表所有者，则 *table-spec* 必须可明确标识为游标中的表名。
- 已定位的 DELETE 语句可以用于在某视图上打开的游标，只要该视图可更新即可。

权限

必须有游标中使用的表的 DELETE 权限。

副作用

无。

另请参见

- “UPDATE 语句”一节第 735 页
- “UPDATE（定位）语句 [ESQL] [SP]”一节第 740 页
- “INSERT 语句”一节第 616 页
- “PUT 语句 [ESQL]”一节第 660 页

标准和兼容性

- **SQL/2003** 核心特性。如果 ansi_update_constraints 选项设置为 Off，则可更新的游标的范围可能包含服务商扩充。

示例

下面的语句从数据库中删除当前行。

```
DELETE
WHERE CURRENT OF cur_employee;
```

DESCRIBE 语句 [ESQL]

此语句用于获取有关存储从数据库中检索的数据所需的主机变量的信息，以及向数据库传递数据所需的主机变量的信息。

语法

DESCRIBE
[**USER TYPES**]
[**ALL** | **BIND VARIABLES FOR** | **INPUT** | **OUTPUT**
| **SELECT LIST FOR**]
[**LONG NAMES** [*long-name-spec*] | **WITH VARIABLE RESULT**]
[**FOR**] { *statement-name* | **CURSOR** *cursor-name* }
INTO *sqlda-name*

long-name-spec :
OWNER.TABLE.COLUMN
| **TABLE.COLUMN**
| **COLUMN**

statement-name : *identifier* or *hostvar*

cursor-name : declared cursor

sqlda-name : *identifier*

参数

- **USER TYPES 子句** 含有 USER TYPES 子句的 DESCRIBE 语句可返回列的域信息。通常，在前一个 DESCRIBE 返回 DT_HAS_USERTYPE_INFO 指示符时会执行这样的 DESCRIBE。

  返回的信息与不带 USER TYPES 关键字的 DESCRIBE 返回的信息相同，只不过 sqlname 字段保存的是域名而不是列名。

  如果 DESCRIBE 使用 LONG NAMES 子句，则 sqldata 字段会保存此信息。

- **ALL 子句** DESCRIBE ALL 可用于通过一个向数据库服务器发出的请求来描述 INPUT 和 OUTPUT。这可以提高性能。首先由 OUTPUT 信息填充 SQLDA，然后是 INPUT 信息。sqld 字段包含 INPUT 和 OUTPUT 变量的总数。指示符变量中的 DT_DESCRIBE_INPUT 位是为 INPUT 变量而设，对于 OUTPUT 变量则清零。

- **INPUT 子句** 绑定变量是数据库执行该语句时由应用程序提供的值。绑定变量可以看作是语句的参数。DESCRIBE INPUT 用绑定变量名填充 SQLDA 中的名称字段。DESCRIBE INPUT 还会将绑定变量的数目放在 SQLDA 的 sqlda 字段中。

  DESCRIBE 使用 SQLDA 中的指示符变量提供其它信息。DT_PROCEDURE_IN 和 DT_PROCEDURE_OUT 位是在描述 CALL 语句时在指示符变量中设置的位。DT_PROCEDURE_IN 指示 IN 或 INOUT 参数，而 DT_PROCEDURE_OUT 则指示 INOUT 或 OUT 参数。过程的 RESULT 列将清除这两个位。在说明 OUTPUT 之后，可以使用这些位来区分具有结果集的语句（需要使用 OPEN、FETCH、RESUME 和 CLOSE）和不具有结果集的语句（需要使用 EXECUTE）。DESCRIBE INPUT 仅在绑定变量是 CALL 语句的参数时才相应设置 DT_PROCEDURE_IN 和 DT_PROCEDURE_OUT；作为 CALL 语句参数的表达式中的绑定变量不会设置这些位。

- **OUTPUT 子句** DESCRIBE OUTPUT 语句将填充 SQLDA 中每个选择列表项的数据类型和长度。名称字段也用选择列表项的名称填充。如果为选择列表项指定了别名，则名称就是该别名。否则，名称从选择列表项派生：如果该项是简单列名，则使用该项；否则使用表达式的子字符串。DESCRIBE 还会将选择列表项的数目放在 SQLDA 的 sqld 字段中。

  如果所描述的语句是两个或更多 SELECT 语句的 UNION，则为 DESCRIBE OUTPUT 返回的列名就是为第一个 SELECT 语句返回的列名。

  如果描述 CALL 语句，则 DESCRIBE OUTPUT 语句将为过程中的每个 INOUT 或 OUT 参数填充 SQLDA 中的数据类型、长度和名称。DESCRIBE OUTPUT 还会将 INOUT 或 OUT 参数的数目放在 SQLDA 的 sqld 字段中。

  如果用结果集描述 CALL 语句，则 DESCRIBE OUTPUT 语句将为过程定义中的每个 RESULT 列填充 SQLDA 中的数据类型、长度和名称。DESCRIBE OUTPUT 还会将结果列的数目放在 SQLDA 的 sqld 字段中。

- **LONG NAMES 子句** LONG NAMES 子句用于为语句或游标检索列名。若无此子句，则列名长度限定于 29 个字符；若有此子句，则支持任意长度的名称。

  如果使用了 LONG NAMES，长名称就会放到 SQLDA 的 SQLDATA 字段中，就像从游标中读取一样。将不填充其它任何字段（SQLLEN、SQLTYPE 等）。SQLDA 必须像 FETCH SQLDA 一样设置：它必须为每个列都包含一个条目，且该条目必须是字符串类型。如果存在指示符变量，则会照常指示截断操作。

  长名称的缺省规范为 **TABLE.COLUMN**。

- **WITH VARIABLE RESULT 子句** 此子句用于描述可以有一个以上结果集（具有不同列数或列类型）的过程。

  如果使用了 WITH VARIABLE RESULT，则数据库服务器会在 DESCRIBE 语句后将 SQLCOUNT 值设置为以下值之一：

  - **0** 结果集可以更改。应该在执行每个 OPEN 语句后再次描述过程调用。
  - **1** 结果集是固定的。不需要重新进行描述。

    有关 SQLDA 结构的用法的详细信息，请参见“SQL 描述符区域 (SQLDA)”一节 《SQL Anywhere 服务器 - 编程》。

注释

DESCRIBE 语句设置指定的 SQLDA 以描述指定语句的 OUTPUT（等效于 SELECT LIST）或 INPUT (BIND VARIABLES)。

在使用 INPUT 的情况下，DESCRIBE BIND VARIABLES 不会在 SQLDA 中设置数据类型：这需要由应用程序完成。ALL 关键字用于在同一个 SQLDA 中描述 INPUT 和 OUTPUT。

如果指定了语句名，则先前必须已使用 PREPARE 语句以同一语句名准备了该语句，并且先前必须已分配了 SQLDA（请参见“ALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 343 页）。

如果指定了游标名，则先前必须已声明和打开了该游标。缺省操作是描述 OUTPUT。仅 SELECT 语句和 CALL 语句有 OUTPUT。将 SQLDA 的 sqld 字段设置为零后，对其它任何语句或非动态游标执行 DESCRIBE OUTPUT 时都不会显示输出。

在嵌入式 SQL 中，缺省情况下将 NCHAR、NVARCHAR 和 LONG NVARCHAR 分别描述为 DT_FIXCHAR、DT_VARCHAR 和 DT_LONGVARCHAR。如果已经调用了 db_change_nchar_charset 函数，则将这些数据类型分别描述为 DT_NFIXCHAR、DT_NVARCHAR 和 DT_LONGNVARCHAR。请参见“db_change_nchar_charset 函数”一节 《SQL Anywhere 服务器 - 编程》。

有关如何描述 NCHAR 数据类型的详细信息，请参见数据类型的相关文档：“NCHAR 数据类型”一节第 78 页、“NVARCHAR 数据类型”一节第 80 页和“LONG NVARCHAR 数据类型”一节第 77 页。

权限

无。

副作用

无。

另请参见

- “ALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 343 页
- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “OPEN 语句 [ESQL] [SP]”一节第 647 页
- “PREPARE 语句 [ESQL]”一节第 657 页

标准和兼容性

- **SQL/2003** 核心特性。一些子句是服务商扩充。

示例

下面的示例显示如何使用 DESCRIBE 语句：

```
sqlda = alloc_sqlda( 3 );
EXEC SQL DESCRIBE OUTPUT
  FOR employee_statement
  INTO sqlda;
if( sqlda->sqld  >  sqlda->sqln ) {
  actual_size = sqlda->sqld;
  free_sqlda( sqlda );
  sqlda = alloc_sqlda( actual_size );
  EXEC SQL DESCRIBE OUTPUT
```



```
        FOR employee_statement
        INTO sqlda;
    }
```

DESCRIBE 语句 [Interactive SQL]

DESCRIBE 语句会返回有关给定数据库对象的信息。

语法 1 - 描述数据库对象

DESCRIBE [[**INDEX FOR**] **TABLE** | **PROCEDURE**] [*owner.*]*object-name*

object-name: *table*, *view*, *materialized view*, *procedure*, or *function*

语法 2 - 描述当前连接

DESCRIBE CONNECTION

参数

- **INDEX FOR 子句** 指明想要查看指定 *object-name* 的索引。
- **TABLE 子句** 指明要描述的 *object-name* 是表或视图。
- **PROCEDURE 子句** 指明 *object-name* 是过程或函数。

注释

DESCRIBE TABLE 用于列出指定表或视图中的所有列。DESCRIBE TABLE 语句将对每个表列都返回一个数据行，包括：

- **Column** 列的名称。
- **Type** 列中的数据类型。
- **Nullable** 是否允许空值（1= 是，0= 否）。
- **Primary Key** 列是否在主键中（1= 是，0= 否）。

DESCRIBE INDEX FOR TABLE 用于列出指定表的所有索引。DESCRIBE TABLE 语句将对每个索引都返回一个数据行，包括：

- **Index Name** 索引的名称。
- **Columns** 索引中的列。
- **Unique** 索引是否唯一（1= 是，0= 否）。
- **Type** 索引的类型。可能的值为：Clustered、Statistic、Hashed 和 Other。

DESCRIBE PROCEDURE 用于列出指定过程或函数所使用的所有参数。DESCRIBE PROCEDURE 语句将对每个参数都返回一行，包括：

- **Parameter** 参数的名称。
- **Type** 参数的数据类型。

- **In/Out** 有关传递到参数，或从参数返回的内容的信息。可能的值为：
  - **输入** 将参数传递到过程，但未进行修改。
  - **输出** 过程忽略参数的初始值，并在过程返回时设置参数的值。
  - **输入/输出** 将参数传递到过程，并过程返回时由过程设置参数的值。
  - **结果** 参数返回一个结果集。
  - **返回** 参数返回一个已声明的返回值。

如果未指定 TABLE 或 PROCEDURE（例如，DESCRIBE *object-name*），则 Interactive SQL 会假定对象是一个表。但如果不存在此类表，Interactive SQL 会尝试将对象描述为过程或函数。

使用语法 2 可以列出有关与 Interactive SQL 连接的数据库或数据库服务器的信息。返回以下属性：

- **Database Product** Interactive SQL 连接到的数据库产品的名称和版本号（例如，SQL Anywhere 11.0.0.83）。
- **Host Name** 运行数据库服务器的计算机的网络名称。
- **Host TCP/IP Address** 运行数据库服务器的计算机的 IP 地址。
- **Host Operating System** 运行数据库服务器的计算机所使用的操作系统的名称和版本号。
- **服务器名** 数据库服务器的名称。
- **服务器 TCP/IP 端口** 数据库服务器的当前连接所使用的端口号。
- **Database Name** 与 Interactive SQL 连接的数据库的名称。
- **数据库字符集** 数据库中用于 CHAR 列的字符集。
- **Connection String** 用于连接到数据库或数据库服务器的连接字符串。用三个星号替换口令。

忽略不适用于当前连接的属性。例如，如果连接到使用共享内存的数据库服务器，则忽略 TCP/IP 端口。

权限

无

副作用

无

另请参见

- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

描述 Departments 表中的列：


```
DESCRIBE TABLE Departments;
```

下面是此语句的结果集示例：

Column	Type	Nullable	Primary key
DepartmentID	integer	0	1
DepartmentName	char(40)	0	0
DepartmentHeadID	integer	0	0

列出 Customers 表的索引：

```
DESCRIBE INDEX FOR TABLE Customers;
```

下面是此语句的结果示例：

Index Name	Columns	Unique	Type
IX_customer_name	Surname,GivenName	0	Clustered

DETACH TRACING 语句

使用此语句结束诊断跟踪会话。

语法

DETACH TRACING { WITH | WITHOUT } SAVE

参数

- **WITH SAVE 子句** 指定 WITH SAVE 以在诊断表中保存任何未保存的诊断数据。
- **WITHOUT SAVE 子句** 如果不想保存任何未保存的跟踪数据，则指定 WITHOUT SAVE。

注释

从正在分析的数据库中发出此语句可以停止向诊断表发送诊断信息。如果指定 WITHOUT SAVE 子句，则稍后仍可通过使用 sa_save_trace_data 系统过程保存数据（假定跟踪数据库仍在运行且另一个跟踪会话尚未启动）。请参见“sa_save_trace_data 系统过程”一节第 883 页。

若要确认数据库的当前跟踪级别，请查看 sa_diagnostic_tracing_level 表。请参见“sa_diagnostic_tracing_level 表”一节第 780 页。

注意
跟踪信息*不会*作为数据库卸载或重装操作的一部分被卸载。如果需要从一个数据库向另一个数据库传输跟踪信息，必须通过复制 sa_diagnostic_* 表的内容来手动执行；但是，不建议采用此方法。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “ATTACH TRACING 语句”一节第 388 页
- “REFRESH TRACING LEVEL 语句”一节第 669 页
- “使用诊断跟踪进行高级应用程序分析”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_diagnostic_tracing_level 表”一节第 780 页
- “sa_save_trace_data 系统过程”一节第 883 页

标准和兼容性

- **SQL/2003** 服务商扩充。

DISCONNECT 语句 [ESQL] [Interactive SQL]

此语句用于删除数据库的当前连接。

语法

DISCONNECT [*connection-name* | **CURRENT** | **ALL**]

connection-name : *identifier*, *string*, or *hostvar*

注释

DISCONNECT 语句用于删除与数据库服务器的连接并释放该连接使用的所有资源。如果在 CONNECT 语句中指定了要删除的连接，则可以指定此名称。指定 ALL 会删除应用程序与所有数据库环境的所有连接。CURRENT 是缺省设置，它会删除当前连接。

关闭数据库连接前，如果 commit_on_exit 选项设置为 On，则 Interactive SQL 自动执行 COMMIT 语句。如果此选项设置为 Off，则 Interactive SQL 执行隐式 ROLLBACK。缺省情况下，commit_on_exit 选项设置为 On。

有关删除非当前连接的信息，请参见“DROP CONNECTION 语句”一节第 541 页。

此语句在过程、触发器、事件或批处理中不受支持。

权限

无。

副作用

无。

另请参见

- “CONNECT 语句 [ESQL] [Interactive SQL]”一节第 410 页
- “SET CONNECTION 语句 [Interactive SQL] [ESQL]”一节第 700 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL/基础特性。

示例

下面的语句说明如何在嵌入式 SQL 中使用 DISCONNECT：

```
EXEC SQL DISCONNECT :conn_name
```

下面的语句显示了如何从 Interactive SQL 使用 DISCONNECT 断开所有连接：

```
DISCONNECT ALL;
```

DROP CONNECTION 语句

此语句用于删除用户与数据库的连接。

语法

DROP CONNECTION *connection-id*

注释

DROP CONNECTION 语句通过删除与数据库的连接将用户与数据库断开。

connection-id 参数是一个整数常量。使用 sa_conn_info 系统过程可以获得 *connection-id*。

此语句在过程、触发器、事件或批处理中不受支持。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “CONNECT 语句 [ESQL] [Interactive SQL]”一节第 410 页
- “sa_conn_info 系统过程”一节第 807 页
- “在过程和触发器中使用异常处理程序”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下过程删除由连接号标识的连接。注意，在过程内执行 DROP CONNECTION 语句时，应使用 EXECUTE IMMEDIATE 语句执行此操作，如下面示例所示：

```
CREATE PROCEDURE drop_connection_by_id( IN conn_number INTEGER )
 BEGIN
   EXECUTE IMMEDIATE 'DROP CONNECTION ' || conn_number;
 END;
```

以下语句删除 ID 号为 4 的连接。

```
DROP CONNECTION 4;
```

DROP DATABASE 语句

此语句用于删除与数据库关联的数据库文件。

语法

DROP DATABASE *database-name* [**KEY** *key*]

注释

DROP DATABASE 语句将磁盘中所有关联的数据库文件物理删除。如果数据库文件不存在，或者不满足启动数据库的条件，则会生成错误。

DROP DATABASE 不能用在存储过程、触发器、事件或批处理中。

权限

所需的权限使用数据库服务器 -gu 选项来设置。缺省设置为要求具有 DBA 权限。

数据库必须未被使用才能进行删除。

如果要删除高度加密的数据库，必须指定密钥

Windows Mobile 上不支持。

副作用

除了从磁盘上删除数据库文件外，任何关联的事务日志文件或事务日志镜像文件也被删除。

另请参见

- “CREATE DATABASE 语句”一节第 413 页
- “DatabaseKey 连接参数 [DBKEY]”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

删除 *C:\temp* 目录中的数据库 *temp.db*：

```
DROP DATABASE 'c:\temp\temp.db';
```


DROP DATATYPE 语句

此语句用于从数据库中删除数据类型。

语法

DROP DATATYPE *datatype-name*

注释

如果您不希望在 DROP 语句试图删除不存在的数据库对象时返回错误，请使用 IF EXISTS 子句。

建议使用 DROP DOMAIN 而不是 DROP DATATYPE，因为 DROP DOMAIN 是 ANSI/ISO SQL3 草案中使用的语法。不能从数据库中删除系统定义的数据类型（例如 MONEY 或 UNIQUEIDENTIFIERSTR）。

权限

任何拥有对象或 DBA 权限的用户都可以执行 DROP DATATYPE 语句。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。

另请参见

- “CREATE DOMAIN 语句”一节第 424 页
- “ALTER DOMAIN 语句”一节第 350 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyDatatype。如果该数据类型不存在，则返回一条错误。

```
DROP DATATYPE MyDatatype;
```

DROP DBSPACE 语句

此语句用于从数据库中删除 dbspace。

语法

DROP DBSPACE *dbspace-name*

注释

在删除 dbspace 之前必须先删除 dbspace 中的所有表。您无法使用 DROP DBSPACE 语句删除预定义的 system、temporary、temp、translog 或 translogmirror dbspace。请参见“预定义 dbspace”一节《SQL Anywhere 服务器 - 数据库管理》。

如果您不希望在 DROP DBSPACE 语句试图删除不存在的数据库对象时返回错误，请使用 IF EXISTS 子句。

只要 DROP DBSPACE 语句影响了当前正由其它连接使用的对象，就会禁止该语句。

权限

您必须拥有该对象或具有 DBA 权限，并且必须是唯一的数据库连接。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。

另请参见

- “CREATE DBSPACE 语句”一节第 420 页
- “ALTER DBSPACE 语句”一节第 348 页
- “删除 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyDBSpace。如果该 dbspace 不存在，则返回一条错误。

```
DROP DBSPACE MyDBSpace;
```

DROP DOMAIN 语句

此语句用于从数据库中删除一个域。

语法

DROP DOMAIN *domain-name*

注释

如果您不希望在 DROP DOMAIN 语句试图删除不存在的数据库对象时返回错误，请使用 IF EXISTS 子句。

如果表列、过程或函数参数中使用数据类型，则会禁止 DROP DOMAIN 语句。要删除数据类型，必须更改使用域定义的所有列的数据类型。建议使用 DROP DOMAIN 而不是 DROP DATATYPE，因为 DROP DOMAIN 是 ANSI/ISO SQL3 草案中使用的语法。不能从数据库中删除系统定义的数据类型（例如 MONEY 或 UNIQUEIDENTIFIERSTR）。

权限

任何拥有对象或 DBA 权限的用户都可以执行 DROP DOMAIN 语句。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。

另请参见

- “CREATE DOMAIN 语句”一节第 424 页
- “ALTER DOMAIN 语句”一节第 350 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除域 MyDomain。如果该域不存在，则返回一条错误。

```
DROP DOMAIN MyDomain;
```

DROP EVENT 语句

此语句用于从数据库中删除一个事件。

语法

DROP EVENT [**IF EXISTS**] [*owner.*]*event-name*

注释

如果您不希望在 DROP EVENT 语句试图删除不存在的事件时返回错误，请使用 IF EXISTS 子句。

权限

任何拥有对象或 DBA 权限的用户都可以执行 DROP EVENT 语句。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。

另请参见

- “CREATE EVENT 语句”一节第 429 页
- “ALTER EVENT 语句”一节第 351 页
- “TRIGGER EVENT 语句”一节第 726 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyEvent。如果该事件不存在，则返回一条错误。

```
DROP EVENT MyEvent;
```

DROP EXTERNLOGIN 语句

此语句用于从 SQL Anywhere 目录中删除外部登录。

语法

DROP EXTERNLOGIN *login-name* **TO** *remote-server*

参数

- **DROP 子句** 指定本地用户登录名
- **TO 子句** 指定远程服务器的名称。该服务器的本地用户的备用登录名和口令是已删除的外部登录。

注释

DROP EXTERNLOGIN 从 SQL Anywhere 目录中删除外部登录。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE EXTERNLOGIN 语句”一节第 436 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

```
DROP EXTERNLOGIN DBA TO sybase1;
```

DROP FUNCTION 语句

此语句用于从数据库中删除一个函数。

语法

DROP FUNCTION [**IF EXISTS**] [*owner.*]*function-name*

注释

如果您不希望在 DROP FUNCTION 语句试图删除不存在的函数时返回错误，请使用 IF EXISTS 子句。

只要 DROP FUNCTION 语句影响了当前正由其它连接使用的对象，就会禁止该语句。

权限

任何拥有对象或 DBA 权限的用户都可以执行 DROP FUNCTION 语句。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。

另请参见

- “CREATE FUNCTION 语句（Web 服务）”一节第 445 页
- “ALTER FUNCTION 语句”一节第 354 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyFunction。如果该函数不存在，则返回一条错误。

```
DROP FUNCTION MyFunction;
```

DROP INDEX 语句

此语句用于从数据库中删除一个索引。

语法

DROP INDEX { [[*owner.*]*table-name.*]*index-name* | [[*owner.*]*materialized-view-name.*]*index-name* }

注释

如果您不希望在 DROP INDEX 语句试图删除不存在的数据库对象时返回错误，请使用 IF EXISTS 子句。

只要 DROP INDEX 语句影响了当前正由其它连接使用的对象，就会禁止该语句。

权限

具有表的 REFERENCES 权限的用户可以执行 DROP INDEX。

如果存在使用 WITH HOLD 子句打开的使用语句或事务快照的游标，则无法执行 DROP INDEX 语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。DROP INDEX 语句将关闭用于当前连接的所有游标。

如果使用 DROP INDEX 语句删除局部临时表的索引，将返回 [**未找到索引**] 错误。请使用 DROP TABLE 语句来删除局部临时表。当局部临时表超出范围时，会自动删除局部临时表上的索引。

另请参见

- “CREATE INDEX 语句”一节第 449 页
- “ALTER INDEX 语句”一节第 356 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyIndex。如果该索引不存在，则返回一条错误。

```
DROP INDEX MyIndex;
```

DROP LOGIN POLICY 语句

此语句用于删除登录策略。

语法

DROP LOGIN POLICY *policy-name*

参数

- **policy-name** 登录策略的名称。

注释

如果删除指派给用户的策略，语句会失败。根登录策略无法删除。使用 ALTER USER 语句更改用户的策略分配。请参见“ALTER USER 语句”一节第 385 页。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “ALTER LOGIN POLICY 语句”一节第 357 页
- “ALTER USER 语句”一节第 385 页
- “COMMENT 语句”一节第 406 页
- “CREATE LOGIN POLICY 语句”一节第 453 页
- “CREATE USER 语句”一节第 518 页
- “DROP USER 语句”一节第 562 页
- “管理登录策略概述”一节 《SQL Anywhere 服务器 - 数据库管理》
- “删除登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例创建了一个登录策略 Test11，然后将其删除。

```
CREATE LOGIN POLICY Test11;
DROP LOGIN POLICY Test11;
```


DROP MATERIALIZED VIEW 语句

此语句用于从数据库中删除一个实例化视图。

语法

DROP MATERIALIZED VIEW [**IF EXISTS**] [*owner.*]*materialized-view-name*

注释

表中的所有数据都会在删除过程中被自动删除。同时还会删除实例化视图的所有索引和键。

如果您不希望在 DROP MATERIALIZED VIEW 语句试图删除不存在的实例化视图时返回错误，请使用 IF EXISTS 子句。

您不能对当前正由其它连接使用的对象执行 DROP MATERIALIZED VIEW 语句。

执行 DROP MATERIALIZED VIEW 语句会将所有相关常规视图的状态更改为 INVALID。要在删除实例化视图之前确定视图依赖性，请使用 sa_dependent_views 系统过程。请参见“sa_dependent_views 系统过程”一节第 817 页。

权限

任何拥有该对象或具有 DBA 权限的用户都可以执行 DROP MATERIALIZED VIEW 语句。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。关闭用于当前连接的所有游标。

删除视图时，所有过程和触发器都从内存中卸载，因此引用此视图的任何过程或触发器都会反映此视图不存在的事实。如果按常规删除和创建视图，则卸载和装载过程与触发器会影响性能。

另请参见

- “CREATE MATERIALIZED VIEW 语句”一节第 455 页
- “ALTER MATERIALIZED VIEW 语句”一节第 358 页
- “REFRESH MATERIALIZED VIEW 语句”一节第 665 页
- “实例化视图状态和属性”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

从数据库中删除 MyMaterializedView。如果该实例化视图不存在，则返回一条错误。

```
DROP MATERIALIZED VIEW MyMaterializedView;
```

DROP MESSAGE 语句

此语句用于从数据库中删除一条消息。

语法

DROP MESSAGE *msgnum*

注释

无。

权限

任何拥有对象或 DBA 权限的用户都可以执行 DROP MESSAGE 语句。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。

另请参见

- “MESSAGE 语句”一节第 645 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyMessage。如果该消息不存在，则返回一条错误。

```
DROP MESSAGE MyMessage;
```

DROP PROCEDURE 语句

此语句用于从数据库中删除一个过程。

语法

DROP PROCEDURE [**IF EXISTS**] [*owner.*]*procedure-name*

注释

如果您不希望在 DROP PROCEDURE 语句试图删除不存在的过程时返回错误，请使用 IF EXISTS 子句。

如果 DROP PROCEDURE 语句对当前正由其它连接使用的对象造成影响，则不能执行该语句。

权限

任何拥有对象或 DBA 权限的用户都可以执行 DROP PROCEDURE 语句。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。

另请参见

- “CREATE PROCEDURE 语句（Web 服务）”一节第 471 页
- “ALTER PROCEDURE 语句”一节第 361 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyProcedure。如果该过程不存在，则返回一条错误。

```
DROP PROCEDURE MyProcedure;
```

DROP PUBLICATION 语句 [MobiLink] [SQL Remote]

此语句用于删除发布。在 MobiLink 中，发布可以标识 SQL Anywhere 远程数据库中的同步数据。在 SQL Remote 中，发布可以标识统一数据库和远程数据库中的复制数据。

语法

DROP PUBLICATION [*owner*.]*publication-name*

owner, *publication-name* : *identifier*

注释

此语句仅适用于 MobiLink 和 SQL Remote。

权限

必须具有 DBA 权限。

副作用

自动提交。删除发布的所有预订。

另请参见

- “ALTER PUBLICATION 语句 [MobiLink] [SQL Remote]”一节第 362 页
- “CREATE PUBLICATION 语句 [MobiLink] [SQL Remote]”一节第 476 页
- SQL Anywhere MobiLink 客户端： “发布数据”一节 《MobiLink - 客户端管理》
- UltraLite MobiLink 客户端： “UltraLite DROP PUBLICATION 语句”一节 《UltraLite - 数据库管理和参考》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句删除 pub_contact 发布。

```
DROP PUBLICATION pub_contact;
```

DROP REMOTE MESSAGE TYPE 语句 [SQL Remote]

此语句用于从数据库中删除消息类型定义。

语法

DROP REMOTE MESSAGE TYPE *message-system*

message-system:
FILE
| **FTP**
| **SMTP**

注释

此语句从数据库中删除消息类型。

权限

必须具有 DBA 权限。若要删除类型，必须在没有为任何用户授予此类型的 REMOTE 或 CONSOLIDATE 权限的情况下才能删除。

副作用

自动提交。

另请参见

- “CREATE REMOTE MESSAGE TYPE 语句 [SQL Remote]”一节第 479 页
- “SQL Remote 消息系统”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句从数据库中删除 FILE 消息类型。

```
DROP REMOTE MESSAGE TYPE file;
```

DROP SERVER 语句

此语句用于从 SQL Anywhere 目录中删除远程服务器。

语法

DROP SERVER *server-name*

注释

DROP SERVER 从 SQL Anywhere 目录中删除远程服务器。必须先删除已为远程服务器定义的所有代理表才能成功执行此语句。

权限

只有 DBA 用户可以删除远程服务器。

Windows Mobile 上不支持。

副作用

自动提交。

另请参见

- “CREATE SERVER 语句”一节第 481 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

```
DROP SERVER ase_prod;
```

DROP SERVICE 语句

此语句用于删除 Web 服务。

语法

DROP SERVICE *service-name*

注释

此语句删除系统表 ISYSWEBSERVICE 中列出的 Web 服务。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “ALTER SERVICE 语句”一节第 366 页
- “CREATE SERVICE 语句”一节第 484 页
- “ISYSWEBSERVICE 系统表”一节第 766 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

若要删除名为 tables 的 Web 服务，请执行以下语句：

```
DROP SERVICE tables;
```

DROP STATEMENT 语句 [ESQL]

此语句用于释放语句资源。

语法

DROP STATEMENT [*owner.*]*statement-name*

statement-name :
identifier
| *hostvar*

注释

DROP STATEMENT 语句释放指定的预准备语句所使用的资源。这些资源由成功执行的 PREPARE 语句分配，通常直到释放数据库连接时才释放。

权限

必须已经准备了此语句。

副作用

无。

另请参见

- “PREPARE 语句 [ESQL]”一节第 657 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下是 DROP STATEMENT 的用法示例：

```
EXEC SQL DROP STATEMENT S1;
EXEC SQL DROP STATEMENT :stmt;
```

DROP STATISTICS 语句

此语句用于消除指定列的所有列统计信息。

语法

DROP STATISTICS [**ON**] [*owner.*]*object-name* [**(** *column-list* **)**]

object-name :
table-name
| *materialized-view-name*
| *temp-table-name*

注释

SQL Anywhere 优化程序使用列统计信息来确定执行每个语句的最佳策略。SQL Anywhere 自动收集和更新这些统计信息。列统计信息永久地存储在数据库的系统表 ISYSCOLSTAT 中。处理某条语句时所收集的列统计信息可以在搜索执行后续语句的有效方式时使用。

有时，列统计信息会变得不准确，或者相关统计信息可能不可用。最有可能发生这种情况的时候是：添加、更新或删除了大量数据后，只执行了很少的查询。

DROP STATISTICS 语句从系统表 ISYSCOLSTAT 的指定列中删除所有内部统计数据。这种剧烈的动作导致优化程序无法访问基本的统计信息。若没有这些统计信息，优化程序可能会生成低效数据访问计划，从而导致数据库性能低下。

DROP STATISTICS 语句需要在当前执行它的表上有一个独占锁。这表明在引用该表的所有其它连接提交或回滚了引用事务，或关闭了引用该表的任何打开游标之前，不能继续执行该语句。

仅当在确定问题时或者向数据库中重新装入与原数据有很大差别的数据时，才应该使用此语句。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE STATISTICS 语句”一节第 491 页
- “优化程序估计值和列统计信息”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “ISYSCOLSTAT 系统表”一节第 757 页

标准和兼容性

- **SQL/2003** 服务商扩充。

DROP SUBSCRIPTION 语句 [SQL Remote]

此语句用于从发布中删除用户的预订。

语法

DROP SUBSCRIPTION TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*, ...

subscription-value: *string*

subscriber-id: *string*

参数

- **publication-name** 用户所预订的发布的名称。这可以包括发布的所有者。
- **subscription-value** 与发布的预订表达式进行比较的字符串。此值是必需的，因为一个用户可以多次预订某个发布。

- **subscriber-id** 发布预订者的用户 ID。

注释

删除某个用户 ID 对当前数据库中的发布进行的 SQL Remote 预订。当发布中的数据更改时，此用户 ID 不再接收更新。

在 SQL Remote 中，发布和预订是一种双向关系。如果删除远程用户对统一数据库上的发布的预订，则还应删除远程数据库上统一数据库的预订，以防止远程数据库的更新发送到统一数据库。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE SUBSCRIPTION 语句 [SQL Remote]”一节第 492 页
- “ISYSSUBSCRIPTION 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句删除用户 SamS 对发布 pub_contact 的预订。

```
DROP SUBSCRIPTION TO pub_contact
FOR SamS;
```

DROP SYNCHRONIZATION PROFILE 语句 [MobiLink]

此语句用于删除 SQL Anywhere 同步配置文件。同步配置文件定义 SQL Anywhere 数据库如何与 MobiLink 服务器同步。

语法

DROP SYNCHRONIZATION PROFILE *name*

参数

- **name** 要删除的同步配置文件的名称。

注释

无。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE SYNCHRONIZATION PROFILE 语句 [MobiLink]”一节第 493 页
- “ALTER SYNCHRONIZATION PROFILE 语句 [MobiLink]”一节第 369 页

标准和兼容性

- **SQL/2003** 服务商扩充。

DROP SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]

此语句用于删除 MobiLink 远程数据库中的同步预订。

语法

DROP SYNCHRONIZATION SUBSCRIPTION
TO *publication-name*
[**FOR** *ml_username*, ...]

参数

- **TO 子句** 指定发布名称。
- **FOR 子句** 再指定一个 MobiLink 用户。

  省略此子句将删除发布的缺省设置。

权限

必须具有 DBA 权限。要求对发布中引用的所有表具有独占访问权限。

副作用

自动提交。

另请参见

- “ALTER SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节第 370 页
- “CREATE SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节第 494 页
- “ISYSSYNC 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例删除 MobiLink 用户 ml_user1 对名为 sales_publication 的发布的预订：

```
DROP SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR "ml_user1";
```

以下示例省略了 FOR 子句，因而删除名为 sales_publication 的发布的缺省设置：

```
DROP SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication;
```

DROP SYNCHRONIZATION USER 语句 [MobiLink]

此语句用于从 SQL Anywhere 远程数据库中删除一个或多个同步用户。

语法

DROP SYNCHRONIZATION USER *ml_username*, ...

ml_username: *identifier*

注释

从 MobiLink 远程数据库中删除一个或多个同步用户。

权限

必须具有 DBA 权限。要求对发布中引用的所有表具有独占访问权限。

副作用

所有与用户关联的预订也将删除。

另请参见

- “ALTER SYNCHRONIZATION USER 语句 [MobiLink]”一节第 372 页
- “CREATE SYNCHRONIZATION USER 语句 [MobiLink]”一节第 496 页
- “ISYSSYNC 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

从数据库中删除 MobiLink 用户 ml_user1。

```
DROP SYNCHRONIZATION USER ml_user1;
```

DROP TABLE 语句

此语句用于从数据库中删除一个表。

语法

DROP TABLE [**IF EXISTS**] [*owner*.]*table-name*

注释

删除表时，该表中的所有数据都会在删除过程中被自动删除。同时还会删除表的所有索引和键。

如果您不希望在 DROP TABLE 语句试图删除不存在的表时返回错误，请使用 IF EXISTS 子句。

如果 DROP TABLE 语句对当前正由其它连接使用的表造成影响，则不能执行该语句。如果存在依赖于表的实例化视图，则也会禁止执行 DROP TABLE 语句。

执行 DROP TABLE 语句时，会将所有相关常规视图的状态更改为 INVALID。要在删除表之前确定视图依赖性，请使用 sa_dependent_views 系统过程。请参见“sa_dependent_views 系统过程”一节第 817 页。

权限

任何拥有对象或 DBA 权限的用户都可以执行 DROP TABLE 语句。

除非引用全局临时表的所有用户都已经断开，否则不能删除全局临时表。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。执行 DROP TABLE 语句将关闭用于当前连接的所有游标。

可使用 DROP TABLE 语句来删除局部临时表。

另请参见

- “CREATE TABLE 语句”一节第 497 页
- “ALTER TABLE 语句”一节第 373 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyTable。如果该表不存在，则返回一条错误。

```
DROP TABLE MyTable;
```

从数据库中删除 MyTable（如果存在）。如果该表不存在，则不会返回错误。

```
DROP TABLE IF EXISTS MyTable;
```

DROP TEXT CONFIGURATION 语句

删除文本配置对象。

语法

DROP TEXT CONFIGURATION [*owner.*]*text-config-name*

注释

试图删除具有相关文本索引的文本配置对象将产生错误。必须先删除相关文本索引，再删除文本配置对象。

文本配置对象存储在 ISYSTEXTCONFIG 系统表中。

要确定引用文本配置对象的文本索引，请参见“查看数据库中的文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

必须是文本配置对象的所有者或具有 DBA 权限。

副作用

自动提交

另请参见

- “DROP TEXT INDEX 语句”一节第 560 页
- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本配置对象”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “SYSTEXTCONFIG 系统视图”一节第 979 页
- “CREATE TEXT CONFIGURATION 语句”一节第 508 页
- “ALTER TEXT CONFIGURATION 语句”一节第 381 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句用于创建和删除 mytextconfig 文本配置对象：

```
CREATE TEXT CONFIGURATION mytextconfig FROM default_char;
DROP TEXT CONFIGURATION mytextconfig;
```

DROP TEXT INDEX 语句

从数据库中删除文本索引。

语法

DROP TEXT INDEX *text-index-name*
ON [*owner.*]*table-name*

参数

- **ON 子句** 此子句用于指定在其上构建文本索引的表。

注释

必须先删除相关文本索引，然后才可以删除文本配置对象。

权限

必须是基础表的所有者，或者具有 DBA 权限，或者具有 REFERENCES 权限。

当存在使用 WITH HOLD 子句打开的使用语句快照或事务快照的游标时，不能执行此语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

副作用

自动提交

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “SYSTEXTCONFIG 系统视图”一节第 979 页
- “CREATE TEXT INDEX 语句”一节第 509 页
- “ALTER TEXT INDEX 语句”一节第 383 页
- “DROP TEXT INDEX 语句”一节第 560 页
- “REFRESH TEXT INDEX 语句”一节第 668 页
- “TRUNCATE TEXT INDEX 语句”一节第 728 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句用于创建和删除 TextIdx 文本索引：

```
CREATE TEXT INDEX TextIdx ON MarketingInformation ( Description )
DROP TEXT INDEX TextIdx ON MarketingInformation;
```

DROP TRIGGER 语句

此语句用于从数据库中删除一个触发器。

语法

DROP TRIGGER [**IF EXISTS**] [*owner*.] [*table-name*.]*trigger-name*

注释

如果您不希望在 DROP 语句试图删除不存在的数据库对象时返回错误，请使用 IF EXISTS 子句。

权限

具有表的 ALTER 权限的用户可以执行 DROP TRIGGER。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。

另请参见

- “CREATE TRIGGER 语句”一节第 511 页
- “ALTER TRIGGER 语句”一节第 384 页
- “ROLLBACK TRIGGER 语句”一节第 687 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyTrigger。如果该触发器不存在，则返回一条错误。

```
DROP TRIGGER MyTrigger;
```

DROP USER 语句

此语句用于删除用户。

语法

DROP USER *user-name*

参数

- **user-name** 所删除的用户的名称。

权限

必须具有 DBA 权限。

注释

无。

副作用

无。

另请参见

- “ALTER LOGIN POLICY 语句”一节第 357 页
- “ALTER USER 语句”一节第 385 页
- “COMMENT 语句”一节第 406 页
- “CREATE LOGIN POLICY 语句”一节第 453 页
- “CREATE USER 语句”一节第 518 页
- “DROP LOGIN POLICY 语句”一节第 548 页
- “管理登录策略概述”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例从数据库中删除用户 SQLTester。

```
DROP USER SQLTester;
```


DROP VARIABLE 语句

此语句用于消除 SQL 变量。

语法

DROP VARIABLE [**IF EXISTS**] *identifier*

注释

DROP VARIABLE 语句消除以前用 CREATE VARIABLE 语句创建的 SQL 变量。释放数据库连接时，变量会被自动删除。变量通常用于大对象，因此在使用后消除它们或者将它们设置为 NULL 可以释放大量资源（主要是磁盘空间）。

如果您不希望在 DROP 语句试图删除不存在的数据库对象时返回错误，请使用 IF EXISTS 子句。

权限

无。

副作用

无。

另请参见

- “CREATE VARIABLE 语句”一节第 517 页
- “SET 语句”一节第 696 页

标准和兼容性

- **SQL/2003** 服务商扩充。

DROP VIEW 语句

此语句用于从数据库中删除一个视图。

语法

DROP VIEW [**IF EXISTS**] [*owner.*]*view-name*

注释

如果您不希望在 DROP VIEW 语句试图删除不存在的视图时返回错误，请使用 IF EXISTS 子句。

执行 DROP VIEW 语句时，会将所有相关常规视图的状态更改为 INVALID。要在删除视图之前确定视图依赖性，请使用 sa_dependent_views 系统过程。请参见“sa_dependent_views 系统过程”一节第 817 页。

权限

任何拥有对象或 DBA 权限的用户都可以执行 DROP VIEW 语句。

副作用

自动提交。清除 Interactive SQL 中 [**结果**] 窗格上 [**结果**] 选项卡中的内容。执行 DROP VIEW 语句将关闭用于当前连接的所有游标。

删除视图时，所有过程和触发器都从内存中卸载，因此引用此视图的任何过程或触发器都会反映此视图不存在的事实。如果按常规删除和创建视图，则卸载和装载过程与触发器会影响性能。

另请参见

- “CREATE VIEW 语句”一节第 520 页
- “ALTER VIEW 语句”一节第 387 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

从数据库中删除 MyView。如果该视图不存在，则返回一条错误。

```
DROP VIEW MyView;
```


SQL 语句 (E-O)

以下各节定义了 SQL 语句 E-O 的语法信息。

另请参见

- “SQL 语句 (A-D)”一节第 343 页
- “SQL 语句 (P-Z)”一节第 655 页
- “SQL 语法中的常见元素”一节第 340 页
- “语法约定”一节第 341 页
- “语句适用性指示符”一节第 342 页

EXCEPT 子句

返回位于 EXCEPT 之前的查询块结果集中的行，这些行在 EXCEPT 之后的查询块结果集中不存在。

语法

[**WITH** *temporary-views*] *main-query-block*
EXCEPT [**ALL** | **DISTINCT**] *except-query-block*
[**ORDER BY** [*integer* | *select-list-expression-name*] [**ASC** | **DESC**], ...]
[**FOR XML** *xml-mode*]
[**OPTION(** *query-hint*, ... **)**]

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* = *option-value*

main-query-block：查询块。请参见“SQL 语法中的常见元素”一节第 340 页。

except-query-block：查询块。请参见“SQL 语法中的常见元素”一节第 340 页。

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

参数

- **main-query-block** 要将一个或多个查询块与其进行比较的查询块。
- **except-query-block** 位于 EXCEPT 子句右侧的查询块。*except-query-block* 的结果将与 *main-query-block* 的结果进行比较，以标识仅存在于 *main-query-block* 中的行。
- **EXCEPT 子句** 如果 *main-query-block* 中存在与 *except-query-block* 中的行不匹配的重复行，则在指定了 EXCEPT ALL 的情况下重复行会出现在结果中。要消除结果中的重复行，则指定 EXCEPT 或 EXCEPT DISTINCT。

- **OPTION 子句** 此子句用于指定执行语句时的提示。支持以下提示：
  - MATERIALIZED VIEW OPTIMIZATION *option-value*
  - FORCE OPTIMIZATION
  - *option-name* = *option-value*

  有关这些选项的说明，请参见“SELECT 语句”一节第 689 页的 OPTIONS 子句。

注释

使用 EXCEPT 子句将 main-query-block 的结果与一个或多个 except-query-block 的结果进行比较，然后返回仅存在于 main-query-block 中的行。如果希望在结果中 main-query-block 的重复行不作为重复行出现，请指定 EXCEPT 或 EXCEPT DISTINCT。否则，请指定 EXCEPT ALL。

EXCEPT 与 EXCEPT DISTINCT 相同。

各个 *query-block* 在选择列表中必须具有相同数目的项目。

EXCEPT ALL 的结果集中的行数等于各个查询的结果集行数之间的差。

EXCEPT 的结果和 EXCEPT ALL 相同，只是在使用 EXCEPT 时，重复的行在计算结果集间的差异之前已被排除。

如果两个选择列表中的相应项具有不同的数据类型，则 SQL Anywhere 为结果中的相应列选择数据类型，并自动相应地转换各 *query-block* 中的列。UNION 的第一个查询说明用于确定要与 ORDER BY 子句匹配的名称。

显示的列名与为第一个 *query-block* 显示的列名相同。另一种自定义结果集列名的方法是在 *query-block* 上使用 WITH 子句。

权限

必须具有对各 *query-block* 的 SELECT 权限。

副作用

无

另请参见

- “EXCEPT 子句”一节第 565 页
- “INTERSECT 子句”一节第 623 页
- “UNION 子句”一节第 729 页
- “SELECT 语句”一节第 689 页

标准和兼容性

- **SQL/2003** EXCEPT DISTINCT 是核心特性。EXCEPT ALL 是特性 F304。

示例

有关 EXCEPT 用法的示例，请参见“集合运算符和 NULL”一节 《SQL Anywhere 服务器 - SQL 的用法》。

EXECUTE 语句 [ESQL]

此语句用于执行预准备 SQL 语句。

语法 1

EXECUTE *statement*
[**USING** { *hostvar-list* | **DESCRIPTOR** *sqlda-name* }]
[**INTO** { *into-hostvar-list* | **DESCRIPTOR** *into-sqlda-name* }]
[**ARRAY** *:row-count*]

row-count : *integer* or *hostvar*

statement : *identifier* | *hostvar* | *string*

sqlda-name : *identifier*

into-sqlda-name : *identifier*

语法 2

EXECUTE IMMEDIATE *statement*

statement : *string* | *hostvar*

参数

- **USING 子句** SELECT 语句或 CALL 语句的结果放在变量列表的变量中或放在指定的 SQLDA 所描述的程序数据区中。OUTPUT（选择列表或参数）与主机变量列表或 SQLDA 描述符数组之间是一对一的对应关系。
- **INTO 子句** 如果 EXECUTE INTO 与 INSERT 语句一起使用，则插入的行在第二个描述符中返回。例如，当使用自动增量主键或生成主键值的 BEFORE INSERT 触发器时，EXECUTE 语句提供一种机制，可立即重新读取行并确定指派给该行的主键值。对自动增量键使用 @@identity 也可以获得同样的效果。
- **ARRAY 子句** 可选的 ARRAY 子句可与预准备 INSERT 语句一起使用，从而允许宽插入，即一次插入多行，这样可以提高性能。整数值是插入的行数。SQLDA 中，每个条目必须有一个变量（行数 * 列数）。第一行放在 SQLDA 变量 0 和（每行的列数）-1 之间，依此类推。

注释

EXECUTE 语句可用于任何可以预准备的 SQL 语句。游标用于返回数据库中的许多行的 SELECT 语句或 CALL 语句（请参见“在嵌入式 SQL 中使用游标”一节 《SQL Anywhere 服务器 - 编程》）。

成功执行 INSERT、UPDATE 或 DELETE 语句后，SQLCA (SQLCOUNT) 中的 *sqlerrd*[*2*] 字段中填入受上述操作影响的行数。

语法 1 执行指定的动态语句，该语句是先前已预准备的语句。如果动态语句包含为请求（绑定变量）提供信息的主机变量占位符，则要么 *sqlda-name* 必须指定一个指向 SQLDA（它包含了足够的描述符供语句中出现的所有绑定变量使用）的 C 指针变量，要么必须在 *hostvar -list* 中提供绑定变量。

语法 2 PREPARE 和 EXECUTE 语句的简写形式，不包含绑定变量或输出。字符串或主机变量中包含的 SQL 语句会立即得以执行，并且在完成后被删除。

权限

检查所执行的语句的权限。

副作用

无。

另请参见

- “EXECUTE IMMEDIATE 语句 [SP]”一节第 570 页
- “PREPARE 语句 [ESQL]”一节第 657 页
- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页

标准和兼容性

- **SQL/2003** 核心 SQL 外部的特性。

示例

执行 DELETE。

```
EXEC SQL EXECUTE IMMEDIATE
'DELETE FROM Employees WHERE EmployeeID = 105';
```

执行预准备 DELETE 语句。

```
EXEC SQL PREPARE del_stmt FROM
'DELETE FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE del_stmt USING :employee_number;
```

执行预准备查询。

```
EXEC SQL PREPARE sel1 FROM
'SELECT Surname FROM Employees WHERE EmployeeID = :a';
EXEC SQL EXECUTE sel1 USING :employee_number INTO :surname;
```

EXECUTE 语句 [T-SQL]

可使用语法 1 调用过程，它是 CALL 语句的替代方法，与 Adaptive Server Enterprise 兼容。语法 2 用于在 Transact-SQL 中执行预准备 SQL 语句。

语法 1

EXECUTE [*@return_status* =] [*creator.*]*procedure_name* [*argument*, ...]

argument :
 [*@parameter-name* =] *expression*
| [*@parameter-name* =] *@variable* [*output*]

语法 2

EXECUTE (*string-expression* **)**

注释

语法 1 执行存储过程，选择性地提供过程参数以及检索输出值和返回状态信息。

实现 EXECUTE 语句是为了与 Transact-SQL 兼容，但此语句可以在 Transact-SQL 或 Watcom-SQL 批处理和过程中使用。

若使用语法 2，则可在 Transact-SQL 存储过程和触发器内执行语句。EXECUTE 语句扩展了可以从过程和触发器内执行的语句的范围。它使您可以执行动态预准备语句，比如用传递到过程中的参数构造的语句。此语句中的字符串必须用单引号括起来，并且此语句必须位于一行上。

Transact-SQL EXECUTE 语句无法表明需要结果集。指示 Transact-SQL 过程返回结果集的一种方法是包含类似下面的语句：

```
IF 1 = 0 THEN
      SELECT 1 AS a
```

也可以在 Transact-SQL 存储过程和触发器内执行语句。请参见“EXECUTE IMMEDIATE 语句 [SP]”一节第 570 页。

权限

必须是过程的所有者，具有过程的 EXECUTE 权限或者有 DBA 权限。

副作用

无。

另请参见

- “CALL 语句”一节第 400 页
- “EXECUTE 语句 [ESQL]”一节第 567 页
- “EXECUTE IMMEDIATE 语句 [SP]”一节第 570 页

示例

下面的过程阐释了语法 1。

```
CREATE PROCEDURE p1( @var INTEGER = 54 )
AS
PRINT 'on input @var = %1!', @var
DECLARE @intvar integer
SELECT @intvar=123
SELECT @var=@intvar
PRINT 'on exit @var = %1!', @var;
```

以下语句使用参数输入值 23 执行过程。如果是从 Open Client 或 JDBC 应用程序连接，则 PRINT 消息显示在客户端窗口上。如果是从 ODBC 或嵌入式 SQL 应用程序连接，则消息显示在数据库服务器消息窗口中。

```
EXECUTE p1 23;
```

以下是另一种执行过程的方法，当有多个参数时该方法很有用。

```
EXECUTE p1 @var = 23;
```

以下语句使用参数的缺省值执行过程。

```
EXECUTE p1;
```


以下语句执行过程，并将返回值存储在用于检查状态的变量中。

```
EXECUTE @status = p1 23;
```

EXECUTE IMMEDIATE 语句 [SP]

此语句用于使动态构造的语句能够从过程内执行。

语法 1

EXECUTE IMMEDIATE [*execute-option*] *string-expression*

execute-option:
WITH QUOTES [**ON** | **OFF**]
| **WITH ESCAPES** { **ON** | **OFF** }
| **WITH RESULT SET** { **ON** | **OFF** }

语法 2

EXECUTE (*string-expression* **)**

参数

- **WITH QUOTES 子句** 当指定 WITH QUOTES 或 WITH QUOTES ON 时，字符串表达式中的任何双引号都被假定用于界定标识符。当未指定 WITH QUOTES 或指定了 WITH QUOTES OFF 时，对字符串表达式中的双引号的处理方式取决于 quoted_identifier 选项的当前设置。

  如果使用传递到存储过程中的对象名构造要执行的语句，但该名称可能需要双引号，并且当 quoted_identifier 设置为 Off 时可能调用该过程，这时使用 WITH QUOTES 非常有用。请参见“quoted_identifier 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **WITH ESCAPES 子句** 使用 WITH ESCAPES OFF 可以忽略字符串表达式中的任何转义序列（例如 \n、\x 或 \\）。例如，两个连续的反斜线仍保留为两个反斜线，而不会转换为一个反斜线。缺省设置与 WITH ESCAPES ON 相同。

  WITH ESCAPES OFF 的用途之一是为了更简便地执行引用了包含反斜杠的文件名的动态构造语句。

  在某些环境下，*string-expression* 中的转义序列在执行 EXECUTE IMMEDIATE 语句之前已被转换。例如，复合语句在执行之前进行语分析，转义序列会在分析过程中被转换，而不论 WITH ESCAPES 如何设置。在此类环境下，WITH ESCAPES OFF 可防止进一步的转换。例如：

```
BEGIN
   DECLARE String1 LONG VARCHAR;
   DECLARE String2 LONG VARCHAR;
   EXECUTE IMMEDIATE
        'SET String1 = ''One backslash: \\\\ ''';
     EXECUTE IMMEDIATE WITH ESCAPES OFF
        'SET String2 = ''Two backslashes: \\\\ ''';
   SELECT String1, String2
END
```

- **WITH RESULT SET 子句** 可以通过指定 WITH RESULT SET ON 让 EXECUTE IMMEDIATE 语句返回结果集。使用该子句，包含过程被标记为返回结果集。如果未包括此子句，在该语句生成结果集的情况下，调用该过程时将报告一个错误。

> **注意**
> 缺省选项为 WITH RESULT SET OFF，表示语句执行时不产生结果集。

注释

EXECUTE 语句扩展了可以从过程和触发器内执行的语句的范围。它使您可以执行动态预准备语句，比如用传递到过程中的参数构造的语句。

此语句中的字符串必须用单引号括起来，并且此语句必须位于一行上。

EXECUTE IMMEDIATE 执行的语句只能引用全局变量。

在 Transact-SQL 存储过程和触发器的内部只能使用语法 2。

权限

无。此语句以过程所有者的权限执行，而不是以调用过程的用户的权限执行。

副作用

无。但是，如果此语句是副作用为自动提交的数据定义语句，则会提交。

有关在过程中使用 EXECUTE IMMEDIATE 语句的详细信息，请参见“在过程中使用 EXECUTE IMMEDIATE 语句”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “CREATE PROCEDURE 语句（Web 服务）”一节第 471 页
- “BEGIN 语句”一节第 395 页
- “EXECUTE 语句 [ESQL]”一节第 567 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL/基础特性。

示例

以下过程创建一个表，其中表名作为过程的参数提供。EXECUTE IMMEDIATE 语句必须全部在一行上。

```
CREATE PROCEDURE CreateTableProc(
                  IN tablename char(30)
                  )
BEGIN
   EXECUTE IMMEDIATE
   'CREATE TABLE ' || tablename ||
   ' ( column1 INT PRIMARY KEY)'
END;
```

调用过程并创建名为 mytable 的表：

```
CALL CreateTableProc( 'mytable' );
```

有关对返回结果集的查询使用 EXECUTE IMMEDIATE 的示例，请参见“在过程中使用 EXECUTE IMMEDIATE 语句”一节 《SQL Anywhere 服务器 - SQL 的用法》。

EXIT 语句 [Interactive SQL]

此语句用于退出 Interactive SQL。

语法

{ **EXIT** | **QUIT** | **BYE** } [*return-code*]

return-code: *number* | *connection-variable*

注释

将 Interactive SQL 作为窗口式程序运行时，此语句会关闭 Interactive SQL 窗口，而以命令提示符（批处理）模式运行时，此语句会完全终止 Interactive SQL。在这两种情况下，数据库连接也被关闭。关闭数据库连接前，如果 commit_on_exit 选项设置为 On，则 Interactive SQL 自动执行 COMMIT 语句。如果此选项设置为 Off，则 Interactive SQL 执行隐式 ROLLBACK。缺省情况下，commit_on_exit 选项设置为 On。

在批处理文件中可以使用可选的返回代码来指示 Interactive SQL 命令文件中的命令是成功还是失败。缺省返回代码是 0。

权限

无。

副作用

如果选项 commit_on_exit is 设置为 On（缺省），则此语句自动执行提交；否则执行隐式回退。

在 Windows 操作系统中，提供的可选返回值是 ERRORLEVEL。

另请参见

- “SET OPTION 语句”一节第 702 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

如果表 T 有行，下面的示例将把 Interactive SQL 返回值设置为 1，如果 T 中没有行，则设置为 0。

```
CREATE VARIABLE rowCount INT;
CREATE VARIABLE retcode INT;
SELECT COUNT(*) INTO rowCount FROM T;
IF( rowCount > 0 ) THEN
    SET retcode = 1;
ELSE
    SET retcode = 0;
END IF;
EXIT retcode;
```


注意
不能编写以下语句，因为 EXIT 是 Interactive SQL 语句（而不是 SQL 语句），而且不能在其它 SQL 块语句中包含 Interactive SQL 语句。

```
CREATE VARIABLE rowCount INT;
SELECT COUNT(*) INTO rowCount FROM T;
IF( rowCount > 0 ) THEN
    EXIT 1;     //  <-- not allowed
ELSE
    EXIT 0;     //  <-- not allowed
END IF;
```

EXPLAIN 语句 [ESQL]

此语句用于检索用于特定游标的优化策略的文本说明。

语法

EXPLAIN PLAN FOR CURSOR *cursor-name*
{ **INTO** *hostvar* | **USING DESCRIPTOR** *sqlda-name* }

cursor-name : *identifier* or *hostvar*

sqlda-name : *identifier*

注释

EXPLAIN 语句检索指定游标的文本形式优化策略。游标必须已事先声明并处于打开状态。

hostvar 或 *sqlda-name* 变量必须为字符串类型。优化字符串指定搜索表的顺序，还指定搜索所使用的索引（如果有）。

根据查询的不同，该字符串可能很长，并且具有以下格式：

```
table (index), table (index), ...
```

如果为表指定了相关名，将出现相关名而不是表名。表名在列表中出现的顺序就是数据库服务器访问它们的顺序。每个表名后是一个括在括号内的索引名。此索引即是用于访问该表的索引。如果不使用任何索引（按顺序扫描表），将显示字母 "seq" 作为索引名。如果特定的 SQL SELECT 语句涉及子查询，将用冒号 (:) 分隔每个子查询的优化字符串。这些子查询部分将按照数据库服务器执行查询的顺序出现。

成功执行 EXPLAIN 语句之后，SQLCA (SQLIOESTIMATE) 的 sqlerrd 字段中将填入读取查询的所有行所需的输入/输出操作次数的一个估计值。

在“提高数据库性能”《SQL Anywhere 服务器 - SQL 的用法》中有很多优化字符串示例的讨论。

权限

必须已经打开指定的游标。

副作用

无。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “PREPARE 语句 [ESQL]”一节第 657 页
- “FETCH 语句 [ESQL] [SP]”一节第 574 页
- “CLOSE 语句 [ESQL] [SP]”一节第 405 页
- “OPEN 语句 [ESQL] [SP]”一节第 647 页
- “在嵌入式 SQL 中使用游标”一节 《SQL Anywhere 服务器 - 编程》
- “SQL 通信区域 (SQLCA)”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例阐释了 EXPLAIN 的用法：

```
EXEC SQL BEGIN DECLARE SECTION;
char plan[300];
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE employee_cursor CURSOR FOR
   SELECT EmployeeID, Surname
   FROM Employees
   WHERE Surname like :pattern;
EXEC SQL OPEN employee_cursor;
EXEC SQL EXPLAIN PLAN FOR CURSOR employee_cursor INTO :plan;
printf( "Optimization Strategy: '%s'.n", plan );
```

plan 变量包含以下字符串：

```
'Employees <seq>'
```

FETCH 语句 [ESQL] [SP]

此语句用于重新定位游标，然后从游标中获取数据。

语法

FETCH *cursor-position cursor-name*
[**INTO** { *hostvar-list* | *variable-list* } | **USING DESCRIPTOR** *sqlda-name*]
[**PURGE**]
[**BLOCK** *n*]
[**FOR UPDATE**]
[**ARRAY** *fetch-count*]
INTO *variable-list* [**FOR UPDATE**]

cursor-position :
  NEXT | **PRIOR** | **FIRST** | **LAST**
| { **ABSOLUTE** | **RELATIVE** } *row-count*

row-count : *number* or *hostvar*

cursor-name : *identifier* or *hostvar*

hostvar-list : may contain indicator variables

variable-list : stored procedure variables

sqlda-name : *identifier*

fetch-count : *integer* or *hostvar*

参数

- **INTO 子句** INTO 子句是可选的。如果未指定该子句，则 FETCH 语句仅定位游标。*hostvar-list* 仅供嵌入式 SQL 使用。
- **cursor position** 一个可选定位参数，它允许在读取行之前移动游标。如果读取操作中包括定位参数，且位置在允许的游标位置外，则会发出 SQLE_NOTFOUND 警告，并且由 SQLCOUNT 字段指示与有效位置之间的偏移量。

  OPEN 语句最初将游标定位在第一行的前面。
- **NEXT 子句** NEXT 是缺省定位参数，它使游标在读取行之前向前移动一行。
- **PRIOR 子句** 使游标在读取前向后移动一行。
- **RELATIVE 子句** RELATIVE 定位用于在读取前将游标沿任一方向移动指定的行数。正数表示向前移动，负数表示向后移动。因此，NEXT 与 RELATIVE 1 等效，PRIOR 与 RELATIVE -1 等效。RELATIVE 0 检索的行与此游标的上一读取语句检索的行相同。
- **ABSOLUTE 子句** ABSOLUTE 定位参数用于转到具体行。0 表示第一行前面的位置（请参见“在过程和触发器中使用游标”一节 《SQL Anywhere 服务器 - SQL 的用法》）。

  1 表示第一行，依此类推。负数用于指定游标末尾的绝对位置。-1 表示游标的最后一行。
- **FIRST 子句** ABSOLUTE 1 的缩写形式。
- **LAST 子句** ABSOLUTE -1 的缩写形式。

> **游标定位问题**
> 插入 DYNAMIC SCROLL 游标并对其进行某些更新会导致与游标定位有关的问题。除非 SELECT 语句中有 ORDER BY 子句，否则数据库服务器不将插入的行放在游标内可预知的位置。在有些情况下，插入的行要等到关闭并再次打开游标后才会出现。
>
> 如果必须创建临时表才能打开游标，就会出现这种情况。有关说明，请参见“在查询处理中使用工作表（使用 All-rows 优化目标）”一节 《SQL Anywhere 服务器 - SQL 的用法》。
>
> UPDATE 语句可能导致行在游标中移动。如果游标中具有使用现有索引的 ORDER BY 子句（未创建临时表），则会发生这种情况。

- **BLOCK 子句** 客户端应用程序一次可读取多行。这类读取称为块读取、预读或多行读取。第一个读取会使数据库服务器发回若干行。客户端将这些行放入缓冲区，后续读取从这些缓冲区中检索，而不对数据库服务器发出新的请求。

  BLOCK 子句仅在嵌入式 SQL 中使用。它提示客户端和服务器：应用程序可读取多少行。特殊值 0 表示将请求发送到数据库服务器，并且只返回一行（没有行块）。BLOCK 子句会减少对 BLOCK 值的下一次预读中包含的行数。要增加预读的行数，可使用 PrefetchRows 连接参数。

如果没有指定 BLOCK 子句，则使用执行 OPEN 时指定的值。请参见“OPEN 语句 [ESQL] [SP]”一节第 647 页。

FETCH RELATIVE 0 总是重新读取行。

如果禁用游标的预读，则会忽略 BLOCK 子句，且每次读取一行。如果还指定了 ARRAY，则读取 ARRAY 指定的行数。

- **PURGE 子句** PURGE 子句仅供嵌入式 SQL 使用。它使客户端刷新其所有行的缓冲区，然后向数据库服务器发送读取请求。请注意，此读取请求可能返回行块。
- **FOR UPDATE 子句** FOR UPDATE 子句指示读取的行随后将由 UPDATE WHERE CURRENT OF CURSOR 语句更新。使用此子句会使数据库服务器在行上加一个意图锁。该锁会一直保持到当前事务结束时为止。请参见“锁定的工作方式”一节 《SQL Anywhere 服务器 - SQL 的用法》和“SELECT 语句”一节第 689 页的 FOR UPDATE 子句。
- **ARRAY 子句** ARRAY 子句仅供嵌入式 SQL 使用。它允许所谓的宽读取（即同时检索多行）并会提高性能。

  要在嵌入式 SQL 中使用宽读取，请将 fetch 语句包括在代码中，如下所示：

  ```
  EXEC SQL FETCH ... ARRAY nnn
  ```

  其中 ARRAY *nnn* 是 FETCH 语句的最后一项。读取计数 *nnn* 可以是一个主机变量。SQLDA 必须包含 nnn *（每行的列数）个变量。第一行放在 SQLDA 变量 0 和（每行的列数）-1 之间，依此类推。

  有关使用宽读取的详细示例，请参见“一次读取多个行”一节 《SQL Anywhere 服务器 - 编程》。

注释

FETCH 语句从指定的游标中检索一行。游标在此前必须已打开。

嵌入式 SQL 使用 在 C 源代码中，DECLARE CURSOR 语句必须出现在 FETCH 语句的前面，而 OPEN 语句必须在 FETCH 语句之前执行。如果主机变量用于游标名，则实际上是 DECLARE 语句生成代码，而且 DECLARE 语句必须在 FETCH 语句之前执行。

服务器在 SQLCOUNT 中返回读取的记录数，并且在没有错误或警告的情况下，返回的 SQLCOUNT 总是大于零。

如果在读取时返回 SQLSTATE_NOTFOUND 警告，则 SQLCA (SQLCOUNT) 的 *sqlerrd*[*2*] 字段中包含的值为尝试读取超出允许的游标位置的行数。如果未找到行但位置有效，则值为 0；例如，当定位到游标的最后一行上时执行 FETCH RELATIVE 1。如果所尝试的读取超出了游标的末尾，则为正值；如果所尝试的读取位于游标开头的前面，则为负值。如果所尝试的读取超出了游标的末尾，则游标定位到最后一行，如果所尝试读取位于游标开头的前面，则游标定位到第一行。

成功执行了读取语句后，SQLCA (SQLIOCOUNT) 的 *sqlerrd*[*1*] 字段递增，增量为执行读取所需的输入/输出操作数。此字段实际上在每个数据库语句执行时都递增。

单行读取 SELECT 语句结果中的一行被放入变量表的变量中。选择列表与主机变量列表之间是一对一的对应关系。

多行读取 SELECT 语句结果中的一行或多行放在 *variable-list* 的变量中或 *sqlda-name* 描述的程序数据区中。在上述任一种情况下，*select-list* 与 *hostvar-list* 或 *sqlda-name* 描述符数组之间都是一对一的对应关系。

权限

游标必须是打开的，并且用户必须对游标声明所引用的表具有 SELECT 权限。

副作用

无。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “PREPARE 语句 [ESQL]”一节第 657 页
- “OPEN 语句 [ESQL] [SP]”一节第 647 页
- “在嵌入式 SQL 中使用游标”一节 《SQL Anywhere 服务器 - 编程》
- “在过程和触发器中使用游标”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “FOR 语句”一节第 577 页
- “RESUME 语句”一节第 677 页

标准和兼容性

- **SQL/2003** 核心特性。在过程中使用的是持久存储模块特性。

示例

以下是嵌入式 SQL 的示例：

```
EXEC SQL DECLARE cur_employee CURSOR FOR
SELECT EmployeeID, Surname FROM Employees;
EXEC SQL OPEN cur_employee;
EXEC SQL FETCH cur_employee
INTO :emp_number, :emp_name:indicator;
```

以下是过程示例：

```
BEGIN
   DECLARE cur_employee CURSOR FOR
      SELECT Surname
      FROM Employees;
   DECLARE name CHAR(40);
   OPEN cur_employee;
   lp: LOOP
      FETCH NEXT cur_employee into name;
      IF SQLCODE <> 0 THEN LEAVE lp END IF;
       ...
   END LOOP;
   CLOSE cur_employee;
END
```

FOR 语句

此语句用于对游标中的每一行重复执行一次语句列表。

语法

[*statement-label* :]
FOR *for-loop-name* **AS** *cursor-name* [*cursor-type*] **CURSOR**
{ **FOR** *statement* [**FOR** { **UPDATE** *cursor-concurrency* | **FOR READ ONLY** }]
| **USING** *variable-name* }
DO *statement-list*
END FOR [*statement-label*]

cursor-type :
NO SCROLL
| **DYNAMIC SCROLL**
| **SCROLL**
| **INSENSITIVE**
| **SENSITIVE**

cursor-concurrency : **BY** { **VALUES** | **TIMESTAMP** | **LOCK** }

variable-name : *identifier*

参数

- **NO SCROLL 子句** 声明为 NO SCROLL 的游标仅限于使用 FETCH NEXT 和 FETCH RELATIVE 0 查找操作在结果集中前进。

  由于一旦游标离开行，行就不能返回，所以对游标没有敏感性限制。当请求 NO SCROLL 游标时，SQL Anywhere 提供效率最高的游标，即敏感性未定型游标。请参见“敏感性未定型游标”一节 《SQL Anywhere 服务器 - 编程》。

- **DYNAMIC SCROLL 子句** DYNAMIC SCROLL 是缺省游标类型。DYNAMIC SCROLL 游标可以使用 FETCH 语句的所有形式。

  当请求 DYNAMIC SCROLL 游标时，SQL Anywhere 提供敏感性未定型游标。当使用游标时，总是存在效率与一致性的平衡问题。敏感性未定型游标以一致性为代价提供高效性能。请参见“敏感性未定型游标”一节 《SQL Anywhere 服务器 - 编程》。

- **SCROLL 子句** 声明了 SCROLL 的游标可以使用 FETCH 语句的所有形式。当请求 SCROLL 游标时，SQL Anywhere 提供对值敏感的游标。请参见“对值敏感的游标”一节 《SQL Anywhere 服务器 - 编程》。

  SQL Anywhere 必须以保证结果集成员资格的方式执行对值敏感的游标。DYNAMIC SCROLL 游标更高效，应加以使用，除非要求 SCROLL 游标的一致行为。

- **INSENSITIVE 子句** 声明为 INSENSITIVE 的游标在打开时修复其成员资格；系统会使用所有原始行的副本创建一个临时表。INSENSITIVE 游标中的 FETCHING 看不到其它任何 INSERT、UPDATE 或 DELETE 语句的效果，也看不到另一个游标上的其它任何 PUT、UPDATE WHERE CURRENT、DELETE WHERE CURRENT 操作。它看不到对同一游标执行 PUT、UPDATE WHERE CURRENT、DELETE WHERE CURRENT 操作的效果。请参见“不敏感游标”一节 《SQL Anywhere 服务器 - 编程》。

- **SENSITIVE 子句** 声明为 SENSITIVE 的游标对结果集的成员资格或值的更改敏感。请参见“敏感游标”一节 《SQL Anywhere 服务器 - 编程》。

- **FOR UPDATE 子句** FOR UPDATE 是缺省设置。对于不带 ORDER BY 子句的单表查询，或者在 ansi_update_constraints 选项设置为 Off 的情况下，游标的缺省设置为 FOR UPDATE。如果 ansi_update_constraints 选项设置为 Cursors 或 Strict，则停留在包含 ORDER BY 子句的查询上的游标的缺省设置为 READ ONLY。但可使用 FOR UPDATE 子句将游标显式标记为可更新。
- **READ ONLY 子句** 声明为 FOR READ ONLY 的游标不能在 UPDATE（定位）或 DELETE（定位）操作中使用。由于允许通过 ORDER BY 子句或连接来更新游标的开销大，因此，停留在包含两个或更多表连接的查询上的游标为 READ ONLY，并且无法设置为可更新。为响应指定 FOR UPDATE 的游标的任何请求，SQL Anywhere 提供了对值敏感的游标或敏感性未定型的游标。不敏感的游标或者敏感性未定型游标不能更新。

注释

FOR 语句是一个控制语句，它允许对游标中的每一行执行一次 SQL 语句列表。FOR 语句等效于这样一个复合语句：游标有一个 DECLARE，游标结果集中的每一列有一个变量 DECLARE，后面跟一个循环，该循环将游标中的一行读入局部变量，并对游标中的每一行执行一次 *statement-list*。

有效的游标类型包括动态滚动（缺省）、滚动、非滚动、区分大小写以及不区分大小写。

每个局部变量的名称和数据类型都是从游标中使用的 *statement* 派生的。在 SELECT 语句中，数据类型为选择列表中表达式的数据类型。名称是选择列表项的别名（如果有），否则名称是列的名称。任何不是简单列引用的选择列表项都必须有别名。在 CALL 语句中，名称和数据类型取自过程定义中的 RESULT 子句。

LEAVE 语句可用于指示继续执行 END FOR 之后第一条语句。如果指定结尾 *statement-label*，则它必须与起始 *statement-label* 匹配。

权限

无。

副作用

无。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “FETCH 语句 [ESQL] [SP]”一节第 574 页
- “CONTINUE 语句 [T-SQL]”一节第 413 页
- “LOOP 语句”一节第 637 页

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

下面的代码段阐释了 FOR 循环的用法。

```
FOR names AS curs INSENSITIVE CURSOR FOR
SELECT Surname
FROM Employees
DO
```

```
    CALL search_for_name( Surname );
END FOR;
```

此代码段还阐释了 FOR 循环的用法。

```
BEGIN
    FOR names AS curs SCROLL CURSOR FOR
    SELECT EmployeeID, GivenName FROM Employees where EmployeeID < 130
    FOR UPDATE BY VALUES
    DO
        MESSAGE 'emp: ' || GivenName;
    END FOR;
END
```

以下示例展示了在 myproc 过程内部使用的 FOR 循环，该循环会根据调用过程时指定的排序顺序（asc 表示升序，desc 表示降序）返回 Employees 表中的前 10 名雇员。

```
CALL sa_make_object( 'procedure', 'myproc' ) ;
ALTER PROCEDURE myproc (
    IN @order_by VARCHAR(20) DEFAULT NULL
)
RESULT ( Surname person_name_t )
BEGIN
    DECLARE @sql LONG VARCHAR;
    DECLARE @msg LONG VARCHAR;
    DECLARE LOCAL TEMPORARY TABLE temp_names( surnames person_name_t );
    SET @sql = 'SELECT TOP(10) * FROM Employees AS t ' ;

    CASE @order_by
    WHEN 'asc' THEN
        SET @sql = @sql || 'ORDER BY t.Surname ASC';
        SET @msg = 'Sorted ascending by last name: ';
    WHEN 'desc' THEN
        SET @sql = @sql || 'ORDER BY t.Surname DESC';
        SET @msg = 'Sorted ascending by last name: ';
    END CASE;

    FOR loop_name AS SCROLL CURSOR USING @sql
    DO
        INSERT INTO temp_names( surnames ) VALUES( Surname );
        MESSAGE( @msg || Surname ) ;
    END FOR;
    SELECT * FROM temp_names;
END ;
```

如果调用 myproc 过程时指定了 asc（例如 `CALL myproc( 'asc' );`），则返回以下结果：

Surname
Ahmed
Barker
Barletta
Bertrand
Bigelow

Surname
Blaikie
Braun
Breault
Bucceri
Butterfield

FORWARD TO 语句

此语句用于将本地语法 SQL 语句发送到远程服务器。

语法 1

FORWARD TO *server-name sql-statement*

语法 2

FORWARD TO [*server-name*]

注释

FORWARD TO 语句使用户能够指定需要直通连接的服务器。此语句有两种用法：

- **语法 1** 将单条语句发送到远程服务器。
- **语法 2** 将 SQL Anywhere 设为直通模式，以将一系列语句发送到远程服务器。所有后续语句都直接传递到远程服务器。若要关闭直通模式，在发出 FORWARD TO 时不要指定 *server-name*。

  如果在直通模式下遇到来自远程服务器的错误，仍需发出 FORWARD TO 语句关闭直通模式。

当以用户的名义与 server-name 建立连接时，数据库服务器使用以下方式之一：

- 使用 CREATE EXTERNLOGIN 设置远程登录别名
- 如果未设置远程登录别名，则使用与 SQL Anywhere 通信时使用的名称和口令

如果无法与指定的服务器建立连接，在返回给用户的消息中会指出原因。

在将语句传递给请求的服务器后，所有结果都转换成客户端程序可以识别的形式。

server-name 远程服务器的名称。

SQL-statement 远程服务器的本地 SQL 语法中的命令。该命令或命令组用大括号 ({}) 或单引号括起来。

> **注意**
> FORWARD TO 语句是服务器指令，不能用在存储过程、触发器、事件或批处理中。

权限

无

副作用

在 FORWARD TO 会话的持续时间内，远程连接设置为 AUTOCOMMIT（非链接）模式。在 FORWARD TO 语句之前未完成的任何工作都自动提交。

示例

以下示例向远程服务器 RemoteASE 发送 SQL 语句：

```
FORWARD TO RemoteASE { SELECT * FROM titles };
```

下面示例演示与远程服务器 aseprod 的直通会话：

```
FORWARD TO aseprod
  SELECT * FROM titles
      SELECT * FROM authors
FORWARD TO;
```

标准和兼容性

- **SQL/2003** 服务商扩充。

FROM 子句

此子句用于指定 DELETE、SELECT 或 UPDATE 语句中涉及的数据库表或视图。当在 SELECT 语句中使用时，FROM 子句也可以用在 MERGE 或 INSERT 语句中。

语法

FROM *table-expression*, ...

table-expression :
  table-name
| *view-name*
| *procedure-name*
| *derived-table*
| *lateral-derived-table*
| *join-expression*
| **(** *table-expression*, ... **)**
| *openstring-expression*
| *apply-expression*
| *contains-expression*

table-name :
[*userid*.]*table-name*
[[**AS**] *correlation-name*]
[**WITH (** *hint* [...] **)**]

view-name :
[*userid*.]*view-name* [[**AS**] *correlation-name*]
[**WITH (** *table-hint* **)**]

procedure-name :
[*owner*.]*procedure-name* **(** [*parameter*, ...] **)**
[**WITH (** *column-name data-type*, ... **)**]
[[**AS**] *correlation-name*]

derived-table :
(*select-statement* **)**
[**AS**] *correlation-name* [**(** *column-name*, ... **)**]

lateral-derived-table :
LATERAL (*select-statement* | *table-expression* **)**
[**AS**] *correlation-name* [**(** *column-name*, ... **)**]

join-expression :
table-expression join-operator table-expression
[**ON** *join-condition*]

join-operator :
[**KEY** | **NATURAL**] [*join-type*] **JOIN**
| **CROSS JOIN**

join-type :
INNER
| **LEFT** [**OUTER**]
| **RIGHT** [**OUTER**]
| **FULL** [**OUTER**]

hint :
table-hint | *index-hint*

table-hint :
READPAST
| **UPDLOCK**
| **XLOCK**
| **FASTFIRSTROW**
| **HOLDLOCK**
| **NOLOCK**
| **READCOMMITTED**
| **READUNCOMMITTED**
| **REPEATABLEREAD**
| **SERIALIZABLE**

index-hint :
NO INDEX
| **INDEX (** *index-name* [, ...] **)** [**INDEX ONLY** { **ON** | **OFF** }]
| **FORCE INDEX (** *index-name* **)**

openstring-expression :
OPENSTRING ({ **FILE** | **VALUE** } *string-expression* **)**
WITH (*rowset-schema* **)**
[**OPTION (** *scan-option ...* **)**]
[**AS**] *correlation-name*

apply-expression :
table-expression { **CROSS** | **OUTER** } **APPLY** *table-expression*

contains-expression :
{*table-name* | *view-name* } **CONTAINS (** *column-name* [,...], *contains-query* **)** [[**AS**] *score-correlation-name*]

rowset-schema :
column-schema-list
| **TABLE** [*owner*.]*table-name* [**(** *column-list* **)**]

column-schema-list :
{ *column-name user-or-base-type* | **filler( )** } [, ...]

column-list :
{ *column-name* | **filler( )** } [, ...]

scan-option :
BYTE ORDER MARK { **ON** | **OFF** }
| **COMMENTS INTRODUCED BY** *comment-prefix*
| **DELIMITED BY** *string*
| **ENCODING** *encoding*
| **ESCAPE CHARACTER** *character*
| **ESCAPES** { **ON** | **OFF** }
| **FORMAT** { **TEXT** | **BCP** }
| **HEXADECIMAL** { **ON** | **OFF** }
| **QUOTE** *string*
| **QUOTES** { **ON** | **OFF** }
| **ROW DELIMITED BY** *string*
| **SKIP** *integer*
| **STRIP** { **ON** | **OFF** | **LTRIM** | **RTRIM** | **BOTH** }

contains-query : string

参数

- **table-name** 基表或临时表。其他用户拥有的表可以通过指定用户 ID 来限定。缺省情况下，如果未指定用户 ID，找到的是由当前用户所属的组拥有的表（请参见“引用组拥有的表”一节 《SQL Anywhere 服务器 - 数据库管理》）。
- **view-name** 指定要在查询中包含的视图。同表一样，其他用户拥有的视图可以通过指定用户 ID 来限定。缺省情况下，如果未指定用户 ID，找到的是由当前用户所属的组拥有的视图。

  尽管语法上允许在视图上使用表提示，但这类提示不起作用。
- **procedure-name** 返回结果集的存储过程。此子句仅适用于 SELECT 语句的 FROM 子句。过程名后需要加括号，即使该过程没有参数。如果存储过程返回多个结果集，则仅使用第一个结果集。

  WITH 子句提供了一种为过程结果集指定列名别名的方法。如果指定了 WITH 子句，列数必须与过程结果集的列数一致，而且数据类型也必须与过程结果集中的数据类型兼容。如果未指定 WITH 子句，则列名和类型为由过程定义中确定的列名和类型。以下查询说明了 WITH 子句的用法：

```
SELECT sp.ident, sp.quantity, Products.name
FROM ShowCustomerProducts( 149 ) WITH ( ident INT, description CHAR(20),
```



```
quantity INT ) sp
   JOIN Products
ON sp.ident = Products.ID;
```

- **derived-table** 可以在 FROM 子句中提供 SELECT 语句来代替的表名或视图名。以这种方式使用 SELECT 语句被称为派生表，并且必须要向其提供一个别名。例如，以下语句包含一个派生表 MyDerivedTable，它按照 UnitPrice 对 Products 表中的产品进行排序。

```
SELECT TOP 3 *
       FROM ( SELECT Description,
                     Quantity,
                     UnitPrice,
                     RANK() OVER ( ORDER BY UnitPrice ASC )
                  AS Rank
                  FROM Products ) AS MyDerivedTable
ORDER BY Rank;
```

  有关派生表的详细信息，请参见“查询派生表”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **lateral-derived-table** 可能包含对父语句中对象的引用（外部引用）的派生表、存储过程或连接表。如果要在 FROM 子句中使用外部引用，则必须使用横向派生表。

  只能将外部引用用于 FROM 子句中位于横向派生表之前的表。例如，不能将外部引用用于 *select-list* 中的项。

  表和外部引用必须用逗号分开。例如，以下查询全部有效：

```
SELECT *
 FROM A, LATERAL( B LEFT OUTER JOIN C ON ( A.x = B.x ) ) LDT;

SELECT *
 FROM A, LATERAL( SELECT * FROM B WHERE A.x = B.x ) LDT;

SELECT *
 FROM A, LATERAL( procedure-name( A.x ) ) LDT;
```

  指定 LATERAL (*table-expression*) 等同于指定 LATERAL (SELECT * FROM *table-expression*)。

- **openstring-expression** 指定 OPENSTRING 子句以便在文件或 BLOB 中进行查询，此时将这些源的内容视作行的集合。这样做时，由于不是对已定义的结构（如表或视图）进行查询，因此还需为将要生成的结果集指定有关文件或 BLOB 的模式的信息。此子句适用于 SELECT 语句的 FROM 子句。UPDATE 或 DELETE 语句不支持此子句。

  ROWID 函数可用于对 OPENSTRING 表达式生成的表的结果集进行操作。

  以下所示为 OPENSTRING 子句的次级子句和参数，以及如何使用它们在文件和 BLOB 中定义及查询数据：

  - **FILE 和 VALUE 子句** 使用 FILE 子句可指定要查询的文件。使用 VALUE 子句可指定要查询的 BLOB 表达式。假定 BLOB 表达式的数据类型为 LONG BINARY。可以将 READ_CLIENT_FILE 函数指定为 VALUE 子句的值。

    如果 FILE 或 VALUE 关键字都未指定，则假定关键字为 VALUE。

  - **WITH 子句** 此子句用于指定要查询的数据的行集模式（列名和数据类型）。可以直接指定列（例如，`WITH ( Surname CHAR(30), GivenName CHAR(30) )`）。也可以使用 TABLE 次级子句引用要使用的表，以便从中获取模式信息（例如，`WITH TABLE dba.Employees ( Surname, GivenName )`）。您必须拥有指定的表，或者具有对该表的 SELECT 权限。

指定列时，可以在输入数据中对要跳过的列指定 filler( )（例如，`WITH ( filler( ), Surname CHAR(30), GivenName CHAR(30) )`）。有关 filler( ) 用法的详细信息，请参见“LOAD TABLE 语句”一节第 626 页。

- **OPTION 子句** 使用 OPTION 子句可指定要对输入文件使用的分析选项，如转义字符、分隔符、编码等。支持的选项包括 LOAD TABLE 语句中用于控制输入文件分析的那些选项。请参见“LOAD TABLE 语句”一节第 626 页。

- **scan-option** 有关各个扫描选项的说明，请参见“LOAD TABLE 语句”一节第 626 页中介绍的装载选项。

- **apply-expression** 此子句用于指定一个连接条件，即针对左侧 *table-expression* 的每一行计算右侧的 *table-expression*。例如，可以使用 apply 表达式为表表达式中的每一行计算函数、过程或派生表。请参见“从 apply 表达式生成的连接”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **contains-expression** 在表名后使用 CONTAINS 子句可对表进行过滤，从而仅返回与 *contains-query* 所指定的全文查询相匹配的行。表的每个匹配行与分数列一起返回，可以使用 *score-correlation-name*（如果已指定）引用此分数列。如果未指定 *score-correlation-name*，则使用缺省相关名 contains 来引用分数列。

  如果可选的相关名参数出现异常，则 CONTAINS 子句将采用与 CONTAINS 搜索条件相同的参数。请参见“CONTAINS 搜索条件”一节第 46 页。

  在 CONTAINS 子句中列出的列必须具有文本索引。请参见“文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **correlation-name** *correlation-name* 用于为 FROM 子句中的表或视图指定替换名。然后即可在语句的任何其它位置引用该替换名。例如，emp 和 dep 分别是 Employees 表和 Departments 表的相关名。

```
SELECT Surname, GivenName, DepartmentName
   FROM Employees emp, Departments dep,
   WHERE emp.DepartmentID=dep.DepartmentID;
```

- **WITH table-hint 子句** WITH *table-hint* 子句允许指定仅供此表和仅供此语句使用的行为。使用此子句可以在不更改隔离级别或者设置数据库选项或连接选项的情况下更改行为。表提示可用于基表、临时表和实例化视图。

> **小心**
> WITH *table-hint* 子句是一项高级功能，只应在需要时使用，且只能由经验丰富的数据库管理员使用。此外，并非在所有情况下都应用此设置。

- **隔离级别相关的表提示** 隔离级别表提示用于指定查询表时的隔离级别行为。它们指定仅供指定表使用的锁定方法，且仅用于当前查询。不能指定快照隔离级别作为表提示。

  以下是所支持的隔离级别相关表提示的列表：

表提示	说明
HOLDLOCK	将行为设置为与隔离级别 3 相当。此表提示与 SERIALIZABLE 同义。

表提示	说明
NOLOCK	将行为设置为与隔离级别 0 相当。此表提示与 READUNCOMMITTED 同义。
READCOMMITTED	将行为设置为与隔离级别 1 相当。
READPAST	指示数据库服务器忽略而不是阻塞写锁定的行。此表提示只能与隔离级别 1 一起使用。只有当 FROM 子句中的相关名引用基表或全局共享临时表时才会考虑 READPAST 提示。在其它情况下（视图、代理表和表函数）则忽略 READPAST 提示。只要为基表的相关名指定了提示，在视图中的查询就可能使用 READPAST 提示。由于服务器中锁定和谓语计算的相互作用，使用 READPAST 表提示可能会导致异常。另外，您不能对 DELETE、INSERT、或 UPDATE 语句的目标表使用 READPAST 提示。
READUNCOMMITTED	将行为设置为与隔离级别 0 相当。此表提示与 NOLOCK 同义。
REPEATABLEREAD	将行为设置为与隔离级别 2 相当。
SERIALIZABLE	将行为设置为与隔离级别 3 相当。此表提示与 HOLDLOCK 同义。
UPDLOCK	指示将使用意图锁锁定由提示表中的语句所处理的行。在事务结束前，受影响的行将保持锁定状态。UPDLOCK 可以工作在所有隔离级别且使用意图锁。请参见“意图锁”一节 《SQL Anywhere 服务器 - SQL 的用法》。
XLOCK	指示将独占锁定由提示表中的语句所处理的行。在事务结束前，受影响的行将保持锁定状态。XLOCK 可以工作在所有隔离级别且使用写锁。请参见“写锁定”一节 《SQL Anywhere 服务器 - SQL 的用法》。

有关隔离级别的信息，请参见“隔离级别和一致性”一节 《SQL Anywhere 服务器 - SQL 的用法》。

对 MobiLink 同步使用 READPAST

如果正在为参与 MobiLink 同步的数据库编写查询，建议不要在同步脚本中使用 READPAST 表提示。

有关详细信息，请参见：

- “download_cursor 表事件”一节 《MobiLink - 服务器管理》
- “download_delete_cursor 表事件”一节 《MobiLink - 服务器管理》
- “upload_fetch 表事件”一节 《MobiLink - 服务器管理》

如果由于应用程序执行了许多影响下载性能的更新而考虑使用 READPAST，则可以将快照隔离作为一种替代解决方案。请参见“MobiLink 隔离级别”一节 《MobiLink - 服务器管理》。

- **优化表提示 (FASTFIRSTROW)** FASTFIRSTROW 表提示允许您在不将 optimization_goal 选项设置为 First-row 的情况下为查询设置优化目标。使用 FASTFIRSTROW 时，SQL Anywhere 会选择一种能够缩短读取查询结果第一行所需时间的访问计划。请参见“optimization_goal 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **WITH (index-hint) 子句** 使用 WITH (*index-hint*) 子句可替换查询优化程序的计划选择算法，并告知优化程序使用索引来访问表的准确方式。索引提示可用于基表、临时表和实例化视图。
- **NO INDEX** 使用此子句强制对表执行顺序扫描（不使用索引）。请注意，顺序扫描的开销可能非常大。
- **INDEX (index-name [,...])** 使用此子句可指定索引（最多四个），优化程序必须使用这些索引来满足查询。如果有任何一个指定的索引不能使用，则会返回错误。
- **INDEX ONLY { ON | OFF }** 此子句用于控制是否对数据执行仅索引检索。如果使用 INDEX ONLY ON 指定了 INDEX (*index-name...*) 子句，则数据库服务器会使用指定的索引尝试仅索引检索。如果任一指定的索引无法满足仅索引检索（例如，没有索引，或现有的索引不能满足查询），则返回错误。

  指定 INDEX ONLY OFF 可阻止仅索引检索。
- **FORCE INDEX (index-name)** 此子句用于指定索引，优化程序必须使用该索引在满足查询的表中查找行。出于兼容性考虑，语法为 FORCE INDEX (*index-name*)，它不支持指定一个以上的索引。

> **小心**
> 索引提示会替换查询优化程序的决策逻辑，因此只应由经验丰富的用户使用。使用索引提示可能会导致访问计划达不到最优，并会导致性能较差。

注释

SELECT、UPDATE 和 DELETE 语句需要用一个表列表来指定此语句要使用的表。

> **视图和派生表**
> 虽然 FROM 子句描述针对的是表，但它同样适用于视图和派生表，除非另外说明。

FROM 子句创建由所有指定表中的所有列组成的结果集。组件表中行的所有组合最初都在结果集中，但 JOIN 条件和/或 WHERE 条件通常会减少组合数。

您不能将 ON 短语与 CROSS JOIN 一起使用。

对于语法 2，如果 FILE 子句或 VALUE 子句都未指定，则假定指定了 VALUE 子句。也就是说假定 *string-expression* 为要查询的值。

权限

openstring-expression 的 FILE 子句需要 DBA 权限或 READFILE 权限。

openstring-expression 的 TABLE 子句要求用户拥有指定的表，或是具有对指定表的 SELECT 权限。

副作用

无。

另请参见

- “DELETE 语句”一节第 530 页
- “SELECT 语句”一节第 689 页
- “UPDATE 语句”一节第 735 页
- “INSERT 语句”一节第 616 页
- “MERGE 语句”一节第 638 页
- “连接：从多个表检索数据” 《SQL Anywhere 服务器 - SQL 的用法》
- “MultipleIndexScan 方法 (MultIdx)”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心特性，但以下列出的内容除外。FROM 子句的复杂性意味着应根据标准检查各个子句。
  - KEY JOIN，服务商扩充
  - FULL OUTER JOIN 和 NATURAL JOIN，核心 SQL 外部的 SQL/基础特性
  - READPAST 表提示，服务商扩充
  - LATERAL (*table-expression*)，服务商扩充。注意，LATERAL (*select-statement*) 是 ANSI SQL 标准中的特性 T491。
  - 派生表是特性 F591
  - FROM 子句（表函数）中的过程是特性 T326
  - 公用表表达式是特性 T121
  - 递归表表达式是特性 T131

示例

以下是有效的 FROM 子句：

```
...
FROM Employees
...

...
FROM Employees NATURAL JOIN Departments
...

...
FROM Customers
KEY JOIN SalesOrders
KEY JOIN SalesOrderItems
KEY JOIN Products
...

...
FROM Employees CONTAINS ( Street, ' Way ' )
...
```

以下查询说明如何在查询中使用派生表：

```
SELECT Surname, GivenName, number_of_orders
FROM Customers JOIN
     ( SELECT CustomerID, COUNT(*)
```

```
        FROM SalesOrders
         GROUP BY CustomerID )
      AS sales_order_counts( CustomerID,
                             number_of_orders )
  ON ( Customers.ID = sales_order_counts.CustomerID )
  WHERE number_of_orders > 3;
```

以下查询说明如何从存储过程结果集中选择行：

```
SELECT t.ID, t.QuantityOrdered AS q, p.name
FROM ShowCustomerProducts( 149 ) t JOIN Products p
ON t.ID = p.ID;
```

以下示例说明如何使用 OPENSTRING 子句执行对文件的查询。CREATE TABLE 语句创建名为 testtable 的表，该表含有两列（column1 和 columns2）。UNLOAD 语句通过卸载 RowGenerator 表中的行创建一个名为 *testfile.dat* 的文件。SELECT 语句在 FROM 子句中使用了 OPENSTRING 子句，并同时使用 testtable 和 RowGenerator 这两个表中的模式信息来查询 *testfile.dat* 文件。此查询将返回具有值 49 的行。

```
CREATE TABLE testtable( column1 CHAR(10), column2 INT );
UNLOAD SELECT * FROM RowGenerator TO 'testfile.dat';
SELECT A.column2
  FROM OPENSTRING( FILE 'testfile.dat' )
  WITH ( TABLE testtable( column2 ) ) A, RowGenerator B
  WHERE A.column2 = B.row_num
  AND A.column2 < 50
  AND B.row_num > 48;
```

以下示例说明如何使用 OPENSTRING 子句执行对字符串值的查询。SELECT 语句在 FROM 子句中使用了 OPENSTRING 子句，并同时使用 WITH 子句中提供的模式信息来查询字符串值。该查询将返回两列三行。

```
SELECT *
  FROM OPENSTRING( VALUE '1,"First"$2,"Second"$3,"Third"')
  WITH (c1 INT, c2 VARCHAR(30))
  OPTION ( DELIMITED BY ',' ROW DELIMITED BY '$')
  AS VALS
```

GET DATA 语句 [ESQL]

此语句用于获取游标当前行一个列的字符串或二进制数据。GET DATA 通常用于读取 LONG BINARY 或 LONG VARCHAR 字段。请参见“SET 语句”一节第 696 页。

语法

GET DATA *cursor-name*
COLUMN *column-num*
OFFSET *start-offset*
[**WITH TEXTPTR**]
USING DESCRIPTOR *sqlda-name* | **INTO** *hostvar*, ...

cursor-name : *identifier*, or *hostvar*

column-num : *integer* or *hostvar*

start-offset : *integer* or *hostvar*

sqlda-name : *identifier*

参数

- **COLUMN 子句** *column-num* 的值从 1 开始，标识要从中读取数据的列。该列必须是字符串或二进制类型。
- **OFFSET 子句** *start-offset* 指示在字段值中跳过的字节数。通常情况下，该数字是先前已读取的字节数。执行此 GET DATA 语句时读取的字节数由目标主机变量的长度决定。

  目标主机变量的指示符值属短整型，因此不能总是包含截断的字节数。然而，如果字段包含 NULL 值，则该指示符值包含负值；如果值被截断，则包含正值（不一定是截断的字节数）；如果非 NULL 值没有截断，则包含零。

  同样，如果 LONG VARCHAR 或 LONG VARCHAR 主机变量的偏移量大于零，则 untrunc_len 字段不会准确地指示截断前的大小。
- **WITH TEXTPTR 子句** 如果给出 WITH TEXTPTR 子句，则文本指针检索到第二个主机变量中或 SQLDA 的第二个字段中。该文本指针可用于 Transact-SQL READ TEXT 和 WRITE TEXT 语句。该文本指针是一个 16 位的二进制值，可按如下方式声明：

  ```
  DECL_BINARY( 16 ) textptr_var;
  ```

  WITH TEXTPTR 子句只能用于长数据类型（LONG BINARY、LONG VARCHAR、TEXT 和 IMAGE）。如果尝试将其用于其它数据类型，则会返回错误 INVALID_TEXTPTR_VALUE。

  数据的总长度在 SQLCA 结构的 SQLCOUNT 字段中返回。

注释

从当前游标位置处的行中获取一个列值的片段。游标必须是打开的，并且必须用 FETCH 定位在某行上。

权限

无。

副作用

无。

另请参见

- “FETCH 语句 [ESQL] [SP]”一节第 574 页
- “READTEXT 语句 [T-SQL]”一节第 664 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例使用 GET DATA 读取二进制大对象（也称为 BLOB）。

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY(1000) piece;
short ind;
```

```
EXEC SQL END DECLARE SECTION;
int size;
/* Open a cursor on a long varchar field */
EXEC SQL DECLARE big_cursor CURSOR FOR
SELECT long_data FROM some_table
WHERE key_id = 2;
EXEC SQL OPEN big_cursor;
EXEC SQL FETCH big_cursor INTO :piece;
for( offset = 0; ; offset += piece.len ) {
   EXEC SQL GET DATA big_cursor COLUMN 1
   OFFSET :offset INTO :piece:ind;
   /* Done if the NULL value */
   if( ind < 0 ) break;
   write_out_piece( piece );
   /* Done when the piece was not truncated */
   if( ind == 0 ) break;
}
EXEC SQL CLOSE big_cursor;
```

GET DESCRIPTOR 语句 [ESQL]

此语句用于检索描述符区内某变量的信息或检索它的值。

语法

GET DESCRIPTOR *descriptor-name*
{ *hostvar* = **COUNT** | **VALUE** { *integer* | *hostvar* } *assignment*, ... }

assignment :
hostvar =
TYPE
| **LENGTH**
| **PRECISION**
| **SCALE**
| **DATA**
| **INDICATOR**
| **NAME**
| **NULLABLE**
| **RETURNED_LENGTH**

descriptor-name : identifier

注释

GET DESCRIPTOR 语句用于检索描述符区域内某变量的信息或检索该变量的值。

值 { *integer* | *hostvar* } 指定描述符区域内要检索其信息的变量。执行 GET ...DATA 时会进行类型检查，以确保主机变量和描述符变量的数据类型相同。LONG VARCHAR 和 LONG BINARY 不受 GET DESCRIPTOR ...DATA 支持。

如果出现错误，则在 SQLCA 中返回该错误。

权限

无。

副作用

无。

另请参见

- “ALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 343 页
- “DEALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 522 页
- “SET DESCRIPTOR 语句 [ESQL]”一节第 701 页
- “SQL 描述符区域 (SQLDA)”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 核心特性。

示例

下面的示例返回在 sqlda 中的位置为 col_num 的列的类型。

```
int get_type( SQLDA *sqlda, int col_num )
{
    EXEC SQL BEGIN DECLARE SECTION;
    int ret_type;
    int col = col_num;
    EXEC SQL END DECLARE SECTION;
EXEC SQL GET DESCRIPTOR sqlda VALUE :col :ret_type = TYPE;
    return( ret_type );
}
```

有关详细示例，请参见“ALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 343 页。

GET OPTION 语句 [ESQL]

此语句用于获取选项的当前设置。建议改用 CONNECTION_PROPERTY 函数。

语法

GET OPTION [*userid*.]*option-name*
{ **INTO** *hostvar* | **USING DESCRIPTOR** *sqlda-name* }

userid : *identifier*, *string*, or *hostvar*

option-name : *identifier*, *string*, or *hostvar*

hostvar : indicator variable allowed

sqlda-name : *identifier*

注释

提供 GET OPTION 语句是为了与软件的早期版本兼容。建议通过使用 CONNECTION_PROPERTY 系统函数获取选项的值。

GET OPTION 语句为用户 *userid* 或连接的用户（如果未指定 *userid*）获取选项 *option-name* 的选项设置。这是用户的个人设置，如果连接的用户没有设置，则是 PUBLIC 设置。如果指定的选项是数据库选项，并且用户具有该选项的临时设置，则会检索临时设置。

如果 *option-name* 不存在，则 GET OPTION 返回警告 SQLE_NOTFOUND。

权限

必须是 [无]。

副作用

无。

另请参见

- “SET OPTION 语句”一节第 702 页
- “按字母顺排序的系统过程列表”一节第 792 页
- “CONNECTION_PROPERTY 函数 [System]”一节第 150 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句阐释了 GET OPTION 的用法。

```
EXEC SQL GET OPTION 'date_format' INTO :datefmt;
```

GOTO 语句 [T-SQL]

此语句用于分支到带标签的语句。

语法

label : **GOTO** *label*

注释

Transact-SQL 过程、触发器或批处理中的任何语句都可以带标签。标签名是一个有效标识符后跟一个冒号。在 GOTO 语句中不使用冒号。

权限

无。

副作用

无。

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

以下 Transact-SQL 批处理在数据库服务器消息窗口上输出四次消息 "yes"：

```
DECLARE @count SMALLINT
SELECT @count = 1
restart:
   PRINT 'yes'
   SELECT @count = @count + 1
   WHILE @count <=4
    GOTO restart
```

GRANT 语句

此语句用于创建新用户 ID、为指定用户授予或更改权限以及创建或更改口令。

可以对禁用的对象授予权限。对禁用对象的权限存储在数据库中，在对象被启用时生效。

语法 1 - 授予权限

GRANT *authority*, ...
TO *userid*, ...

authority :
BACKUP
| **DBA**
| **PROFILE**
| **READCLIENTFILE**
| **READFILE**
| [**RESOURCE** | **ALL**]
| **VALIDATE**
| **WRITECLIENTFILE**

语法 2 - 在组中授予组状态或成员资格

GRANT { **GROUP** | **MEMBERSHIP IN GROUP** *userid*, ... }
TO *userid*, ...

语法 3 - 授予数据库对象权限

GRANT *permission*, ...
ON [*owner.*]*object-name*
TO *userid*, ...
[**WITH GRANT OPTION**]
[**FROM** *userid*]

permission :
ALL [**PRIVILEGES**]
| **ALTER**
| **DELETE**
| **INSERT**
| **REFERENCES** [**(** *column-name*, ... **)**]
| **SELECT** [**(** *column-name*, ... **)**]
| **UPDATE** [**(** *column-name*, ... **)**]

语法 4 - 授予执行权限

GRANT EXECUTE ON [*owner*.]{ *procedure-name* | *user-defined-function* }
TO *userid*, ...

语法 5 - 授予集成登录权限

GRANT INTEGRATED LOGIN TO *user-profile-name*, ...
AS USER *userid*

语法 6 - 授予 Kerberos 登录权限

GRANT KERBEROS LOGIN TO *client-Kerberos-principal*, ...
AS USER *userid*

语法 7 - 授予连接权限

GRANT CONNECT TO *userid*, ...
[**AT** *starting-id*]
[**IDENTIFIED BY** *password*, ...]

Syntax 8 - 授予在 dbspace 中执行创建操作的权限

GRANT CREATE ON *dbspace-name*
TO *userid*, ...

参数

- **AT *starting-id* 子句** 此子句不用于通用用途。此子句指定用于列表中第一个用户 ID 的内部数字值。

  AT starting-id 子句由卸载使用程序使用。

- **GRANT authority 子句** 此子句用于授予以下列出的权限之一：
  - **BACKUP 特权** 此权限允许用户备份数据库。请参见“BACKUP 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
  - **DBA 特权** 此权限允许用户执行所有任务。此权限通常是为组织中负责管理数据库的人员保留的。请参见“DBA 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
  - **PROFILE 特权** 此权限允许用户执行分析和诊断操作。请参见“PROFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
  - **READCLIENTFILE 权限** 此权限允许用户从客户端计算机上的文件中进行读取（例如在装载数据时）。请参见“READCLIENTFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
  - **READFILE 特权** 此权限允许用户使用 OPENSTRING 子句对文件执行 SELECT 语句。请参见“READFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
  - **RESOURCE 或 ALL 权限** 此权限允许用户创建表和视图。ALL 是与 Sybase Adaptive Server Enterprise 兼容的 RESOURCE 的同义词。请参见“RESOURCE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **VALIDATE 特权** 此权限允许用户执行由不同 VALIDATE 语句支持的校验操作，例如校验数据库、校验表和索引，以及校验校验和。它还允许用户使用 Sybase Central 中的校验实用程序 (dbvalid) 和 [校验数据库向导]。请参见“VALIDATE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **WRITECLIENTFILE 权限** 此权限允许用户对客户端计算机上的文件进行写入（例如在卸载数据时）。请参见“WRITECLIENTFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **GROUP 子句** 此权限允许用户拥有成员。请参见“管理组”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **MEMBERSHIP IN GROUP 子句** 此权限授予用户某一个组的成员资格。用户将继承在组级别设置的可继承权限和特权。请参见“管理组”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **GRANT permission 子句** GRANT *permission* 子句允许授予对单个表或视图的权限。表权限可分别指定，也可以使用 ALL 同时授予所有权限。以下是可以授予的权限列表：
  - **ALL 权限** 此权限可授予 ALTER、DELETE、INSERT、REFERENCES、SELECT 和 UPDATE 权限。ALL 是 RESOURCE 的同义词。
  - **ALTER 权限** 此权限允许用户使用 ALTER TABLE 语句更改指定的表。对视图不允许此权限。
  - **DELETE 权限** 此权限允许用户从指定的表或视图中删除行。
  - **INSERT 权限** 此权限允许用户向指定的表或视图插入行。
  - **REFERENCES 权限** 此权限允许用户为指定的表创建索引以及引用该表的外键。如果指定了列名，则用户只能引用那些列。列的 REFERENCES 权限不能授予视图，只能授予表。INDEX 是 REFERENCES 的同义词。
  - **SELECT 权限** 此权限允许用户查看视图或表中的信息。如果指定了列名，则用户只能查看那些列。列的 SELECT 权限不能授予视图，只能授予表。
  - **UPDATE 权限** 此权限允许用户更新视图或表中的行。如果指定了列名，则用户只能更新那些列。
- **FROM 子句** 如果指定了 FROM *userid*，则该 *userid* 将作为一个授权者用户 ID 被记录到系统表中。此子句是供卸载实用程序 (dbunload) 使用的。不要直接使用或修改此选项。
- **CONNECT TO 子句**

> **注意**
> 建议使用 CREATE USER 语句创建用户。请参见“CREATE USER 语句”一节第 518 页。

创建新用户。GRANT CONNECT 也可由任何用户用来更改他们自己的口令。若要创建使用空字符串作为口令的用户，请使用：

```
GRANT CONNECT TO userid IDENTIFIED BY "";
```

若要创建没有口令的用户，请使用：

```
GRANT CONNECT TO userid;
```

没有口令的用户不能连接到数据库。如果要创建组，但不希望任何人使用组用户 ID 连接到数据库，则这很有用。用户 ID 必须是有效的标识符。

用户 ID 和口令不能：

- 以空格、单引号或双引号开头
- 以空格结尾
- 含有分号

口令必须是有效的标识符，或者是用单引号引上的字符串（最大 255 字节）。有关指定有效口令的信息，请参见“设置口令”一节 《SQL Anywhere 服务器 - 数据库管理》。

verify_password_function 选项可用于指定函数，以实现口令规则（例如，口令至少必须包含一位）。如果使用口令验证函数，则不能在 GRANT CONNECT 语句中指定多个用户 ID 和口令。请参见“verify_password_function 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **CREATE ON 子句** 允许用户在指定的 dbspace 中创建数据库对象。CREATE 权限可以通过组成员资格继承。在用户可以创建对象前，他们必须也有 RESOURCE 权限。请参见“RESOURCE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。

注释

GRANT 语句用于为个别用户 ID 和组授予数据库权限和特权。它也可用于创建用户和组。

如果指定了 WITH GRANT OPTION，则指定的用户 ID 也被授权向其它用户 ID GRANT（授予）同样的权限。如果将 WITH GRANT OPTION 授予组，则组的成员不会继承 WITH GRANT OPTION。

GRANT 语句的语法 4 用于授予执行过程的权限。

GRANT 语句的语法 5 在一个或多个 Windows 用户或组配置文件与现有数据库用户 ID 之间创建显式集成登录映射，使成功登录到其本地计算机的用户不必提供用户 ID 或口令就可以连接到数据库。*user-profile-name* 可以采用 *domain\user-name* 的形式。集成登录映射到的数据库用户 ID 必须具有口令。请参见“使用集成登录”一节 《SQL Anywhere 服务器 - 数据库管理》。

GRANT 语句的语法 6 创建从一个或多个 Kerberos 主体到现有数据库用户 ID 的 Kerberos 验证登录映射。这使成功登录到 Kerberos 的用户（具有一个有效 Kerberos 票据授予票据的用户）不必提供用户 ID 或口令，就可以连接到数据库。Kerberos 登录映射到的数据库用户 ID 必须具有口令。*client-Kerberos-principal* 必须采用 *user/instance@REALM* 的格式，其中 */instance* 是可选的。必须指定包括域在内的完整主体，只在实例或域内有差异的主体将被视为不同主体。

主体区分大小写，因此必须指定正确的大小写。不支持映射只有大小写存在差异的多个主体（例如，无法同时映射 jjordan@MYREALM.COM 和 JJordan@MYREALM.COM）。

如果没有为 Kerberos 主体创建显式映射，但存在 Guest 数据库用户 ID 且具有口令，则 Kerberos 主体使用 Guest 数据库用户 ID （与集成登录的 Guest 数据库用户 ID 相同）进行连接。

有关 Kerberos 验证的详细信息，请参见“Kerberos 验证”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

语法 3 如果指定了 FROM 子句，则必须具有 DBA 权限。否则，必须拥有表，或必须已经用 WITH GRANT OPTION 授予了对表的权限。

语法 4 必须拥有过程，或必须具有 DBA 权限。

语法 5 和 6 必须具有 DBA 权限。

语法 7 必须使用 GRANT CONNECT 更改自己的口令，或必须具有 DBA 权限。如果正在更改其他用户的口令（利用 DBA 权限），则其他用户肯定不能连接到数据库。

副作用

自动提交。

另请参见

- “REVOKE 语句”一节第 679 页
- “数据库权限和特权概述”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CREATE USER 语句”一节第 518 页

标准和兼容性

- **SQL/2003** 语法 3 是核心特性。语法 4 是持久存储模块特性。其它语法是服务商扩充。

示例

创建新的数据库用户。

```
GRANT CONNECT TO SQLTester
IDENTIFIED BY welcome
```

向用户 Laurel 授予对 Employees 表的权限。

```
GRANT
SELECT, UPDATE ( Street )
ON Employees
TO Laurel;
```

可以在一个语句中授予多个权限。各权限用逗号分隔。

允许用户 Hardy 执行 Calculate_Report 过程。

```
GRANT EXECUTE ON Calculate_Report
TO Hardy;
```

GRANT CONSOLIDATE 语句 [SQL Remote]

此语句用于标识 SQL Remote 层次中位于当前数据库上一层的数据库，该数据库将从当前数据库接收消息。

语法

GRANT CONSOLIDATE
TO *userid*
TYPE *message-system*, ...

ADDRESS *address-string*, ...
[**SEND** { **EVERY** | **AT** } *hh*:*mm*:*ss*]

message-system:
FILE | **FTP** | **SMTP**

address: *string*

参数

- **userid** 被授予权限的用户的用户 ID。
- **message-system** SQL Remote 支持的消息系统之一。
- **address** 指定的消息系统的地址。

注释

在 SQL Remote 安装中，必须为 SQL Remote 层次中位于当前数据库正上方的数据库授予 CONSOLIDATE 权限。远程数据库中发出 GRANT CONSOLIDATE 以标识它的统一数据库。每个数据库只能有一个用户 ID 具有 CONSOLIDATE 权限：一个远程数据库不能有多个统一数据库。

统一用户由消息系统标识，以标识出统一用户收发消息的方法。地址名必须是消息系统的有效地址，用单引号括起来。每个远程数据库只能有一个统一用户。

对于 FILE 消息类型，地址是 SQLREMOTE 环境变量所指向目录的子目录。

统一数据库需要使用 GRANT CONSOLIDATE 语句来接收消息，但此语句本身并不为统一数据库预订任何数据。若要预订数据，必须为统一用户 ID 创建对当前数据库中的某个发布的预订。在统一数据库中运行数据库抽取实用程序可创建一个已发出正确 GRANT CONSOLIDATE 语句的远程数据库。

可选的 SEND EVERY 和 SEND AT 子句指定发送消息的频率。该字符串包含各消息之间的时间长度（对于 SEND EVERY）或一天中发送消息的具体时间（对于 SEND AT）。使用 SEND AT 时，每天发送一次消息。

如果授予用户远程权限时未指定 SEND EVERY 或 SEND AT 子句，则消息代理处理消息，然后停止。要继续运行消息代理，必须确保为每个具有 REMOTE 权限的用户都指定了频率 SEND AT 或 SEND EVERY。

预计在许多远程数据库中，消息代理都定期运行，并且统一数据库将不指定 SEND 子句。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CONSOLIDATE 权限”一节 《SQL Remote》
- “GRANT PUBLISH 语句 [SQL Remote]”一节第 601 页
- “GRANT REMOTE 语句 [SQL Remote]”一节第 602 页
- “REVOKE CONSOLIDATE 语句 [SQL Remote]”一节第 681 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

```
GRANT CONSOLIDATE TO con_db
TYPE SMTP
ADDRESS 'Singer, Samuel';
```

GRANT PUBLISH 语句 [SQL Remote]

此语句用于标识当前数据库的发布者。

语法

GRANT PUBLISH TO *userid*

注释

在外发消息中，SQL Remote 系统中的每个数据库都以用户 ID 加以标识，这些用户 ID 称为发布者。GRANT PUBLISH 语句标识与这些外发消息关联的发布者用户 ID。

只能有一个用户 ID 具有 PUBLISH 权限。有 PUBLISH 权限的用户 ID 由特殊常量 CURRENT PUBLISHER 标识。以下的查询标识当前发布者：

```
SELECT CURRENT PUBLISHER;
```

如果没有发布者，则特殊常量为 NULL。

当前发布者特殊常量可用作列的缺省设置。在复制表时将 CURRENT PUBLISHER 列用作主键的一部分通常很有用，因为这有助于防止因多个站点的更新而导致主键冲突。

要更改发布者，必须先用 REVOKE PUBLISH 语句删除当前发布者，然后用 GRANT PUBLISH 语句创建新的发布者。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “PUBLISH 权限”一节 《SQL Remote》
- “GRANT PUBLISH 语句 [SQL Remote]”一节第 601 页
- “GRANT CONSOLIDATE 语句 [SQL Remote]”一节第 599 页
- “REVOKE PUBLISH 语句 [SQL Remote]”一节第 682 页
- “CREATE SUBSCRIPTION 语句 [SQL Remote]”一节第 492 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

```
GRANT PUBLISH TO publisher_ID;
```

GRANT REMOTE 语句 [SQL Remote]

此语句用于标识 SQL Remote 层次中位于当前数据库下一层的数据库，该数据库将从当前数据库接收消息。这些数据库的用户称为远程用户。

语法

GRANT REMOTE TO *userid*, ...
TYPE *message-system*, ...
ADDRESS *address-string*, ...
[**SEND** { **EVERY** | **AT** } *send-time*]

参数

- **userid** 被授予权限的用户的用户 ID。
- **message-system** SQL Remote 支持的消息系统之一。它必须是以下值之一：
  - FILE
  - FTP
  - SMTP
- **address-string** 一个字符串，包含指定消息系统的有效地址。
- **send-time** 一个字符串，包含 *hh*:*mm*:*ss* 格式的时间说明。

注释

在 SQL Remote 安装中，每个从当前数据库接收消息的数据库都必须被授予 REMOTE 权限。

唯一的例外是 SQL Remote 层次中位于当前数据库上一层的数据库，该数据库必须被授予 CONSOLIDATE 权限。

远程用户按消息系统进行标识，所标识的是统一用户收发消息的方法。地址名必须是消息系统的有效地址，用单引号括起来。

对于 FILE 消息类型，地址是 SQLREMOTE 环境变量所指向目录的子目录。

远程数据库需要使用 GRANT REMOTE 语句来接收消息，但此语句本身并不为远程用户预订任何数据。要预订数据，必须使用数据库抽取实用程序或 CREATE SUBSCRIPTION 语句，为用户 ID 创建对当前数据库中的某个发布的预订。

可选的 SEND EVERY 和 SEND AT 子句指定发送消息的频率。该字符串包含各消息之间的时间长度（对于 SEND EVERY）或一天中发送消息的具体时间（对于 SEND AT）。使用 SEND AT 时，每天发送一次消息。

如果授予用户远程权限时未指定 SEND EVERY 或 SEND AT 子句，则消息代理处理消息，然后停止。要继续运行消息代理，必须确保为每个具有 REMOTE 权限的用户都指定了频率 SEND AT 或 SEND EVERY。

预计在许多统一数据库中，消息代理都连续运行，因此所有远程数据库都指定了 SEND 子句。典型安装可能涉及每天向膝上型电脑用户发送消息 (SEND AT) 和每隔一两个小时向远程服务器发送一次消息 (SEND EVERY)。为提高效率，所使用的时间应尽量相同。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “REMOTE 权限”一节 《SQL Remote》
- “GRANT PUBLISH 语句 [SQL Remote]”一节第 601 页
- “REVOKE REMOTE 语句 [SQL Remote]”一节第 683 页
- “GRANT CONSOLIDATE 语句 [SQL Remote]”一节第 599 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句将远程权限授予用户 SamS，使用 SMTP 电子邮件系统，每隔两小时向 Singer, Samuel 这一地址发送一次消息：

```
GRANT REMOTE TO SamS
TYPE SMTP
ADDRESS 'Singer, Samuel'
SEND EVERY '02:00';
```

GRANT REMOTE DBA 语句 [MobiLink] [SQL Remote]

此语句用于授予用户 ID 远程 DBA 权限。

语法

GRANT REMOTE DBA
TO *userid*, ...
[**IDENTIFIED BY** *password*]

参数

- **IDENTIFIED BY** IDENTIFIED BY 子句对于此语句来说可选。如果包括，则用户口令会变更。

注释

拥有 REMOTE DBA 权限的用户 ID 仅在以下情况下具有完全的 DBA 权限：

- 在 MobiLink 中，当从 SQL Anywhere 同步客户端 (dbmlsync) 实用程序建立连接时，REMOTE DBA 权限使 dbmlsync 能够具有数据库的完全访问权限以进行包含在消息中的所有更改。任何

其它使用同一用户 ID 的连接都不会被授予特殊权限。请参见“dbmlsync 权限”一节 《MobiLink - 客户端管理》。

- 在 SQL Remote 中，当从消息代理建立连接时，REMOTE DBA 权限使消息代理能够具有数据库的完全访问权限以进行包含在消息中的所有更改。任何其它使用同一用户 ID 的连接都不会被授予特殊权限。

REMOTE DBA 权限避免了必须授予用户 ID 完全 DBA 权限，从而避免了与分发 DBA 用户 ID 和口令相关的安全问题。

例如，具有 REMOTE DBA 权限的 SQL Remote 用户 ID 对除 [消息代理] 之外的任何连接均不具有额外的权限。即使 REMOTE DBA 用户的用户 ID 和口令已广泛分发，也不存在安全问题。只要用户 ID 没有被授予除 CONNECT 之外的针对数据库的权限，就没有人可以使用该用户 ID 访问数据库中的数据。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- MobiLink： “启动同步”一节 《MobiLink - 客户端管理》
- SQL Remote： “授予 REMOTE DBA 权限”一节 《SQL Remote》
- “REVOKE REMOTE DBA 语句 [SQL Remote]”一节第 684 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

可以使用以下语句向名为 **dbremote** 的用户 ID 授予 REMOTE DBA 权限：

```
GRANT REMOTE DBA
TO dbremote
IDENTIFIED BY dbremote;
```

GROUP BY 子句

此子句用于对列、别名和函数进行分组，构成 SELECT 语句的组成部分。

语法

GROUP BY
| *group-by-term*, ...]
| *simple-group-by-term*, ... **WITH ROLLUP**
| *simple-group-by-term*, ... **WITH CUBE**
|**GROUPING SETS (** *group-by-term*, ... **)**

group-by-term :
simple-group-by-term
| **(** *simple-group-by-term*, ... **)**
| **ROLLUP (** *simple-group-by-term*, ... **)**
| **CUBE (** *simple-group-by-term*, ... **)**

simple-group-by-term :
expression
| **(** *expression* **)**
| **( )**

参数

- **GROUPING SETS 子句** GROUPING SETS 子句允许从单个查询说明中对多个分组执行集合操作。在 GROUPING SET 子句中指定的每个集都等效于 GROUP BY 子句。

  例如，以下两个查询是等效的：

```
SELECT a, b, SUM( c ) FROM t
GROUP BY GROUPING SETS ( ( a, b ), ( a ), ( b ), ( ) );

SELECT a, b, SUM( c ) FROM t
  GROUP BY a, b
UNION ALL
SELECT a, NULL, SUM( c ) FROM t
  GROUP BY a
UNION ALL
SELECT NULL, b, SUM( c ) FROM t
  GROUP BY b
UNION ALL
SELECT NULL, NULL, SUM( c ) FROM t;
```

  一个分组表达式可能在结果集中反映为 NULL 值，这要根据结果行所属的分组而定。这样一来，便可能弄不清 NULL 是否是其它分组的结果，或者 NULL 是否是基础数据中实际 NULL 值的结果。要区分输入数据中存在的 NULL 值与分组运算符插入的 NULL 值，请使用 GROUPING 函数。请参见“GROUPING 函数 [Aggregate]”一节第 204 页。

  如果在 GROUPING SETS 子句中指定一组空括号 ( )，则会返回一个包含全部集合的行。

  有关在 GROUPING 集中使用空括号的详细信息（包括示例），请参见“指定空分组说明”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **ROLLUP 子句** ROLLUP 子句可用于在单个查询说明中指定多个分组说明，在这一点上它与 GROUPING SETS 子句非常类似。*n* 个 *simple-group-by-term* 的 ROLLUP 子句生成 *n*+1 个分组集，这些分组集的构成方式是：以空括号开头，然后从左到右追加连续的 *group-by-term*。

  例如，以下两个语句是等效的：

```
SELECT a, b, SUM( c ) FROM t
GROUP BY ROLLUP ( a, b );

SELECT a, b, SUM( c ) FROM t
GROUP BY GROUPING SETS ( ( a, b ), a, ( ) );
```

  可以在 GROUPING SETS 子句内使用 ROLLUP 子句。

  有关 ROLLUP 操作的详细信息，请参见“使用 ROLLUP”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **CUBE 子句** CUBE 子句可用于在单个查询说明中指定多个分组说明，在这一点上它与 ROLLUP 和 GROUPING SETS 子句非常类似。CUBE 子句用于表示可由 CUBE 子句中列出的表达式形成的所有可能的组合。

  例如，以下两个语句是等效的：

```
SELECT a, b, SUM( c ) FROM t
GROUP BY CUBE ( a, b, c );

SELECT a, b, SUM( c ) FROM t
GROUP BY GROUPING SETS ( ( a, b, c ), ( a, b ), ( a, c ),
 ( b, c ), a, b, c, () );
```

  可以在 GROUPING SETS 子句内使用 CUBE 子句。

  有关 ROLLUP 操作的详细信息，请参见“使用 CUBE”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **WITH ROLLUP 子句** 这是 ROLLUP 子句的替代语法，是为了与 T-SQL 兼容而提供的。
- **WITH CUBE 子句** 这是 CUBE 子句的替代语法，是为了与 T-SQL 兼容而提供的。

注释

当使用 GROUP BY 子句时，可以按列、别名或函数分组。查询结果中有一行专用于显示每个分组集的各个不同值（或值集）。

另请参见

- “SELECT 语句”一节第 689 页
- “GROUP BY 子句扩展”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 尽管 GROUP BY 子句是核心特性，但 GROUPING SETS、ROLLUP 和 CUBE 是核心 SQL 之外的特性。例如，ROLLUP 子句是特性 T431 的一部分。WITH ROLLUP 和 WITH CUBE 是服务商扩充。

示例

以下示例返回显示订单总数的结果集，然后提供每年（2000 年和 2001 年）的订单数小计。

```
SELECT year ( OrderDate ) Year, Quarter ( OrderDate ) Quarter, count(*)
Orders
FROM SalesOrders
GROUP BY ROLLUP ( Year, Quarter )
ORDER BY Year, Quarter;
```

如同上一个 ROLLUP 操作示例，以下 CUBE 查询示例返回一个显示订单总数的结果集，并提供每年（2000 年和 2001 年）的订单数小计。与 ROLLUP 不同，此查询还给出了每个季度（1、2、3 和 4）的订单数小计。

```
SELECT year (OrderDate) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM SalesOrders
GROUP BY CUBE ( Year, Quarter )
ORDER BY Year, Quarter;
```


以下示例返回一个提供 2000 年和 2001 年订单数小计的结果集。GROUPING SETS 操作可用于选择要进行小计的列，而不是像 CUBE 操作那样返回小计的所有组合。

```
SELECT year (OrderDate) Year, Quarter ( OrderDate ) Quarter, count(*) Orders
FROM SalesOrders
GROUP BY GROUPING SETS ( ( Year, Quarter ), ( Year ) )
ORDER BY Year, Quarter;
```

HELP 语句 [Interactive SQL]

此语句用于在 Interactive SQL 环境中接收帮助。

语法

HELP ['*topic*']

注释

HELP 语句用于访问 SQL Anywhere 文档。

可以有选择性地指定帮助的 *topic*。必须用单引号将 *topic* 括起来。在某些帮助格式中，无法指定主题；此时，会出现 Interactive SQL 常规帮助页的链接。

可以指定以下 *topic* 值：

- SQL Anywhere 错误代码
- SQL 语句关键字（例如 INSERT、UPDATE、SELECT、CREATE DATABASE）

权限

无。

副作用

无。

另请参见

- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

IF 语句

此语句用于控制 SQL 语句的条件执行。

语法

IF *search-condition* **THEN** *statement-list*
[**ELSEIF** { *search-condition* | *operation-type* } **THEN** *statement-list*] ...

[**ELSE** *statement-list*]
{ **END IF** | **ENDIF** }

注释

IF 语句是控制语句，它允许按条件执行第一个 *search-condition* 计算为 TRUE 的 SQL 语句列表。如果没有 *search-condition* 计算为 TRUE，且存在 ELSE 子句，则执行 ELSE 子句中的 *statement-list*。

执行在 END IF 后的第一个语句继续。

IF 语句不同于 IF 表达式
不要混淆 IF 语句和 IF 表达式的语法。

有关 IF 表达式的信息，请参见“IF 表达式”一节第 18 页。

权限

无。

副作用

无。

另请参见

- “BEGIN 语句”一节第 395 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》
- “搜索条件”一节第 33 页

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

下面的过程阐释了 IF 语句的用法：

```
CREATE PROCEDURE TopCustomer2 (OUT TopCompany CHAR(35), OUT TopValue INT)
BEGIN
   DECLARE err_notfound EXCEPTION
   FOR SQLSTATE '02000';
   DECLARE curThisCust CURSOR FOR
   SELECT CompanyName, CAST(    sum(SalesOrderItems.Quantity *
   Products.UnitPrice) AS INTEGER) VALUE
   FROM Customers
   LEFT OUTER JOIN SalesOrders
   LEFT OUTER JOIN SalesOrderItems
   LEFT OUTER JOIN Products
   GROUP BY CompanyName;
   DECLARE ThisValue INT;
   DECLARE ThisCompany CHAR(35);
   SET TopValue = 0;
   OPEN curThisCust;
   CustomerLoop:
   LOOP
      FETCH NEXT curThisCust
      INTO ThisCompany, ThisValue;
      IF SQLSTATE = err_notfound THEN
         LEAVE CustomerLoop;
```



```
      END IF;
      IF ThisValue > TopValue THEN
         SET TopValue = ThisValue;
         SET TopCompany = ThisCompany;
      END IF;
   END LOOP CustomerLoop;
   CLOSE curThisCust;
END;
```

IF 语句 [T-SQL]

此语句用于控制 SQL 语句的条件执行，是 Watcom-SQL IF 语句的替代方法。

语法

IF *expression statement*
[**ELSE** [**IF** *expression*] *statement*]

注释

Transact-SQL IF 条件和 ELSE 条件各控制一个 SQL 语句或复合语句（在关键字 BEGIN 和 END 之间）的执行。

与 Watcom-SQL IF 语句相比，Transact-SQL IF 语句中没有 THEN。Transact-SQL 版本也没有 ELSEIF 或 END IF 关键字。

权限

无。

副作用

无。

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

下面的示例阐释了 Transact-SQL IF 语句的用法：

```
IF (SELECT max(ID) FROM sysobjects) < 100
   RETURN
ELSE
         BEGIN
      PRINT 'These are the user-created objects'
      SELECT name, type, ID
      FROM sysobjects
      WHERE ID < 100
   END
```

以下两个语句块说明了 Transact-SQL 和 Watcom-SQL 的兼容性：

```
/* Transact-SQL IF statement */
IF @v1 = 0
   PRINT '0'
ELSE IF @v1 = 1
```

```
   PRINT '1'
ELSE
   PRINT 'other'
/* Watcom-SQL IF statement */
IF v1 = 0 THEN
   PRINT '0'
ELSEIF v1 = 1 THEN
   PRINT '1'
ELSE
   PRINT 'other'
END IF
```

INCLUDE 语句 [ESQL]

此语句用于将文件包含在将由 SQL 预处理器扫描的源程序中。

语法

INCLUDE *filename*

filename : **SQLDA** | **SQLCA** | *string*

注释

INCLUDE 语句与 C 预处理器 #include 指令非常相似。SQL 预处理器读取嵌入式 SQL 源文件并用 C 语言源代码替换所有嵌入式 SQL 语句。如果某文件包含 SQL 预处理器所需的信息，则请用嵌入式 SQL INCLUDE 语句包含此文件。

对以下两个文件名进行特殊识别：SQLCA 和 SQLDA。以下语句在所有嵌入式 SQL 源文件中必须出现在任何嵌入式 SQL 语句之前。

```
EXEC SQL INCLUDE SQLCA;
```

此语句必须出现在 C 程序中允许使用静态变量声明的位置。许多嵌入式 SQL 语句需要变量（对程序员不可见），它们由 SQL 预处理器在 SQLCA 包含语句的位置声明。如果使用了任何 SQLDA，则必须包括 SQLDA 文件。

权限

无。

副作用

无。

标准和兼容性

- **SQL/2003** 核心特性。

INPUT 语句 [Interactive SQL]

此语句用于将数据从外部文件或键盘导入到数据库表，或从通用 ODBC 数据源导入数据。

语法 1 - 从外部文件或键盘导入

INPUT INTO [*owner.*]*table-name input-options*

input-options :
[**(** *column-name*, ... **)**]
[**BY** { **ORDER** | **NAME**]
[**BYTE ORDER MARK** { **ON** | **OFF** }]
[**COLUMN WIDTHS (** *integer*, ...**)**]
[**DELIMITED BY** *string*]
[**ENCODING** *encoding*]
[**ESCAPE CHARACTER** *character*]
[**ESCAPES** { **ON** | **OFF** }
[**FORMAT** *input-format*]
[**FROM** *filename* | **PROMPT**]
[**NOSTRIP**]

input-format :
TEXT
| **FIXED**

encoding : *identifier* or *string*

语法 2 - 从 ODBC 数据源导入

INPUT
USING *connection-string*
FROM *source-table-name*
INTO *destination-table-name*
[**CREATE TABLE** { **ON** | **OFF** }]

connection-string :
{ **DRIVER=***odbc_driver_name*
| **DSN=***odbc_data_source* } [**;** { *connection_parameter* **=** *value* }]

参数

- **BY 子句** BY 子句允许用户指定输入文件中的列是基于它们在列表中的顺序位置（ORDER，缺省值）还是按它们的名称 (NAME) 与表列进行匹配。并非所有输入格式在文件中都有列名信息。只有那些具有列名信息的格式才可以按 NAME 匹配。

- **BYTE ORDER MARK 子句** 此子句用于指定是否处理数据中的字节顺序标记 (BOM)。

  仅当从 TEXT 格式的文件中读取时，BYTE ORDER MARK 子句才有意义。如果试图将 BYTE ORDER MARK 子句与 FORMAT（而不是 TEXT）子句一起使用，会返回错误。

  仅当读取或写入以 UTF-8 或 UTF-16（及它们的变体）编码的文件时才使用 BYTE ORDER MARK 子句。如果试图将 BYTE ORDER MARK 子句与任何其它编码一起使用，会返回错误。

如果指定了 ENCODING 子句：

- 如果 BYTE ORDER MARK 选项为 ON，并且指定了具有某一种字节序（如 UTF-16BE 或 UTF-16LE）的 UTF-16 编码，则 Interactive SQL 会在数据的开始位置搜索 BOM。如果 BOM 存在，会将它用于验证数据的字节排序方式。如果指定了错误的字节序，则返回错误。
- 如果 BYTE ORDER MARK 选项为 ON，但指定的是没有显式字节序的 UTF-16 编码，则 Interactive SQL 将在数据的开始位置搜索 BOM。如果 BOM 存在，会将它用于确定数据的字节排序方式。否则，会将其假定为操作系统的字节排序方式。
- 如果 BYTE ORDER MARK 选项为 ON，并指定了 UTF-8 编码，则 Interactive SQL 会在数据的开始位置搜索 BOM。如果 BOM 存在，则会被忽略。

如果未指定 ENCODING 子句：

- 如果未指定 ENCODING 子句而 BYTE ORDER MARK 选项为 ON，则 Interactive SQL 会在输入数据的开始位置搜索 BOM。如果找到 BOM，则会根据 BOM 的编码（UTF-16BE、UTF-16LE 或 UTF-8）自动选择源编码，同时不将 BOM 视作要装载的数据的一部分。
- 如果未指定 ENCODING 子句且 BYTE ORDER MARK 选项为 OFF，或在输入数据的开始位置未找到 BOM，则使用数据库 CHAR 编码。

- **COLUMN WIDTHS 子句** 只能为 FIXED 格式指定 COLUMN WIDTHS。它在输入文件中指定列的宽度。如果没有指定 COLUMN WIDTHS，则宽度由数据库的列类型决定。
- **CREATE TABLE 子句** CREATE TABLE 子句用于指定在目标表不存在时是否创建表。缺省值为 ON。
- **DELIMITED BY 子句** DELIMITED BY 子句允许指定用作 TEXT 输入格式中的分隔符的字符串。缺省分隔符是逗号。
- **ENCODING 子句** *encoding* 参数允许您指定用于读取文件的编码。ENCODING 子句只能与 TEXT 格式配合使用。

  有关如何获得 SQL Anywhere 支持的编码列表的详细信息，请参见“支持的字符集”一节《SQL Anywhere 服务器 - 数据库管理》。

  对于 Interactive SQL，如果未指定 *encoding* 子句，则按以下顺序确定用于读取文件的编码：

  - 用 default_isql_encoding 选项指定的编码（如果设置此选项）
  - 运行 Interactive SQL 的计算机上操作系统字符集的缺省编码

  如果使用 OUTPUT 语句创建输入文件并指定了编码，那么在 INPUT 语句中应指定相同的 ENCODING 子句。

  有关 Interactive SQL 和编码的详细信息，请参见“default_isql_encoding 选项 [Interactive SQL]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  指定 BYTE ORDER 子句，使数据中包含字节顺序标记。

- **ESCAPE CHARACTER 子句** 十六进制代码和符号的缺省转义字符是反斜线 (\)。例如，\x0A 是换行符。

  换行符可以指定为 \n。其它字符可以使用十六进制 ASCII 代码（如使用 \x09 表示制表符）指定。两个连续的反斜线字符 (\\ ) 被解释为单个反斜线。后跟除 n、x、X 或 \ 外的任何字符的反斜线解释为两个单独的字符。例如，\q 被解释为反斜线和字母 q。

使用 ESCAPE CHARACTER 子句可更改转义字符。例如，要将感叹号用作转义字符，可按如下方式指定：

```
... ESCAPE CHARACTER '!'
```

- **ESCAPES 子句** ESCAPES 设置为 ON（缺省值）时，数据库服务器会将转义字符后面的字符解释为特殊字符。当 ESCAPES 设置为 OFF 时，会完全按照字符在源代码中的出现方式读取字符。
- **FORMAT 子句** FORMAT 子句允许指定输出的文件格式。

  如果未指定 FORMAT 子句，那么各组值都必须采用 Interactive SQL 的 SET OPTION *input_format* 语句所设置的格式。

  从命令文件输入的内容由包含 END 的行终止。从文件输入的内容在文件末尾终止。

  允许的输入格式为：

  - **TEXT** 将输入行假定为字符，每个输入行占一行，列值以逗号分隔。字母字符串可以用撇号（单引号）或引号（双引号）括起来。包含分隔符的字符串必须用单引号或双引号括起来。如果字符串本身包含单引号或双引号，则字符串内使用的引号字符需要两个。可以使用 DELIMITED BY 子句指定缺省值（逗号）之外的其它分隔符字符串。

    另外，还识别其它三个特殊序列。两个字符 \n 代表换行符，\\ 代表单个 \；而序列 \xDD 代表具有十六进制代码 DD 的字符。

    如果文件中有一些条目指示某个值可能为空，则它会被视为 NULL。如果该位置的值不能为 NULL，则在数字列中插入零，并在字符列中插入空字符串。
  - **FIXED** 输入行采用固定格式。使用 COLUMN WIDTHS 子句可指定列的宽度。如果未指定列宽，则文件中的列宽必须与相应数据库列类型的任何值所要求的最大字符数相同。

    FIXED 格式不能与包含嵌入式换行符和文件结尾字符序列的二进制列一起使用。

  如果要使用其它格式，如 DBASEII、DBASE III、FoxPro、Lotus 123 或 Excel 97，则需要使用 INPUT USING 语句。

- **FROM filename 子句** *filename* 可以带引号也可以不带引号。如果字符串带引号，则其遵循的格式要求与其它 SQL 字符串一样。

  要指示目录路径，反斜线字符 (\) 必须用两个反斜线来表示。将数据从文件 *c:\temp\input.dat* 装载到 Employees 表的语句为：

  ```
  INPUT INTO Employees
  FROM 'c:\\temp\\input.dat';
  ```

  相关 *filename* 的位置按以下方式确定：

  - 如果直接在 Interactive SQL 中执行 INPUT 语句，则相对于正在运行 Interactive SQL 的目录解析 *filename* 的路径。例如，假设您从 *c:\work* 目录打开 Interactive SQL，并执行以下语句：

    ```
    INPUT INTO Employees
     FROM 'inputs\\inputfile.dat';
    ```

    Interactive SQL 将查找 *c:\work\inputs\inputfile.dat*。

- 如果 INPUT 语句在 *.sql* 文件中，则 Interactive SQL 会首先尝试相对于外部文件所在位置来解析 *filename* 的路径。如果未成功，则 Interactive SQL 会在相对于正运行 Interactive SQL 的目录的路径中查找 *filename*。

  例如，假设存在文件 *c:\homework\inputs.sql*，它包含以下语句：

  ```
  INPUT INTO Employees
   FROM 'inputs\\inputfile.dat';
  ```

  Interactive SQL 会首先在 *c:\homework\inputs* 中查找 *inputfile.dat*。如果 Interactive SQL 在该位置没有找到 *inputfile.dat*，则它会在运行 Interactive SQL 的目录中继续查找。

- **FROM source-table-name 子句** *source-table-name* 参数是一个带引号的字符串，它包含源数据库中的表名。表名的形式可以为 *database-name.owner.table-name*、*owner.table-name* 或仅为 *table-name*。在表名中使用句点分隔各个部分，即使它不是源数据库中的本地分隔符。如果源数据库需要数据库名，但不需要所有者名，则 *source-table-name* 的格式必须为 *database..table*（在本例中所有者名为空）。不要给参数中的任何名称加引号（例如，不要使用 `'dba."my-table"'`，而是使用 `'dba.my-table'`。）

- **INTO 子句** 要输入数据的表的名称。

- **PROMPT 子句** 用户可使用 PROMPT 子句为一行中的各列逐一输入值。在窗口模式下运行时，会显示一个窗口，允许用户为新行输入值。如果在命令行上运行 Interactive SQL，Interactive SQL 会在命令行提示您为每一列键入值。

- **NOSTRIP 子句** 通常，对于 TEXT 输入格式，在插入值之前，将去除无引号字符串中的尾随空白。NOSTRIP 可用于取消去除尾随空白。不管是否使用此选项，带引号字符串中的尾随空白都不会去除。无论 NOSTRIP 选项的设置如何，无引号字符串中的前导空白都会去除。

- **USING 子句** USING 子句从 ODBC 数据源输入数据。可以使用 DSN 选项指定 ODBC 数据源名，或用 DRIVER 选项指定 ODBC 驱动程序名和连接参数。*Connection-parameter* 是数据库特定的连接参数的可选列表。

  odbc-data-source 是用户名称或 ODBC 数据源名称。例如，SQL Anywhere demo 数据库的 *odbc-data-source* 为 SQL Anywhere 11 Demo。

  Odbc-driver-name 是 ODBC 驱动程序名。对于 SQL Anywhere 11 数据库，*odbc-driver-name* 是 SQL Anywhere 11；而对于 UltraLite 数据库，*odbc-driver-name* 为 UltraLite 11。

注释

INPUT 语句允许向指定的数据库表中进行高效的大量插入。通过输入窗口从用户处读取（如果指定了 PROMPT）或从文件中读取（如果指定了 FROM *filename*）输入行。如果两者均未指定，则从包含 INPUT 语句的命令文件中读取输入—在 Interactive SQL 中，甚至可以直接从 [SQL 语句] 窗格中读取。

当直接从 [SQL 语句] 窗格读取输入时，必须在要插入到 INPUT 语句结尾处的记录值前指定分号。例如：

```
INPUT INTO Owner.TableName;
value1, value2, value3
value1, value2, value3
value1, value2, value3
value1, value2, value3
END;
```


END 语句为不指定文件也不包括 PROMPT 关键字的 INPUT 语句终止数据。

如果指定了列的列表，则数据将插入到指定表的指定列中。缺省情况下，INPUT 语句假定输入文件中的列值显示顺序与它们在数据库表定义中显示的顺序相同。如果输入文件的列顺序不同，则必须在 INPUT 语句末尾列出输入文件的实际列顺序。

例如，如果使用以下语句创建一个表：

```
CREATE TABLE inventory (
Quantity INTEGER,
item VARCHAR(60)
);
```

并想从依次包含名称值和数量值的输入文件 *stock.txt* 中导入 TEXT 数据，

```
'Shirts', 100
'Shorts', 60
```

则必须在 INPUT 语句末尾列出输入文件的实际列顺序，才能正确地插入数据：

```
INPUT INTO inventory
FROM stock.txt
FORMAT TEXT
(item, Quantity);
```

缺省情况下，当 INPUT 语句试图插入导致错误的行时会停止。通过设置 on_error 和 conversion_error 选项，可以通过不同的方式来处理错误（请参见 SET OPTION）。如果在 INPUT 上截断任何字符串值，则 Interactive SQL 会在 [消息] 选项卡上输出警告。NOT NULL 列中缺少的值被设置为零（对于数值类型）和空字符串（对于非数值类型）。如果 INPUT 试图插入 NULL 行，则输入文件包含空行。

由于 INPUT 语句是 Interactive SQL 命令，它不能用在任何复合语句（例如 IF）或存储过程中。

请参见“过程、触发器、事件和批处理中允许使用的语句”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

必须有表或视图的 INSERT 权限。

副作用

无。

另请参见

- “OUTPUT 语句 [Interactive SQL]”一节第 650 页
- “INSERT 语句”一节第 616 页
- “SET OPTION 语句 [Interactive SQL]”一节第 704 页
- “LOAD TABLE 语句”一节第 626 页
- “导入数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下是一个 TEXT 文件中的 INPUT 语句的示例。

```
INPUT INTO Employees
FROM new_emp.inp
FORMAT TEXT;
```

以下虚构示例将表 ulTest 复制到名为 saTest 的表中。ulTest 是位于文件 *C:\test\myULDatabase.udb* 中的 UltraLite 数据库中的表，saTest 是在 *demo.db* 中创建的表：

```
INPUT USING 'driver=Ultralite 11;dbf=C:\\test\\myULDatabase.udb'
            FROM "ulTest" INTO "saTest";
```

INSERT 语句

此语句用于将单行（语法 1）或从数据库的其它位置选择的一组行（语法 2）插入到表中。

语法 1

INSERT [**INTO**] [*owner.*]*table-name* [**(** *column-name*, ... **)**]
[**ON EXISTING** {
  ERROR
  | **SKIP**
  | **UPDATE** [**DEFAULTS** { **ON** | **OFF** }]
  }]
{ **DEFAULT VALUES**
  | **VALUES (** [*expression* | **DEFAULT**, ...**)**] }
[**OPTION(** *query-hint*, ... **)**]

语法 2

INSERT [**INTO**] [*owner.*]*table-name* [**(** *column-name*, ... **)**]
[**ON EXISTING** {
  ERROR
  | **SKIP**
  | **UPDATE** [**DEFAULTS** { **ON** | **OFF** }]
  }]
[**WITH AUTO NAME**]
select-statement
[**OPTION(** *query-hint*, ... **)**]

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| **FORCE NO OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

参数

- **VALUES 子句** 使用 VALUES 子句可指定要插入的值。如果要插入为列定义的缺省值，请指定 DEFAULT VALUES。还可指定 VALUES ()，其等价于 DEFAULT VALUES。
- **WITH AUTO NAME 子句** WITH AUTO NAME 仅适用于语法 2。如果指定 WITH AUTO NAME，则 SELECT 语句中的项的名称确定这些数据属于哪个列。SELECT 语句项应当是列引用或带别名的表达式。SELECT 语句中未定义的目标列将被赋予缺省值。当目标表中所包含的列数很大时，可以使用此子句。
- **ON EXISTING 子句** INSERT 语句的 ON EXISTING 子句对这两种语法都适用。它根据主键查找将表中的现有行更新为新列值。此子句仅可用于拥有主键的表。如果尝试对没有主键的表使用此子句，则将造成语法错误。不能使用 ON EXISTING 子句向代理表插入值。

注意
如果预期有许多满足 ON EXISTING 条件的行，则应考虑使用 MERGE 语句。MERGE 语句提供更多对用于匹配行的操作的控制。该语句还提供更复杂的语法来定义构成匹配的元素。请参见“MERGE 语句”一节第 638 页。

如果指定 ON EXISTING 子句，则数据库服务器为每个输入行分别执行主键查找。如果表中不存在对应的行，它会插入新行。对于表中已经存在的行，您可以选择在没有任何提示的情况下忽略输入行 (SKIP)，为重复键值生成错误消息 (ERROR)，或使用输入行中的值更新旧值 (UPDATE)。缺省情况下，如果未指定 ON EXISTING 子句，则在尝试向表中插入行时，如果相应的行已存在，则会造成重复键值错误，而这相当于指定 ON EXISTING ERROR 子句。

当使用 ON EXISTING UPDATE 子句时，将输入行与存储行进行比较。在输入行中显式指定的任何列值将替换存储行中的相应列值。同样，如果未在输入行中显式指定列值，则不会更改存储行中的相应列值—具有缺省值的列除外。在将 ON EXISTING UPDATE 子句用于有缺省值的列（包括 DEFAULT AUTOINCREMENT 列）时，您可进一步指定是通过指定 ON EXISTING UPDATE DEFAULTS ON 来更新具有缺省值的列值，还是通过指定 ON EXISTING UPDATE DEFAULTS OFF 使列值保持原样。如果未做任何指定，则缺省行为将是 ON EXISTING UPDATE DEFAULTS OFF。

注意
DEFAULTS ON 和 DEFAULTS OFF 参数不会影响 DEFAULT TIMESTAMP、DEFAULT UTC TIMESTAMP 或 DEFAULT LAST USER 中的值。对于这些列，在执行 UPDATE 期间会始终更新存储行中的值。

使用 ON EXISTING SKIP 和 ON EXISTING ERROR 子句时，如果表包含缺省列，则服务器会计算缺省值，即使对于已存在的行也会如此。因此，缺省值（例如 AUTOINCREMENT）即使对跳过的行也会产生副作用。对于本例中的 AUTOINCREMENT，这导致在 AUTOINCREMENT 序列中跳过某些值。以下示例对此进行说明：

```
CREATE TABLE t1( c1 INT PRIMARY KEY, c2 INT DEFAULT AUTOINCREMENT );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 20 );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 20 );
INSERT INTO t1( c1 ) ON EXISTING SKIP VALUES( 30 );
```

插入在第一个 INSERT 语句中定义的行，并且将 c2 设置为 1。在第二个 INSERT 语句中定义的行将被跳过，原因是它与现有的行匹配。但是，自动增量计数器仍然会增量到 2（不过不会影

响现有行）。插入在第三个 INSERT 语句中定义的行，将 c2 的值设置为 3。因此，以上示例的插入值是：

```
20,1
30,3
```

小心
如果正在使用 SQL Remote，则请勿复制 DEFAULT LAST USER 列。复制列以后，列值会设置为 SQL Remote 用户，而不是所复制的值。

- **OPTION 子句** 此子句用于指定执行语句时的提示。支持以下提示：
  - MATERIALIZED VIEW OPTIMIZATION *option-value*
  - FORCE OPTIMIZATION
  - FORCE NO OPTIMIZATION
  - *option-name* = *option-value*

  有关这些选项的说明，请参见“SELECT 语句”一节第 689 页的 OPTIONS 子句。

注释

INSERT 语句用于将新行添加到数据库表中。

由于对基础表数据的更改影响文本索引和实例化视图，因此在将数据批量装载（LOAD TABLE、INSERT、MERGE）到基础表之前，考虑截断相关的文件索引或实例化视图。请参见“TRUNCATE 语句”一节第 727 页和“TRUNCATE TEXT INDEX 语句”一节第 728 页。

语法 1 插入包含指定表达式列值的单个行。关键字 DEFAULT 可用于插入列的缺省值。如果给出可选的列名列表，则值将逐个插入到指定的列中。如果未指定列名的列表，则值将以创建时所使用的顺序（与用 SELECT * 检索的顺序相同）插入到表列中。行插入到表中的任意位置。（在关系数据库中，表是不排序的）。

语法 2 向表中执行大量插入，其中包含完全通用的 SELECT 语句的结果。除非 SELECT 语句包含 ORDER BY 子句，否则插入将以任意顺序进行。请参见“SELECT 语句”一节第 689 页。

如果指定列名称，选择列表中的列将按顺序与列列表中指定的列匹配，或者按这些列的创建顺序匹配。

如果定义视图的查询说明是可更新的，则可以对视图执行插入操作。有关识别固有*不可更新的*视图的详细信息，请参见“使用常规视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

插入到表中的字符串始终以它们输入时采用的大小写进行保存，而不论数据库是否区分大小写。因此，插入到表中的字符串 Value 在数据库中保存时 V 始终为大写，其余的字母为小写。SELECT 语句返回的字符串为 Value。但是，如果数据库不区分大小写，所有比较都会将 Value 与 value、VALUE 等不同大小写形式的字符串视为相同。而且，如果单列主键已经包含 Value 条目，则会拒绝对 value 执行 INSERT，因为它会导致主键不唯一。

使用 INSERT 语句插入大量数据也会更新列的统计信息。

性能提示
要将许多行插入到表中，应尽可能声明一个游标并通过游标插入行，这比执行许多单独的 INSERT 语句效率要高。插入数据之前，可以指定每个表页上保留供以后的更新使用的可用空间的百分比。请参见“ALTER TABLE 语句”一节第 373 页。

权限

必须有表的 INSERT 权限。

如果指定了 ON EXISTING UPDATE 子句，还需要具有表的 UPDATE 权限。

副作用

无。

另请参见

- “导入数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “MERGE 语句”一节第 638 页
- “INPUT 语句 [Interactive SQL]”一节第 610 页
- “LOAD TABLE 语句”一节第 626 页
- “UPDATE 语句”一节第 735 页
- “DELETE 语句”一节第 530 页
- “PUT 语句 [ESQL]”一节第 660 页
- “访问客户端计算机上的数据”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心特性。INSERT ...ON EXISTING 是服务商扩充。

示例

将 Eastern Sales 部门添加到数据库。

```
INSERT
INTO Departments ( DepartmentID, DepartmentName )
VALUES ( 230, 'Eastern Sales' );
```

创建表 DepartmentHead，然后使用 WITH AUTO NAME 语法用部门主管姓名和部门名填充该表。

```
CREATE TABLE DepartmentHead(
      pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
      DepartmentName VARCHAR(128),
      ManagerName VARCHAR(128) );
INSERT
INTO DepartmentHead WITH AUTO NAME
SELECT GivenName || ' ' || Surname AS ManagerName,
      DepartmentName
FROM Employees JOIN Departments
ON EmployeeID = DepartmentHeadID;
```

创建表 MyTable5，并使用 WITH AUTO NAME 语法填充此表。

```
CREATE TABLE MyTable5(
      pk INT PRIMARY KEY DEFAULT AUTOINCREMENT,
      TableName CHAR(128),
      TableNameLen INT );
```

```
INSERT INTO MyTable5 WITH AUTO NAME
SELECT
      length(t.table_name) AS TableNameLen,
      t.table_name AS TableName
FROM SYS.SYSTAB t
WHERE table_id <= 10;
```

以隔离级别 3 而不是使用数据库的当前隔离级别设置执行此语句，插入新部门。

```
INSERT INTO Departments
   (DepartmentID, DepartmentName, DepartmentHeadID)
   VALUES(600, 'Foreign Sales', 129)
   OPTION( isolation_level = 3 );
```

INSTALL EXTERNAL OBJECT 语句

此语句用于安装可在外部环境中运行的对象。

语法

INSTALL EXTERNAL OBJECT *object-name*
[*update-mode*]
FROM { **FILE** *file-path* | **VALUE** *expression* }
ENVIRONMENT *environment-name*

environment-name :
PERL
| **PHP**

update-mode :
NEW
| **UPDATE**

参数

- **object-name** 用于在数据库内标识安装对象的名称。
- **update-mode** 对象的更新模式。如果忽略更新模式，则假设为 NEW。
- **file-path** 在服务器计算机上要从中安装对象的位置。
- **environment-name** 运行外部对象的外部环境名称。

注释

有关外部环境的详细信息，请参见“外部环境概述”一节 《SQL Anywhere 服务器 - 编程》。

权限

必须具有 DBA 权限。

副作用

无

另请参见

- “外部环境概述”一节 《SQL Anywhere 服务器 - 编程》
- “ALTER EXTERNAL ENVIRONMENT 语句”一节第 353 页
- “REMOVE EXTERNAL OBJECT 语句”一节第 672 页
- “START EXTERNAL ENVIRONMENT 语句”一节第 713 页
- “STOP EXTERNAL ENVIRONMENT 语句”一节第 720 页
- “SYSEXTERNENV 系统视图”一节第 947 页
- “SYSEXTERNENVOBJECT 系统视图”一节第 948 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

此示例将位于文件中的 Perl 脚本安装到数据库中。

```
INSTALL EXTERNAL OBJECT 'PerlScript'
NEW
FROM FILE 'perlfile.pl'
ENVIRONMENT PERL;
```

也可以从表达式构建和安装 Perl 代码，如下所示：

```
INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE 'sub WriteToServerConsole { print $sa_output_handle $_[0]; }'
ENVIRONMENT PERL;
```

还可以从变量构建和安装 Perl 代码，如下所示：

```
CREATE VARIABLE PerlVariable LONG VARCHAR;
SET PerlVariable =
  'sub WriteToServerConsole { print $sa_output_handle $_[0]; }';

INSTALL EXTERNAL OBJECT 'PerlConsoleExample'
NEW
FROM VALUE PerlVariable
ENVIRONMENT PERL;
```

INSTALL JAVA 语句

此语句用于使 Java 类可在数据库中使用。

语法

INSTALL JAVA
[**NEW** | **UPDATE**]
[**JAR** *jar-name*]
FROM { **FILE** *filename* | *expression* }

参数

- **NEW 和 UPDATE keyword 子句** 如果指定 NEW 安装模式，则引用的 Java 类必须是新类，而不是当前安装的类的更新。如果数据库中存在同名的类，且使用 NEW 安装模式，则会出现错误。

  如果指定 UPDATE，则引用的 Java 类可以包含给定数据库中已安装的 Java 类的替换类。

  如果省略 *install-mode*，则缺省模式为 NEW。

- **JAR 子句** 如果指定了此子句，则 *filename* 必须指定一个 jar 文件。JAR 文件的扩展名通常为 *.jar* 或 *.zip*。

  可以压缩或解压缩已安装的 jar 和 zip 文件。

  如果指定了 JAR 选项，则在安装了 jar 包含的类后，该 jar 将保留为 jar 文件。该 jar 是与这些类的每一个关联的 jar。使用 JAR 选项安装在数据库中的 jar 称为数据库的保留 jar。

  jar-name 是最大长度为 255 字节的字符串值。*jar-name* 用于在后面的 INSTALL JAVA、UPDATE 和 REMOVE JAVA 语句中标识保留的 jar。

- **FROM FILE 子句** 指定安装 Java 类的位置。

  file-name 支持的格式包括完全限定文件名（例如 '*c:\libs\jarname.jar*' 和 '*/usr/u/libs/jarname.jar*'）和相对文件名（相对于数据库服务器的当前工作目录）。

  filename 必须标识类文件或者 jar 文件。

- **FROM 子句** 表达式必须计算为二进制类型，其值必须包含有效的类文件或 jar 文件。

注释

每个类的类定义是在首次使用该类时由每个连接的 VM 装载的。当您 INSTALL（安装）类时，将隐式重新启动连接的 VM。因此，不管 INSTALL 的 *install-mode* 是 NEW 还是 UPDATE，都可以直接访问新类。由于 VM 重新启动，所以存储在 Java 静态变量中的所有值都将丢失，任何类型为 Java 类的 SQL 变量也会被删除。

对于其它连接，新类会在下次 VM 首次访问该类时装载。如果该类已由 VM 装载，则在为该连接重新启动 VM 后，该连接才能看到新类。

权限

需要 DBA 权限才能执行 INSTALL JAVA 语句。

任何用户可以任何方式引用所有已安装的类。

Windows Mobile 上不支持。

另请参见

- “REMOVE JAVA 语句”一节第 673 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句通过提供文件名和类的位置，安装用户创建的 Java 类 Demo。

```
INSTALL JAVA NEW
FROM FILE 'D:\JavaClass\Demo.class';
```

以下语句安装包含在 zip 文件中的所有类，并在数据库内将它们与 JAR 文件名关联。

```
INSTALL JAVA
JAR 'Widgets'
FROM FILE 'C:\Jars\Widget.zip';
```

同样，不保留 zip 文件的位置，并且必须使用完全限定的类名（包名和类名）引用类。

INTERSECT 子句

计算两个或多个查询的结果集的交集。

语法

[**WITH** *temporary-views*] *query-block*
INTERSECT [**ALL** | **DISTINCT**] *query-block*
[**ORDER BY** [*integer* | *select-list-expression-name*] [**ASC** | **DESC**], ...]
[**FOR XML** *xml-mode*]
[**OPTION(** *query-hint*, ... **)**]

query-block：请参见“SQL 语法中的常见元素”一节第 340 页

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* **=** *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

参数

- **OPTION 子句** 此子句用于指定执行语句时的提示。支持以下提示：
  - MATERIALIZED VIEW OPTIMIZATION *option-value*
  - FORCE OPTIMIZATION
  - *option-name* = *option-value*

  有关这些选项的说明，请参见“SELECT 语句”一节第 689 页的 OPTIONS 子句。

注释

通过使用 INTERSECT 或 INTERSECT ALL，可以一次获得多个查询块的结果集的交集。INTERSECT DISTINCT 与 INTERSECT 相同。

每个查询块在选择列表中必须具有相同数目的项目。

INTERSECT 的结果和 INTERSECT ALL 相同，只是在使用 INTERSECT 时，重复的行在计算结果集的交集之前已被排除。

如果两个选择列表中的相应项具有不同的数据类型，则 SQL Anywhere 为结果中的相应列选择数据类型，并自动相应地转换各 *query-block* 中的列。UNION 的第一个 *query-block* 用于确定与 ORDER BY 子句匹配的名称。

显示的列名与为第一个 *query-block* 显示的列名相同。另一种自定义结果集列名的方法是在 *query-block* 上使用 WITH 子句。

权限

必须具有对各 *query-block* 的 SELECT 权限。

副作用

无。

另请参见

- “EXCEPT 子句”一节第 565 页
- “INTERSECT 子句”一节第 623 页
- “UNION 子句”一节第 729 页
- “SELECT 语句”一节第 689 页

标准和兼容性

- **SQL/2003** 特性 F302。

示例

有关 INTERSECT 用法的示例，请参见“集合运算符和 NULL”一节 《SQL Anywhere 服务器 - SQL 的用法》。

LEAVE 语句

此语句用于退出复合语句或循环。

语法

LEAVE *statement-label*

注释

LEAVE 语句是控制语句，允许您退出带标签的复合语句或带标签的循环。执行在复合语句或循环之后的第一个语句重新开始。

复合语句是过程或触发器的主体，它具有与过程或触发器同名的隐式标签。

权限

无。

副作用

无。

另请参见

- “LOOP 语句”一节第 637 页
- “FOR 语句”一节第 577 页
- “BEGIN 语句”一节第 395 页
- “使用过程、触发器和批处理” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

下面的代码段显示如何使用 LEAVE 语句退出循环。

```
SET i = 1;
lbl:
LOOP
   INSERT
   INTO Counters ( number )
   VALUES ( i );
   IF i >= 10 THEN
      LEAVE lbl;
   END IF;
   SET i = i + 1
END LOOP lbl
```

以下代码段示例在嵌套循环中使用 LEAVE。

```
outer_loop:
LOOP
   SET i = 1;
   inner_loop:
   LOOP
      ...
      SET i = i + 1;
      IF i >= 10 THEN
         LEAVE outer_loop
      END IF
   END LOOP inner_loop
END LOOP outer_loop
```

LOAD STATISTICS 语句

仅供内部使用。此语句将统计信息装载到 ISYSCOLSTAT 系统表中。dbunload 实用程序使用此语句从旧数据库中卸载列统计信息。不得手工使用此语句。

语法

LOAD STATISTICS [[*owner*.]*table-name*.]*column-name*
format-id, *density*, *max-steps*, *actual-steps*, *step-values*, *frequencies*

参数

- **format-id** 内部字段，用于确定 ISYSCOLSTAT 系统表中其余行的格式。
- **density** 估计的列的单值加权平均选择性，不计算存储在行中的单值选择性大的选择性。
- **max-steps** 直方图中允许的最大梯级数。
- **actual-steps** 此时实际使用的梯级数。
- **step-values** 直方图梯级的界限值。
- **frequencies** 直方图梯级的选择性。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “ISYSCOLSTAT 系统表”一节第 757 页
- “卸载实用程序 (dbunload)”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

LOAD TABLE 语句

此语句用于将批量数据从外部文件导入到数据库表中。

语法

LOAD [**INTO**] **TABLE** [*owner.*]*table-name*
[**(** *column-name*, ... **)**]
load-source
[*load-option* ...]
[*statistics-limitation-option*]

load-source :
{ **FROM** *filename-expression*
| **USING FILE** *filename-expression*
| **USING CLIENT FILE** *client-filename-expression*
| **USING VALUE** *value-expression*
| **USING COLUMN** *column-expression* }

filename-expression : string | variable

client-filename-expression : string | variable

value-expression : expression

column-expression :
column-name
  FROM *table-name*
  ORDER BY *column-list*

load-option :
BYTE ORDER MARK { **ON** | **OFF** }
| **CHECK CONSTRAINTS** { **ON** | **OFF** }
| { **COMPRESSED** | **AUTO COMPRESSED** | **NOT COMPRESSED** }
| { **ENCRYPTED KEY** '*key*' | **NOT ENCRYPTED** }
| **COMMENTS INTRODUCED BY** *comment-prefix*
| **COMPUTES** { **ON** | **OFF** }
| **DEFAULTS** { **ON** | **OFF** }
| **DELIMITED BY** *string*
| **ENCODING** *encoding*
| **ESCAPE CHARACTER** *character*
| **ESCAPES** { **ON** | **OFF** }
| **FORMAT** { **TEXT** | **BCP** }
| **HEXADECIMAL** { **ON** | **OFF** }
| **ORDER** { **ON** | **OFF** }
| **PCTFREE** *percent-free-space*
| **QUOTE** *string*
| **QUOTES** { **ON** | **OFF** }
| **ROW DELIMITED BY** *string*
| **SKIP** *integer*
| **STRIP** { **ON** | **OFF** | **LTRIM** | **RTRIM** | **BOTH** }
| **WITH CHECKPOINT** { **ON** | **OFF** }
| **WITH** { **FILE NAME** | **ROW** | **CONTENT** } **LOGGING**

statistics-limitation-option :
STATISTICS {
  ON [**ALL COLUMNS**]
  | **ON KEY COLUMNS**
  | **ON (** *column-list* **)**
  | **OFF**
  }

comment-prefix : string

encoding : string

参数

- **column-name**　此子句用于指定要装载数据的一个或多个列。如果 DEFAULTS 选项为 OFF，则将列的列表中不存在的任何列都变为 NULL。如果 DEFAULTS 为 ON 并且列具有缺省值，则使用该值。如果 DEFAULTS 为 OFF 并且列列表中省略了不可为空的列，则数据库服务器会试图将空字符串转换为该列的类型。

  如果指定了列列表，它会列出文件中应存在的列以及列的显示顺序。列名不能重复。列表中未出现的列名将设置为 NULL/零/空或 DEFAULT（取决于列的为空性、数据类型和 DEFAULTS 设置）。使用列名 **filler()** 可指定存在于输入文件中但是要被 LOAD TABLE 语句忽略的列。

- **load-source**　此子句用于指定要从中装载数据的数据源。可以从多个数据源中装载数据。以下列表列出了支持的装载源：

- **FROM 子句** 此子句用于指定文件。将 *filename-expression* 以字符串的形式传递到数据库服务器。因此，该字符串遵循的数据库格式要求与其它 SQL 字符串一样。尤其是：
  - 要指示目录路径，反斜线字符 (\) 必须用两个反斜线来表示。将数据从文件 *c:\temp\input.dat* 装载到 Employees 表的语句为：

    ```
    LOAD TABLE Employees
    FROM 'c:\\temp\\input.dat' ...
    ```

  - 路径名相对于数据库服务器，而不是客户端应用程序。
  - 可以使用 UNC 路径名从数据库服务器以外的计算机上的文件装载数据。
- **USING FILE 子句** 此子句用于从文件装载数据。其作用等同于指定 FROM *filename* 子句。
- **USING CLIENT FILE 子句** 此子句用于从客户端计算机上的文件装载数据。当数据库服务器从 *client-filename-expression* 检索数据时，数据在服务器的内存中没有实例化，所以数据库服务器对 BLOB 表达式大小的限制并不适用于文件。因此，客户端文件可以具有任意大小。
- **USING VALUE 子句** 此子句用于从任何类型为 CHAR、NCHAR、BINARY 或 LONG BINARY 的表达式中或 BLOB 字符串中装载数据。以下是此子句的用法示例：
  - 以下语法使用 xp_read_file 系统过程从目标文件获取要装载的值：

    ```
    ... USING VALUE xp_read_file( 'filename' )...
    ```

  - 以下语法直接指定值，并插入其值分别为 4 和 5 的两行：

    ```
    ... USING VALUE '4\n5'...
    ```

  - 以下语法将 READ_CLIENT_FILE 函数的结果作为值使用：

    ```
    ... USING VALUE READ_CLIENT_FILE( client-filename-expression )
    ```

    在本例中，也可以指定 `USING CLIENT FILE client-filename-expression`，因为它们在语义上是等效的。

  如果在 LOAD TABLE 语句中没有指定 ENCODING 子句，则如果值为 CHAR 或 BINARY 类型则假定用数据库字符集 (db_charset) 对值进行编码，而在值为 NCHAR 类型的情况下则假定使用 NCHAR 数据库字符集 (nchar_charset) 对值进行编码。
- **USING COLUMN 子句** 此子句用于从另一个表的单个列装载数据。数据库服务器在通过重新使用 LOAD TABLE ...WITH CONTENT LOGGING 语句进行恢复期间重新使用事务日志时会用到此子句。LOAD TABLE ...WITH CONTENT LOGGING 语句的事务日志记录由连接在一起的行组成的块构成。恢复期间当数据库服务器在事务日志中遇到这些块时，会将这些块装入临时表中，然后再从原始装载操作中装载所有的数据。

  可以在 USING COLUMN 子句中使用以下子句：
  - **table-name** 包含要从中装载数据的列的基表或临时表的名称。当从事务日志进行恢复期间被数据库服务器使用时，该表即为保存待分析和装载的行块的表。
  - **column-name** *table-name* 中列的名称，该列用于保存要装载的行块。
  - **column-list** 目标表中的一个或多个列，用于在装载数据前对行进行排序。
  - **load-option 子句** 可以通过指定若干装载选项来控制数据的装载方式。下面的列表给出了所支持的装载选项：

- **BYTE ORDER MARK 子句** 此子句用于指定字节顺序标记 (BOM) 是否出现在编码中。缺省情况下，此选项为 ON，即允许服务器在数据的开始位置搜索并解释字节顺序标记 (BOM)。如果 BYTE ORDER MARK 为 OFF，则服务器不搜索 BOM。

  如果指定了 ENCODING 子句：

  - 如果 BYTE ORDER MARK 选项为 ON，并且指定了具有某一种字节序（如 UTF-16BE 或 UTF-16LE）的 UTF-16 编码，则数据库服务器会在数据的开始位置搜索 BOM。如果 BOM 存在，会将它用于验证数据的字节排序方式。如果指定了错误的字节序，则返回错误。
  - 如果 BYTE ORDER MARK 选项为 ON，但指定的是没有显式字节序的 UTF-16 编码，则数据库服务器将在数据的开始位置搜索 BOM。如果 BOM 存在，会将它用于确定数据的字节排序方式。否则，会将其假定为操作系统的字节排序方式。
  - 如果 BYTE ORDER MARK 选项为 ON，并指定了 UTF-8 编码，则数据库服务器会在数据的开始位置搜索 BOM。如果 BOM 存在，则会被忽略。

  如果未指定 ENCODING 子句：

  - 如果未指定 ENCODING 子句而 BYTE ORDER MARK 选项为 ON，则数据库服务器会在输入数据的开始位置搜索 BOM。如果找到 BOM，则会根据 BOM 的编码（UTF-16BE、UTF-16LE 或 UTF-8）自动选择源编码，同时不将 BOM 视作要装载的数据的一部分。
  - 如果未指定 ENCODING 子句且 BYTE ORDER MARK 选项为 OFF，或在输入数据的开始位置未找到 BOM，则使用数据库 CHAR 编码。

- **CHECK CONSTRAINTS 子句** 此子句用于控制装载期间是否检查约束。缺省情况下，CHECK CONSTRAINTS 为 ON，但是在 CHECK CONSTRAINTS 为 OFF 时，卸载实用程序 (dbunload) 可写出 LOAD TABLE 语句。将 CHECK CONSTRAINTS 设置为 OFF 禁用检查约束，这很有用，例如，在数据库重建过程。如果表的检查约束调用尚未创建的用户定义函数，则重建将失败，除非将 CHECK CONSTRAINTS 设置为 OFF。

- **COMMENTS INTRODUCED BY 子句** 使用此子句可指定用于在数据文件中引入注释的字符串。使用此选项之后，LOAD TABLE 将忽略任何以字符串 *comment-prefix* 开头的行。例如，在以下示例中，*input.dat* 中以 // 开头的行将被忽略。

  ```
  LOAD TABLE Employees FROM 'c:\\temp\\input.dat' COMMENTS INTRODUCED
  BY '//' ...
  ```

  仅允许在一个新行的开始处使用注释。

  如果省略 COMMENTS INTRODUCED BY，则数据文件不得包含注释，因为此时会将这些注释解释为数据。

- **COMPRESSED 子句** 如果要装载的数据在输入文件中以压缩形式存在，请指定 COMPRESSED。数据库服务器会在装载数据前对其进行解压缩。如果指定了 COMPRESSED 但数据并未压缩，则 LOAD 失败并返回错误。

  指定 AUTO COMPRESSED 可允许数据库服务器确定输入文件中的数据是否经过压缩。如果经过压缩，则数据库服务器会在装载数据前对其进行解压缩。

  如果指定 NOT COMPRESSED，则表示输入文件中的数据未经压缩。如果数据经过压缩，但不希望数据库服务器将其解压缩，这时也可以指定 NOT COMPRESSED。这种情

况下，数据在数据库中仍保持加密状态。但是，如果同时对文件进行了加密和压缩，则 NOT ENCRYPTED 只有在同时使用了 NOT COMPRESSED 的情况下才可以使用。

- **COMPUTES 子句** 缺省情况下，此选项为 ON，即允许对已计算的列进行重新计算。将 COMPUTES 设置为 OFF 会禁用计算列的重新计算。COMPUTES 设置为 OFF 很有用，例如，在您重建数据库但表的某个计算列调用尚未创建的用户定义函数时。如果此选项不设置为 OFF，则重建将失败。

  COMPUTES 设置为 OFF 时卸载实用程序 (dbunload) 会写出 LOAD TABLE 语句。

- **DEFAULTS 子句** 缺省情况下，DEFAULTS 被设置为 OFF。如果 DEFAULTS 为 OFF，则将任何未显示在列的列表中且可为空的列置为 NULL。如果将 DEFAULTS 设置为 OFF 并且列的列表中省略了不可为空的列，则数据库服务器会试图将空字符串转换为该列的类型。如果将 DEFAULTS 设置为 ON 并且列具有缺省值，则使用该值。

- **DELIMITED BY 子句** 此子句用于指定列分隔符字符串。缺省的列分隔符字符串是逗号；但是它可以是任何长度在 255 字节之内的字符串（如 `... DELIMITED BY '###' ...`）。指定字符串作为分隔符时应加上引号。如果您要指定以制表符分隔的值，可使用制表符 (9) 的十六进制转义序列，`... DELIMITED BY '\x09' ...`。

- **ENCODING 子句** 此子句用于指定要装载到数据库中的数据所使用的字符编码。ENCODING 子句只能与 TEXT 格式配合使用。

  如果在装载操作期间发生转换错误，将根据 on_charset_conversion_failure 选项的设置进行报告。请参见“on_charset_conversion_failure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  有关如何获得 SQL Anywhere 支持的编码列表的详细信息，请参见“支持的字符集”一节 《SQL Anywhere 服务器 - 数据库管理》。

  指定 BYTE ORDER 子句，使数据中包含字节顺序标记。

- **ENCRYPTED 子句** 此子句用于指定加密设置。装载加密数据时，应指定 ENCRYPTED KEY，并在其后附加用于加密输入文件中数据的密钥。

  如果指定 NOT ENCRYPTED，则表示输入文件中的数据是未加密的。如果数据是加密的，但不希望数据库服务器将其解密，这时也可以指定 NOT ENCRYPTED。这种情况下，数据在数据库中仍保持加密状态。但是，如果同时对文件进行了加密和压缩，则 NOT ENCRYPTED 只有在同时使用了 NOT COMPRESSED 的情况下才可以使用。

- **ESCAPE CHARACTER 子句** 此子句用于指定数据中使用的转义字符。以十六进制代码和符号存储的字符的缺省转义字符是反斜线 (\)，例如 \x0A 是换行符。使用 ESCAPE CHARACTER 子句可以对此进行更改。例如，要将感叹号用作转义字符，应输入：

  ```
  ESCAPE CHARACTER '!'
  ```

  只有单个的单字节字符才可用作转义字符。

- **ESCAPES 子句** 此子句用于控制是否识别转义字符。在 ESCAPES 设置为 ON（缺省值）的情况下，数据库服务器会识别反斜线字符后面的字符并将其解释为特殊字符。换行符可以作为组合 \n 包含在数据中，其它字符可以作为十六进制 ASCII 代码（如使用 \x09 代替制表符）包含在数据中。两个连续的反斜线字符 (\\ ) 被解释为单个反斜线。任何除 n、x、X 或 \ 以外的字符及前面的反斜线都被解释为两个单独的字符。例如，\q 插入反斜线和字母 q。

- **FORMAT 子句** 此子句用于指定要从中装载数据的数据源的格式。如果选择 TEXT，则假定输入行为字符（由 ENCODING 选项所定义），每个输入行占一行，值由列分隔符字符串分隔。选择 BCP 会允许导入 Adaptive Server Enterprise 生成的包含 BLOB 的 BCP 输出文件。

- **HEXADECIMAL 子句** 此子句用于指定是否将二进制值读取为十六进制值。缺省情况下，HEXADECIMAL 为 ON。当 HEXADECIMAL 为 ON 时，二进制列值显示为 **0x***nnnnnn...*，其中 0x 是零后面跟一个 x，每个 *n* 都是一个十六进制数字。处理多字节字符集时，使用 HEXADECIMAL ON 很重要。

  HEXADECIMAL 子句只能与 FORMAT TEXT 子句配合使用。

- **ORDER 子句** 此子句用于指定装载时数据的排序顺序。ORDER 的缺省值为 ON。如果 ORDER 为 ON，并且声明了一个聚簇索引，则 LOAD TABLE 根据聚簇索引排序输入数据，并以相同的顺序插入行。如果要装载的数据已经排序，则应将 ORDER 设置为 OFF。请参见“使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **PCTFREE 子句** 此子句用于指定要为每个表页保留的可用空间百分比。此设置将替换表的任何永久设置，但只限于装载过程中以及装载的数据。值 *percent-free-space* 是一个介于 0 至 100 之间的整数。值 0 指定每页上不留任何可用空间—每页均会填满。如果值很高，会使每行单独插入到页中。有关 PCTFREE 的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。

- **QUOTE 子句** QUOTE 子句仅适用于 TEXT 数据；*string* 放置于字符串值的两边。缺省值为单引号（撇号）。

- **QUOTES 子句** 此子句用于指定字符串是否用引号括起来。当 QUOTES 设置为 ON（缺省值）时，LOAD TABLE 语句要求将字符串用引号字符括起来。引号字符是撇号（单引号）或引号（双引号）。字符串中出现的第一个这样的字符将被视为该字符串的引号字符。字符串必须以匹配的引号结束。

  当 QUOTES 设置为 ON 时，列分隔符字符串可以包含在列值中。另外，假定引号字符不属于该值的一部分。因此，尽管地址中出现了逗号，以下行也会被视为两个值（而不是三个值）。此外，将地址引起来的引号不会插入到数据库中。

  ```
  '123 High Street, Anytown',(715)398-2354
  ```

  当 QUOTES 设置为 ON 时，要在值中包括引号字符，必须使用两个引号。以下这个行在第三列中包括一个单引号字符值：

  ```
  '123 High Street, Anytown','(715)398-2354',''''
  ```

- **ROW DELIMITED BY 子句** 此子句用于指定表示输入记录结束的字符串。缺省分隔符字符串是换行符 (\n)；但是它可以是任何长度在 255 字节之内的字符串（如 `ROW DELIMITED BY '###'`）。相同的格式要求同样适用于其它 SQL 字符串。如果您要指定以制表符分隔的值，可使用制表符 (9) 的十六进制转义序列，`ROW DELIMITED BY '\x09'`。如果分隔符字符串包含 \n，则它与 \r\n 或 \n 匹配。

- **SKIP 子句** 此子句用于指定是否忽略文件开始位置的行。*integer* 参数指定要跳过的行数。例如，可以用此子句跳过包含列标题的行。如果行分隔符不是缺省值（换行符），则当数据包含嵌入在带引号的字符串中的行分隔符时，此跳过功能可能无法正常工作。

- **STRIP 子句** 此子句用于指定在插入无引号值之前，是否应去除这些值的前导或尾随空白。STRIP 选项接受以下选项：
  - **STRIP OFF** 不去除前导空白或尾随空白。
  - **STRIP LTRIM** 去除前导空白。
  - **STRIP RTRIM** 去除尾随空白。
  - **STRIP BOTH** 去除前导空白和尾随空白。
  - **STRIP ON** 不建议使用。等效于 STRIP RTRIM。
- **WITH CHECKPOINT 子句** 此子句用于指定是否执行检查点。缺省设置为 OFF。如果将此子句设置为 ON，则会在成功完成并记录语句之后执行检查点操作。如果将此子句设置为 ON，而且在发出 CHECKPOINT 前数据库需要自动恢复，则必须具有用来装载表的数据文件才能成功完成恢复。如果指定了 WITH CHECKPOINT ON，并且此后需要恢复，则恢复在检查点之后开始，并且不必存在数据文件。

  如果数据库损坏，需要使用备份并应用当前的日志文件，则无论此子句的设置情况如何，都需要数据文件。

  小心
  如果将数据库选项 conversion_error 设置为 Off，则可能会将错误的数据装载到表中而得不到任何错误报告。如果未指定 WITH CHECKPOINT ON，并且数据库需要恢复，则当恢复过程中 conversion_error 为 On（缺省值）时，恢复可能失败。建议不要在 conversion_error 设置为 Off 且未指定 WITH CHECKPOINT ON 时装载表。

  有关 conversion_error 选项的详细信息，请参见“conversion_error 选项 [兼容性]”一节《SQL Anywhere 服务器 - 数据库管理》。
- **WITH { FILE NAME | ROW | CONTENT } LOGGING** 此子句用于控制装载操作期间记录到事务日志的信息的详细程度。记录级别如下：
  - **WITH FILE NAME LOGGING 子句** WITH FILE NAME LOGGING 子句使在事务日志中只记录 LOAD TABLE 语句。为保证恢复期间使用事务日志时的结果一致，原始装载操作时所使用的文件必须位于其原始位置，并且必须包含原始数据。此记录级别不影响性能，然而如果数据库参与镜像或同步，则不应使用此级别。此外，从表达式或客户端文件装载数据时也不能使用此级别。

    如果没有在 LOAD TABLE 语句中指定记录级别，则指定以下内容时 WITH FILE NAME LOGGING 将为缺省级别：

    - FROM *filename-expression*
    - USING FILE *filename-expression*
  - **WITH ROW LOGGING 子句** WITH ROW LOGGING 子句使每个装载的行在事务日志中都作为 INSERT 语句来记录。建议对参与同步的数据库使用此记录级别，另外在数据库镜像时也可以使用此级别。然而，当装载大量数据时，此记录类型可能会影响性能，并导致事务日志过长。

    对于要装入的表中包含非确定性值（例如计算列）或 CURRENT TIMESTAMP 缺省值的数据库，这种级别最为理想。

- **WITH CONTENT LOGGING 子句** WITH CONTENT LOGGING 子句使数据库服务器将要装载的行中的内容合并为块。随后（例如从事务日志恢复数据时）可将这些数据块重组到行中。装载大量数据时，此记录类型对性能影响非常小，并且能提供更完善的数据保护，只是会导致事务日志较长。建议对参与镜像的数据库使用此记录级别，或在不需要维护原始数据文件用于以后恢复的情况下使用。

  如果数据库参与同步，则不能使用 WITH CONTENT LOGGING 子句。

  如果没有在 LOAD TABLE 语句中指定记录级别，则指定以下内容时 WITH CONTENT LOGGING 将为缺省级别：

  - USING CLIENT FILE *client-filename-expression*
  - USING VALUE *value-expression*
  - USING COLUMN *column-expression*

- **statistics-limitation-option** 用于限制在 LOAD TABLE 执行期间生成统计信息的列。否则，将生成所有列的统计信息。只应在确认在某些列上不使用统计信息后才使用此子句。可以指定 ON ALL COLUMNS（缺省值）、OFF、ON KEY COLUMNS 或为其生成统计信息的列列表。

注释

LOAD TABLE 允许执行从文件到数据库表的高效的大量插入。LOAD TABLE 比 Interactive SQL 语句 INPUT 更有效率。

LOAD TABLE 对整个表进行写锁定。对于基表和全局临时表，执行提交。对于局部临时表，不执行提交。

如果试图对构建了快速文本索引的表或快速视图所引用的表使用 LOAD TABLE，则装载会失败。对于非快速文本索引或实例化视图则不会出现这种情况；但强烈建议您先截断相关索引和实例化视图中的数据，再执行 LOAD TABLE 语句，然后刷新索引和视图。请参见“TRUNCATE 语句”一节第 727 页和“TRUNCATE TEXT INDEX 语句”一节第 728 页。

不要在创建时已指定 ON COMMIT DELETE ROWS 的临时表上使用 LOAD TABLE 语句（无论 ON COMMIT DELETE ROWS 是显式指定的，还是在缺省情况下指定的）。不过，如果已指定 ON COMMIT PRESERVE ROWS 或 NOT TRANSACTIONAL，则*可以* 使用 LOAD TABLE。

使用 FORMAT TEXT 时，不指定任何值就表示 NULL。例如，如果需要三个值且文件包含 `1,,'Fred',`，则所插入的值为 1、NULL 和 Fred。如果文件包含 `1,2,`，则会插入值 1、2 和 NULL。仅由空格组成的值也被视为 NULL 值。例如，如果文件包含 `1, ,'Fred',`，则会插入值 1、NULL 和 Fred。所有其它值均被视为非 NULL。例如 "（单引号 单引号）是一个空字符串。'NULL' 是一个包含四个字母的字符串。

如果由 LOAD TABLE 装载的列不允许使用 NULL 值，且文件值为 NULL，则会为数字列赋予值 0（零），为字符列赋予空字符串 (")。如果由 LOAD TABLE 装载的列允许使用 NULL 值，且文件值为 NULL，则列值将为 NULL（对于所有类型）。

如果表包含 a、b 和 c 三列，输入数据包含 a、b 和 c，而 LOAD 语句仅指定 a 和 b 列作为要将数据装入的列，则会向 c 列插入以下值：

- 如果将 DEFAULTS 指定为 ON 并且 c 列具有缺省值，则使用该缺省值。
- 如果没有为 c 列定义缺省值，但该列允许 NULL，则使用 NULL。

- 如果没有为 c 列定义缺省值，且该列不允许 NULL，则使用零 (0) 或空字符串 (")，或者返回错误，这要根据列的数据类型而定。

LOAD TABLE 和列统计信息 为了创建表列的直方图，LOAD TABLE 在装载数据时捕获列统计信息。优化程序会使用此直方图。有关优化程序如何使用列统计信息的详细信息，请参见“优化程序估计值和列统计信息”一节 《SQL Anywhere 服务器 - SQL 的用法》。

以下是有关装载和列统计信息的其它提示：

- LOAD TABLE 保存基表的统计信息以备将来使用。它不保存全局临时表的统计信息。
- 如果要向以前可能包含数据的空表中装载，在执行 LOAD TABLE 语句前删除列的统计信息可能会有益处。请参见“DROP STATISTICS 语句”一节第 554 页。
- 如果对列执行 LOAD TABLE 时存在列统计信息，将*不*会重新计算该列的统计信息。相反，会将新数据的统计信息插入到现有统计信息中。这意味着如果现有列的统计信息已过期，它们在新数据装载到该列后将仍处于过期状态。如果怀疑列统计信息已过期，应该考虑在执行 LOAD TABLE 语句之前或之后更新它们。请参见“更新列统计信息以提高优化程序性能”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- 只有在表具有五个或更多的行时，LOAD TABLE 才会添加统计信息。如果表至少具有五个行，则按如下方式修改直方图：

数据是否已在表中？	是否存在直方图？	采取的操作
是	是	将更改集成到现有直方图
是	否	不生成直方图
否	是	将更改集成到现有直方图
否	否	建立新的直方图

- 对于超过 90% 的上载行，LOAD TABLE 不为包含 NULL 值的列生成统计信息。

使用动态构造的文件名 通过向变量指派文件名，然后在 LOAD TABLE 语句中使用该变量名，可以使用该动态构造的文件名执行 LOAD TABLE 语句。

权限

执行 LOAD TABLE 语句所需的权限取决于数据库服务器的 -gl 选项，如下所示：

- 如果 -gl 选项设置为 ALL，则您必须是表的所有者，或者具有 DBA 权限或 ALTER 特权。
- 如果 -gl 选项设置为 DBA，则您必须具有 DBA 权限。
- 如果 -gl 选项设置为 NONE，则不允许执行 LOAD TABLE。

请参见“-gl 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

表需要有独占锁。

从客户端计算机上的文件读取时：

- 需要 READCLIENTFILE 权限。请参见“READCLIENTFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
- 需要具有对从中读取的目录的读取权限。
- 必须启用 allow_read_client_file 数据库选项。请参见“allow_read_client_file 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- 必须启用 read_client_file 受保护的功能。请参见“-sf 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

副作用

自动提交。

如果没有指定 WITH ROW LOGGING 子句，日志文件中不会记录插入。因此，系统失败时可能无法恢复已插入的行，这取决于记录的类型。此外，决不能在用作 MobiLink 客户端的数据库中或 SQL Remote 复制所涉及的数据库中使用不带 WITH ROW LOGGING 子句的 LOAD TABLE 语句，因为这些技术会通过分析日志文件复制更改。

LOAD TABLE 语句不触发与表关联的任何触发器。

操作开始时会执行检查点操作。如果指定了 WITH CHECKPOINT ON，在结束时会执行第二个检查点。

如果装载了大量数据，则更新列统计信息。

另请参见

- “导入数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “UNLOAD 语句”一节第 731 页
- “INSERT 语句”一节第 616 页
- “INPUT 语句 [Interactive SQL]”一节第 610 页
- “访问客户端计算机上的数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “导入和导出数据” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下是 LOAD TABLE 的示例。首先，创建一个表，然后使用名为 *input.txt* 的文件将数据装载到表中。

```
CREATE TABLE t( a CHAR(100), let_me_default INT DEFAULT 1, c CHAR(100) );
```

以下是名为 *input.txt* 的文件的内容：

```
ignore_me, this_is_for_column_c, this_is_for_column_a
```

以下 LOAD 语句装载名为 *input.txt* 的文件：

```
LOAD TABLE T ( filler(), c, a ) FROM 'input.txt' FORMAT TEXT DEFAULTS ON;
```

命令 `SELECT * FROM t` 生成结果集：

a	let_me_default	c
this_is_for_column_a	1	this_is_for_column_c

以下语句使用动态构造的文件名通过 EXECUTE IMMEDIATE 语句执行 LOAD TABLE 语句：

```
CREATE PROCEDURE LoadData( IN from_file LONG VARCHAR )
BEGIN
    DECLARE path LONG VARCHAR;
    SET path = 'd:\\data\\' || from_file;
    LOAD TABLE MyTable FROM path;
END;
```

以下示例将 UTF-8 编码的表数据装载到 mytable 中：

```
LOAD TABLE mytable FROM 'mytable_data_in_utf8.dat' ENCODING 'UTF-8';
```

LOCK TABLE 语句

此语句用于防止其它并发事务访问或修改表。

语法

LOCK TABLE *table-name*
[**WITH HOLD**]
IN { **SHARE** | **EXCLUSIVE** } **MODE**

参数

- **table-name** 表的名称。表必须是基表，而不是视图。因为临时表数据是当前连接的本地数据，所以锁定全局临时表或本地临时表都无效。
- **WITH HOLD 子句** 指定此子句可将表锁定，直到连接结束时为止。如果未指定此子句，则提交或回退当前事务时将释放锁定。
- **SHARE MODE 子句** 指定此子句可获取表上的共享表锁，从而阻止其它事务修改此表，但允许它们对此表进行读取访问。如果事务在表上放置了共享锁，则它可以修改表中的数据，前提是没有其它事务持有对所修改行的任意类型的锁。
- **EXCLUSIVE MODE 子句** 指定此子句可获取表上的独占表锁，从而阻止其它事务访问此表。任何其它事务都不能对该表执行查询、更新或任何其它操作。如果使用语句（例如 `LOCK TABLE...IN EXCLUSIVE MODE`）以独占方式锁定表，则缺省行为是不获取表的行锁定。通过将 subsume_row_locks 选项设置为 Off 可以禁用此行为。

注释

LOCK TABLE 语句允许直接控制表级并发，与当前的隔离级别无关。

虽然事务的隔离级别通常控制当前事务执行请求时设置的锁的种类，但 LOCK TABLE 语句允许更显式地控制表行的锁定。

不能对视图执行 LOCK TABLE 语句。但是，如果对基表执行 LOCK TABLE 语句，就会创建一个共享模式锁，它会锁定相关的视图。共享模式锁会一直存在，直到事务被提交或回退为止。

权限

若要以 SHARE 模式锁定表，需要 SELECT 权限。

若要以 EXCLUSIVE 模式锁定表，则必须是表所有者或者有 DBA 权限。

副作用

需要访问锁定表的其它事务可能被延迟或阻塞。

另请参见

- “表锁”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “SELECT 语句”一节第 689 页
- “sa_locks 系统过程”一节第 846 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句禁止其它事务在当前事务期间修改 Customers 表：

```
LOCK TABLE Customers
IN SHARE MODE;
```

LOOP 语句

此语句用于重复执行语句列表。

语法

[*statement-label* :]
[**WHILE** *search-condition*] **LOOP**
    statement-list
END LOOP [*statement-label*]

注释

WHILE 和 LOOP 语句都是控制语句，当 *search-condition* 计算为 TRUE 时，这两个语句允许重复执行 SQL 语句列表。LEAVE 语句可用于在 END LOOP 之后的第一个语句处重新开始执行。

如果指定结尾 *statement-label*，则它必须与起始 *statement-label* 匹配。

权限

无。

副作用

无。

另请参见

- “FOR 语句”一节第 577 页
- “CONTINUE 语句 [T-SQL]”一节第 413 页

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

过程中的 While 循环。

```
...
SET i = 1;
WHILE i <= 10 LOOP
   INSERT INTO Counters( number ) VALUES ( i );
   SET i = i + 1;
END LOOP;
...
```

过程中带标签的循环。

```
SET i = 1;
lbl:
LOOP
   INSERT
   INTO Counters( number )
   VALUES ( i );
   IF i >= 10 THEN
      LEAVE lbl;
   END IF;
   SET i = i + 1;
END LOOP lbl
```

MERGE 语句

此语句用于将多个表、视图和系统过程结果合并到一个表或视图中。

语法

MERGE
INTO *target-object* [*into-column-list*]
USING [**WITH AUTO NAME**] *source-object*
  ON *merge-search-condition*
merge-operation [⋯]
 [**OPTION (** *query-hint*, ... **)**]

target-object:
[*userid.*]*target-table-name* [[**AS**] *target-correlation-name*]
| [*userid.*]*target-view-name* [[**AS**] *target-correlation-name*]
| **(** *select-statement* **)** [**AS**] *target-correlation-name*

source-object :
[*userid.*]*source-table-name* [[**AS**] *source-correlation-name*] [**WITH (** *table-hints* **)**]
| [*userid.*]*source-view-name* [[**AS**] *source-correlation-name*]
| [*userid.*]*source-mat-view-name* [[**AS**] *source-correlation-name*]

| **(** *select-statement* **)** [**AS**] *source-correlation-name* [*using-column-list*]
| *procedure*

procedure :
[*owner.*]*procedure-name* **(** *procedure-syntax* **)**
  [**WITH (** *column-name data-type*, ... **)**]
  [[**AS**] *source-correlation-name*]

merge-search-condition :
search-condition
| **PRIMARY KEY**

merge-operation :
WHEN MATCHED [**AND** *search-condition*] **THEN** *match-action*
| **WHEN NOT MATCHED** [**AND** *search-condition*] **THEN** *not-match-action*

match-action :
DELETE
| **RAISERROR** [*error-number*]
| **SKIP**
| **UPDATE SET** *set-item*, ...
| **UPDATE** [**DEFAULTS** { **ON** | **OFF** }]

not-match-action :
INSERT
| **INSERT** [*insert-column-list*] **VALUES (** *value*, ... **)**
| **RAISERROR** [*error-number*]
| **SKIP**

set-item :
[*target-correlation-name.*]*column-name* **=** { *expression* | **DEFAULT** }
| [*owner-name.*]*target-table-name.column-name* **=** { *expression* | **DEFAULT** }

insert-column-list :
(*column-name*, ... **)**

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* **=** *option-value*

into-column-list :
(*column-name*, ... **)**

using-column-list :
(*column-name*, ... **)**

error-number : positive integer or variable greater than 17000

option-name : identifier

option-value : hostvar (indicator allowed), string, identifier, or number

table-hints：请参见“FROM 子句”一节第 582 页

search-condition：请参见“搜索条件”一节第 33 页

set-clause-list：请参见“SET 语句”一节第 696 页

参数

- **INTO 子句** 此子句用于为 MERGE 语句定义目标对象。*target-object* 可以是基表、常规视图或派生表的名称，但不可以是实例化视图。派生表或视图必须代表可更新的查询块。例如，如果视图或派生表的定义中包含了 UNION、INTERSECT、EXCEPT 或 GROUP BY，那么它不可用作 MERGE 语句的目标对象。

  当 *target-object* 为派生表时，可选的 *into-column-list* 能够用于为派生表的列提供替代名称。以这种方式使用时，*into-column-list* 的大小必须与派生表的列列表匹配，并且两个列表的顺序必须相同。

  当 *target-object* 为基表或视图时，*into-column-list* 可用于将表或视图列的子集指定为与 MERGE 语句的其余部分相关。

  数据库服务器使用 *into-column-list* 来解决：

  - WHEN MATCHED 子句中不带 SET 子句的 UPDATE
  - WHEN NOT MATCHED 子句中不带 VALUES 子句的 INSERT
  - ON 子句中的 PRIMARY KEY 搜索条件
  - USING 子句中的 WITH AUTO NAME 子句

  如果没有指定 *into-column-list*，则假定 *into-column-list* 包含 *target-object* 的所有列。

- **USING 子句** 此子句用于定义要合并的数据的源。*source-object* 可以是基表（包括表提示）、视图、实例化视图、派生表或过程。如果 *source-object* 是派生表，则可以指定 *using-column-list*。如果不指定 *using-column-list*，则使用 *source-object* 的所有列。

- **WITH AUTO NAME 子句** 此子句用于指定服务器在用于合并操作的 *target-object* 中自动使用列名称来匹配 *into-column-list* 中的列。以下两个示例是等效的，它们演示了当指定 WITH AUTO Name 时，*into-column-list* 中列的顺序如何更改以便与 *source-object* 中的列名相匹配：

```
... INTO T ( Name, ID, Description )
   USING WITH AUTO NAME ( SELECT Description, Name, ID FROM PRODUCTS WHERE
Description LIKE '%cap%')
... INTO T ( Description, Name, ID )
   USING ( SELECT Description, Name, ID FROM PRODUCTS WHERE Description
LIKE '%cap%' )
```

- **ON 子句** 此子句用于指定 *source-object* 中的行与 *target-object* 中的行相匹配的条件。

  有关搜索条件语法的详细信息，请参见“搜索条件”一节第 33 页。

  可以指定 ON PRIMARY KEY 以便根据 *target-object* 的主键定义来匹配 *source-object* 的行。*source-object* 不需要主键。然而，*target-object* 必须具有主键。当指定 ON PRIMARY KEY 时：

  - 如果 *target-object* 不是基表，或它没有主键，则返回错误。
  - 如果有一个或多个主键列没有包含在 *into-column-list* 中，则返回错误。
  - 只要 *into-column-list* 中的所有主键列在 *using-column-list* 中都有相应的匹配列，*into-column-list* 和 *using-column-list* 中的列数可以不同。例如，如果 *into-column-list* 为 (I1, I2, I3)，*using-*

column-list 为 (U1, U2)，同时主键列为 (I2, I3)，则由于 *target-object* 主键的 (I3) 列在 *using-column-list* 中没有与其匹配的列，因此返回错误。

- 无论主键的定义为何，*into-column-list* 中的主键列与 *using-column-list* 中的表达式都要根据 *into-column-list* 中主键列的位置来相互匹配。例如，假设将 *target-object* 上的主键定义为 (B, C)，而 *into-column-list* 为 (E, C, F, A, D, B)。指定了 ON PRIMARY KEY 时，将 *target-object* 的 B 列与 *using-column-list* 的第六个元素进行比较，因为 B 列在 *into-column-list* 的第六个位置。同样，*target-object* 的 C 列与 *using-column-list* 的第二个元素进行比较。

  ON PRIMARY KEY 是相应的 ON 条件的语法速记。例如，假定 *into-column-list* 为 (I1, I2, ..I*n*)，而相应匹配的 *using-column-list* 为 (U1, U2, ..U*m*)。同时假定 *target-object* 的主键列为 I1、I2、I3，并且所有的主键列都包含在 *into-column-list* 中。在这种情况下,*merge-search-condition* 将被定义为合取式 `"I1=U1 AND I2=U2 AND I3=U3"`。

- **WHEN MATCHED 和 WHEN NOT MATCHED 子句** WHEN MATCHED 和 WHEN NOT MATCHED 子句用于定义 *source-object* 中的行与 *target-object* 中的行匹配或不匹配时分别采取的操作。可在 THEN 关键字后指定操作。可通过指定附加的 AND 子句来控制对匹配或不匹配行的子集采取的操作。

  ON 子句确定如何将 *source-object* 中的行分为匹配的行和不匹配的行。当 ON 子句对 *target-object* 中的至少一个行返回 TRUE 时，则认为 *source-object* 中行是匹配行。当 ON 子句对 *target-object* 中的任何行都不返回 TRUE 时，则认为 *source-object* 中的行是不匹配行。可使用多个 WHEN MATCHED 和 WHEN NOT MATCHED 子句将匹配行和非匹配行的集合划分成若干无交集子集。每个子集都由 WHEN 子句处理。按 WHEN MATCHED 和 WHEN NOT MATCHED 在 MERGE 语句中出现的顺序来处理这两个子句。

  在 WHEN MATCHED 或 WHEN NOT MATCHED 子句的 AND 子句中指定的搜索条件确定是否由指定子句处理候选行。如果指定了不带 AND 子句的 WHEN MATCHED 或 WHEN NOT MATCHED 子句，则假定 AND 子句中的搜索条件为 TRUE。如果某一行满足多个子句的 AND 条件，则该行将由 MERGE 语句中首次出现的子句处理。

  当 WHEN MATCHED 子句中的任何一个子句多次处理 *target-object* 的同一个行时，将返回错误。如果 *target-object* 的行与 *source-object* 的两个不同的行匹配，那么这个行可多次属于同一个 WHEN MATCHED 子句的同一子集。

  在下面的示例中，因为 *target-object* Products 中 ID 为 300 的行与 *source-object* SalesOrderItems 的 111 个行匹配，所以返回错误。所有的匹配行都属于相应 WHEN MATCHED THEN UPDATE 子句的同一个子集。

```
MERGE INTO Products
   USING SalesOrderItems S
   ON S.ProductID = Products.ID
   WHEN MATCHED THEN UPDATE SET Products.Quantity = 20;
```

  WHEN MATCHED：对于匹配行，可以指定以下 *match-action* 操作之一：

  - **DELETE** 指定 DELETE 将删除 *target-object* 中的行。
  - **RAISERROR** 指定 RAISERROR 将终止合并操作，回退任何更改并返回错误。缺省情况下，当指定了 RAISERROR 时，数据库服务器返回 SQLSTATE 23510 和 SQLCODE -1254。也可以通过在 RAISERROR 关键字后指定 *error-number* 参数来自定义返回的 SQLCODE。自定义 SQLCODE 必须是大于 17000 的正整数，并且可以指定为数字或变量。指定自定义 SQLCODE 时，返回的数值为负数。

例如，如果指定 `WHEN MATCHED AND search-condition THEN RAISERROR 17001`，则找到符合 WHEN 子句条件的行时，合并操作失败，更改被回退，而返回的错误为 SQLSTATE 23510 和 SQLCODE -17001。请参见“使用 RAISERROR 操作”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **SKIP** 指定 SKIP 可跳过某个行，不对其进行任何操作。
- **UPDATE** 指定 UPDATE SET 可使用 *set-item* 值更新行。*set-item* 为简单赋值表达式，其中将列设置为 *expression* 的值。对*表达式*没有任何限制。也可以指定 DEFAULT，将列设置成已为其定义的缺省值。

  例如，`UPDATE SET target-column1=DEFAULT, target-column2=source-column2` 将 target-column1 设置为它的缺省值，将 target-column2 的值设置为与 *source-object* 中 source-column2 的修改行相同。

  如果没有指定 SET 子句，SET 子句会通过 *into-column-list* 和 *using-column-list* 定义。例如，如果 *into-column-list* 为 (I1, I2, .. I*n*)，*using-column-list* 为 (U1, U2, .. U*n*)，则假定 SET 子句为 `"SET I1=U1 , I2=U2 , .. In=Un"`。

WHEN NOT MATCHED：对于不匹配行，可以指定以下 *match-action* 操作之一：

- **INSERT** 指定 INSERT ...VALUES 可使用指定的值插入行。如果指定不带 VALUES 子句的 INSERT 子句，VALUES 子句会通过 *into-column-list* 和 *using-column-list* 定义。例如，如果 *into-column-list* 为 (I1, I2, .. I*n*)，*using-column-list* 为 (U1, U2, .. U*n*)，则不包含 VALUES 子句的 INSERT 相当于 `INSERT (I1, I2, .. In) VALUES (U1, U2, .. Un)`。
- **RAISERROR** 指定 RAISERROR 将终止合并操作，回退任何更改并返回错误。指定 RAISERROR 时，缺省条件下数据库服务器返回 SQLSTATE 23510 和 SQLCODE -1254。另外，在 RAISERROR 关键字之后，可通过指定 *error-number* 参数自定义返回的 SQLCODE。自定义 SQLCODE 必须是大于 17000 的正整数，并且可以指定为数字或变量。指定自定义 SQLCODE 时，返回的数值为负数。

  例如，如果指定 `WHEN NOT MATCHED AND search-condition THEN RAISERROR 17001`，则找到符合 WHEN 子句条件的行时，合并操作失败，更改被回退，而返回的错误为 SQLSTATE 23510 和 SQLCODE -17001。请参见“使用 RAISERROR 操作”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **SKIP** 指定 SKIP 可跳过某个行，不对其进行任何操作。

- **OPTION 子句** 此子句用于指定执行语句时的提示。支持以下提示：
  - MATERIALIZED VIEW OPTIMIZATION *option-value*
  - FORCE OPTIMIZATION
  - *option-name* = *option-value*

  有关这些选项的说明，请参见“SELECT 语句”一节第 689 页的 OPTIONS 子句。

注释

source-object 中的行与 *target-object* 中的行进行比较，并根据它们是否符合 ON 子句的条件来决定二者是匹配还是不匹配。如果 *target-table* 中至少存在一行使 *merge-search-condition* 的值为 true，则将 *source-object* 中的行视为匹配。随后将根据 AND 子句指定的搜索条件，并按照 WHEN

MATCHED 和 WHEN NOT MATCHED 子句中为它们定义的操作将匹配行和不匹配行分组。通过匹配和不匹配操作对行进行分组的过程称作**分支**，每一组称为一个**分支**。

分支完成后，数据库开始执行为分支行定义的操作。分支按其出现的顺序进行处理，该顺序与 WHEN 子句在语句中出现的顺序相一致。如果分支期间 *source-object* 中的多个行为 *target-object* 中的同一行定义了操作，则合并操作失败并返回错误。这样能够防止合并操作对 *target-object* 中的给定行执行多次操作。

处理分支时，在事务日志中将插入、更新和删除操作分别记录为 INSERT、UPDATE 和 DELETE 语句。

有关触发器如何影响合并操作的信息，请参见“使用 MERGE 语句导入数据”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

DBA 权限，或：

- 如果在 MERGE 语句中指定了 INSERT、UPDATE 或 DELETE 操作，需要具有对 *target-object* 的 INSERT、UPDATE 或 DELETE 权限。
- 需要具有对 MERGE 语句中引用的任何对象的 SELECT 权限。
- 需要具有对 MERGE 语句中引用的任何过程的 EXECUTE 权限。

副作用

任何为 *target-object* 定义的触发器都将被触发。

另请参见

- “使用 MERGE 语句导入数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “UPDATE 语句”一节第 735 页
- “INSERT 语句”一节第 616 页
- “DELETE 语句”一节第 530 页
- “SELECT 语句”一节第 689 页
- “搜索条件”一节第 33 页

标准和兼容性

- **SQL/2003** MERGE 语句是 SQL/2003 标准的特性 F312。SQL Anywhere 中的 MERGE 语句与 SQL/2003 标准中的 MERGE 语句规范相符合，并具有附加扩展。MERGE 语句的一部分附加扩展与即将发布的 SQL/2008 标准相符合。其中包括：
  - 多个 WHEN MATCHED 和 WHEN NOT MATCHED 子句
  - 后跟 [AND *search-condition*] 的 WHEN [NOT] MATCHED 子句
  - WHEN MATCHED 子句中的 DELETE
  - WHEN [NOT] MATCHED 子句中的 RAISERROR
  - WHEN [NOT] MATCHED 子句中的 SKIP
  - OPTION 子句

- PRIMARY KEY 子句
- DEFAULTS 子句
- 不带 VALUES 子句的 INSERT 子句
- WITH AUTO NAME 子句
- 不带 SET 子句的 UPDATE 子句

示例

下面示例将派生表的行合并到 Products 表中，实际上添加的新 T 恤衫与现有 T 恤衫有相同属性，只是颜色、大小和产品标识不同。在此示例中，如果标识号为 304 的产品在 Products 表中已存在，则不插入行：

```
MERGE INTO Products ( ID, Name, Description, Size, Color, Quantity,
UnitPrice, Photo )
   USING WITH AUTO NAME (
      SELECT 304 AS ID,
             'Purple' AS Color,
             100 AS Quantity,
             Name,
             Description,
             Size,
             UnitPrice,
             Photo
          FROM Products WHERE Products.ID = 300 ) AS DT
   ON PRIMARY KEY
   WHEN NOT MATCHED THEN INSERT;
```

以下示例与前一个示例相同，只是没有使用语法速记：

```
MERGE INTO Products ( ID, Name, Description, Size, Color, Quantity,
UnitPrice, Photo )
   USING (
      SELECT 304 AS ID,
             'Purple' AS Color,
             100 AS Quantity,
             Name,
             Description,
             Size,
             UnitPrice,
             Photo
          FROM Products WHERE Products.ID = 300 )
          AS DT ( ID, Name, Description, Size, Color, Quantity, UnitPrice,
Photo )
   ON ( Products.ID = DT.ID )
   WHEN NOT MATCHED
      THEN INSERT ( ID, Name, Description, Size, Color, Quantity, UnitPrice,
Photo )
     VALUES ( DT.ID, DT.Name, DT.Description, DT.Size, DT.Color, DT.Quantity,
DT.UnitPrice, DT.Photo );
```

有关 MERGE 语句的详细示例，请参见“使用 MERGE 语句导入数据”一节 《SQL Anywhere 服务器 - SQL 的用法》。

MESSAGE 语句

此语句用于显示消息。

语法

MESSAGE *expression*, ...
[TYPE { INFO | ACTION | WARNING | STATUS }]
[TO { CONSOLE
| CLIENT [FOR { CONNECTION *conn-id* **[IMMEDIATE] | ALL }]**
| [EVENT | SYSTEM] LOG }
[DEBUG ONLY]]

conn-id : integer

参数

- **TYPE 子句** 此子句指定消息类型。可接受的值为 INFO、ACTION、WARNING 和 STATUS。客户端应用程序必须决定如何处理消息。例如，Interactive SQL 在以下位置显示消息：
  - **INFO** [消息] 选项卡。INFO 为缺省类型。
  - **ACTION** 含 [确定] 按钮的窗口。
  - **WARNING** 含 [确定] 按钮的窗口。
  - **STATUS** [消息] 选项卡。
- **TO 子句** 该子句指定消息的目标：
  - **CONSOLE** 将消息发送到数据库服务器消息窗口，并在数据库服务器消息日志文件已指定的情况下将消息发送到该文件。CONSOLE 为缺省值。
  - **CLIENT** 将消息发送到客户端应用程序。您的应用程序必须决定如何处理消息，您可以将 TYPE 用作信息，并根据它作出该决定。
  - **LOG** 将消息发送到 -o 选项指定的服务器日志文件中。如果指定了 EVENT 或 SYSTEM，则还会将消息分别写入数据库服务器消息窗口、位于事件源 SQLANY 11.0 Admin 下面的 Windows 事件日志和位于名称 SQLANY 11.0 Admin（服务器名）下面的 Unix SysLog 中。服务器日志中的消息将按如下方式进行标识：
    - **i** INFO 或 STATUS 类型的消息。
    - **w** WARNING 类型的消息。
    - **e** ACTION 类型的消息。
- **FOR 子句** 对于消息 TO CLIENT，此子句指定哪个连接接收关于消息的通知。缺省情况下，连接将在下次执行 SQL 语句或 WAITFOR DELAY 语句时接收消息。
  - **CONNECTION conn-id** 指定接收者的连接 ID。如果指定 IMMEDIATE，则无论何时执行 SQL 语句，连接都会在几秒钟内接收消息。
  - **ALL** 指定所有打开的连接接收消息。

- **DEBUG ONLY** 此子句允许您控制是否通过更改 debug_messages 选项的设置，来启用或禁用添加到存储过程和触发器中的调试消息。如果指定了 DEBUG ONLY，则只有在 debug_messages 选项设置为 On 时，才会执行 MESSAGE 语句。

  注意
  将 debug_messages 选项设置为 Off 时，DEBUG ONLY 消息并不占用大量资源，因此在生产系统中，通常可以将这些语句保留在存储过程中。不过，在可能会频繁执行这些语句的地方应慎用它们；否则，它们可能会导致性能略微降低。

注释

MESSAGE 语句显示消息，该消息可以是任何表达式。子句可以指定消息类型和显示消息的位置。

发出 MESSAGE ...TO CLIENT 语句的过程必须与连接相关联。

例如，下面的示例中不显示窗口，因为其事件发生在连接之外。

```
CREATE EVENT CheckIdleTime
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
   MESSAGE 'Idle engine' TYPE WARNING TO CLIENT;
END;
```

但是，在以下示例中，会将消息写入到数据库服务器消息窗口中。

```
CREATE EVENT CheckIdleTime
TYPE ServerIdle
WHERE event_condition( 'IdleTime' ) > 100
HANDLER
BEGIN
   MESSAGE 'Idle engine' TYPE WARNING TO CONSOLE;
END;
```

有效的表达式可以包括带引号的字符串或者其它常量、变量或函数。

FOR 子句可用于将数据库服务器上所检测到的事件通知其它应用程序，而不需要该应用程序显式地检查事件。使用 FOR 子句后，接收者在下次执行 SQL 语句时会接收到消息。如果接收者当前正在执行 SQL 语句，则将在该语句完成后接收到消息。如果正在执行的语句是存储过程调用，则将在调用完成前收到消息。

如果应用程序需要在消息发出不久和连接不执行 SQL 语句时得到通知，则应使用 IMMEDIATE 子句实现客户端通知，而不使用多个并发的 WAITFOR DELAY 语句。

通常情况下，使用 IMMEDIATE 子句发送的消息将在 5 秒内传送出去，即使在目标连接未发出数据库服务器请求时也是如此。如果客户端连接每秒发出多次请求，或收到庞大的 BLOB 数据，或者客户端消息回调的执行时间超过 1 秒，均可能会造成消息传送延迟。此外，在两秒内将多个 IMMEDIATE 消息发送给同一连接也可能会延迟消息传送或生成错误消息。如果客户端连接已断开，则可能无法传送成功的 MESSAGE ...IMMEDIATE 语句。

对于未使用 IMMEDIATE 子句发送的消息，只有在客户端执行某个特定请求或执行 WAITFOR DELAY 语句时才会传送它们。因此，消息的传送时间没有限制。

IMMEDIATE 子句需要 SQL Anywhere 11.0.1 版或较新的 DBLib、ODBC 或 iAnywhere JDBC 客户端库。非线程 Unix 客户端库不支持 IMMEDIATE 子句。当消息被发送到不支持 IMMEDIATE 子句的目标连接时，会生成错误消息。

MESSAGE ...TO CLIENT 表达式可以截断为 2048 个字节。对于使用 IMMEDIATE 子句发送的消息，消息表达式可以被截断为小于连接的包大小或 2048 个字节。

嵌入式 SQL 和 ODBC 客户端通过消息回调函数接收消息。在每种情况下，都必须注册这些函数。在嵌入式 SQL 中，使用 DB_CALLBACK_MESSAGE 参数向 db_register_a_callback 注册消息回调。在 ODBC 中，使用 ASA_REGISTER_MESSAGE_CALLBACK 参数向 SQLSetConnectAttr 注册消息回调。

权限

执行包含 FOR 子句、TO EVENT LOG 子句或 TO SYSTEM LOG 子句的 MESSAGE 语句需要具有 DBA 权限。

副作用

无。

另请参见

- “CREATE PROCEDURE 语句（Web 服务）”一节第 471 页
- “debug_messages 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “db_register_a_callback 函数”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下过程在数据库服务器消息窗口中显示消息：

```
CREATE PROCEDURE message_text()
BEGIN
MESSAGE 'The current date and time: ', Now();
END;
```

以下语句在数据库服务器消息窗口中显示后面跟有当前日期和时间的 `The current date and time` 字符串。

```
CALL message_text();
```

OPEN 语句 [ESQL] [SP]

此语句用于打开以前声明的游标以访问数据库中的信息。

语法

OPEN *cursor-name*
[**USING** { **DESCRIPTOR** *sqlda-name* | *hostvar*, ... }]
[**WITH HOLD**]

[**ISOLATION LEVEL** *n*]
[**BLOCK** *n*]

cursor-name : *identifier* or *hostvar*

sqlda-name : *identifier*

参数

- **USING DESCRIPTOR 子句** USING DESCRIPTOR 子句仅用于嵌入式 SQL。它指定要绑定到已声明游标的 SELECT 语句中的占位符绑定变量的主机变量。
- **WITH HOLD 子句** 缺省情况下，在当前事务（COMMIT 或 ROLLBACK）结束时会自动关闭所有游标。可选的 WITH HOLD 子句使游标对后面的事务保持打开。游标一直保持打开，直到当前连接结束或执行了显式 CLOSE 语句。连接终止时会自动关闭游标。
- **ISOLATION LEVEL 子句** ISOLATION LEVEL 子句允许以不同于 isolation_level 选项的当前设置的隔离级别打开此游标。此游标的所有操作都以指定的隔离级别执行，与该选项的设置无关。如果未指定此子句，则打开游标后，在游标打开的整段时间内，游标的隔离级别是 isolation_level 选项的值。请参见“锁定的工作方式”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  支持以下值：

  - 0
  - 1
  - 2
  - 3
  - snapshot
  - statement snapshot
  - readonly statement snapshot

  游标位于第一行之前（请参见“在嵌入式 SQL 中使用游标”一节 《SQL Anywhere 服务器 - 编程》或“在过程和触发器中使用游标”一节 《SQL Anywhere 服务器 - SQL 的用法》）。
- **BLOCK 子句** 此子句仅用于嵌入式 SQL。客户端应用程序一次可读取多行。这类读取称为块读取、预读或多行读取。BLOCK 子句可以减少预读的行数。在 OPEN 上指定 BLOCK 子句与在每个 FETCH 上指定 BLOCK 子句相同。请参见“FETCH 语句 [ESQL] [SP]”一节第 574 页。

注释

OPEN 语句打开指定的游标。该游标必须是以前声明的。

当游标位于 CALL 语句上时，OPEN 导致过程一直执行，直到遇到第一个结果集（不含 INTO 子句的 SELECT 语句）为止。如果该过程结束时未找到任何结果集，则会设置 SQLSTATE_PROCEDURE_COMPLETE 警告。

嵌入式 SQL 用法：在成功执行 OPEN 语句之后，SQLCA (SQLIOESTIMATE) 的 *sqlerrd*[*3*] 字段将被填入读取查询的所有行所需的输入/输出操作数的估计值。此外，SQLCA (SQLCOUNT) 的 *sqlerrd*[*2*] 字段将填入游标中的实际行数（大于或等于 0 的值）或其估计值（绝对值为估计值的负数）。如果数据库服务器不统计该值即可计算出行数，则该值就是实际行数。也可以将数据库配置为始终返回实际的行数（请参见“row_counts 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》），但是这样做的开销很大。如果由标识符或字符串指定 *cursor-name*，则在 C 程序中，

相应的 DECLARE CURSOR 必须出现在 OPEN 之前；如果由主机变量指定 *cursor-name*，则必须在 OPEN 语句之前执行 DECLARE CURSOR 语句。

权限

必须有 SELECT 语句中所有表的 SELECT 权限，或者有 CALL 语句中的过程的 EXECUTE 权限。

副作用

无。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “RESUME 语句”一节第 677 页
- “PREPARE 语句 [ESQL]”一节第 657 页
- “FETCH 语句 [ESQL] [SP]”一节第 574 页
- “RESUME 语句”一节第 677 页
- “CLOSE 语句 [ESQL] [SP]”一节第 405 页
- “FOR 语句”一节第 577 页

标准和兼容性

- **SQL/2003** 嵌入式 SQL 使用是核心特性。过程使用是持久存储模块特性。

示例

以下示例说明如何在嵌入式 SQL 中使用 OPEN。

```
EXEC SQL OPEN employee_cursor;
```

和

```
EXEC SQL PREPARE emp_stat FROM
'SELECT empnum, empname FROM Employees WHERE name like ?';
EXEC SQL DECLARE employee_cursor CURSOR FOR emp_stat;
EXEC SQL OPEN employee_cursor USING :pattern;
```

下面的示例来自过程或触发器。

```
BEGIN
   DECLARE cur_employee CURSOR FOR
   SELECT Surname
   FROM Employees;
   DECLARE name CHAR(40);
   OPEN cur_employee;
   LP: LOOP
         FETCH NEXT cur_employee INTO name;
         IF SQLCODE <> 0 THEN LEAVE LP END IF;
    ...
   END LOOP
   CLOSE cur_employee;
END
```

OUTPUT 语句 [Interactive SQL]

此语句用于将当前的查询结果输出到文件。

语法 1 - 输出到文件

OUTPUT TO *filename*
[**APPEND**]
[**BYTE ORDER MARK** { **ON** | **OFF** }
[**COLUMN WIDTHS (** *integer*, ... **)**]
[**DELIMITED BY** *string*]
[**ENCODING** *encoding*]
[**ESCAPE CHARACTER** *character*]
[**ESCAPES** { **ON** | **OFF** }
[**FORMAT** *output-format*]
[**HEXADECIMAL** { **ON** | **OFF** | **ASIS** }]
[**QUOTE** *string* [**ALL**]]
[**VERBOSE**]

output-format :
TEXT
| **FIXED**
| **HTML**
| **SQL**
| **XML**

encoding : *string* or *identifier*

语法 2 - 输出到 ODBC 数据源

OUTPUT
USING *connection-string*
INTO *destination-table-name*
[**CREATE TABLE** { **ON** | **OFF** }]

connection-string :
{ **DSN =** *odbc_data_source*
| **DRIVER =** *odbc_driver_name* [**;** *connection_parameter* **=** *value* [**;** ...]] }

参数

- **APPEND 子句** 此可选关键字用于将查询结果追加到现有输出文件的末尾，但不覆盖文件以前的内容。如果不使用 APPEND 子句，缺省情况下 OUTPUT 语句将覆盖输出文件的内容。当输出格式为 TEXT、FIXED 或 SQL 时，APPEND 关键字有效。
- **BYTE ORDER MARK 子句** 此子句用于指定是否在 Unicode 文件的开始位置包含字节顺序标记 (BOM)。缺省情况下，此选项为 ON，即指示 Interactive SQL 在文件的开始位置写入字节顺序标记 (BOM)。如果 BYTE ORDER MARK 为 OFF，DBISQL 不写入 BOM。

  只有写入 TEXT 格式的文件时 BYTE ORDER MARK 子句才有意义。如果试图将 BYTE ORDER MARK 子句与 FORMAT（而不是 TEXT）子句一起使用，会返回错误。

  仅当读取或写入以 UTF-8 或 UTF-16（及它们的变体）编码的文件时才使用 BYTE ORDER MARK 子句。如果试图将 BYTE ORDER MARK 子句与任何其它编码一起使用，会返回错误。
- **COLUMN WIDTHS 子句** COLUMN WIDTHS 子句用于为 FIXED 格式的输出指定列宽。

- **CREATE TABLE 子句** CREATE TABLE 子句用于指定在目标表不存在时是否创建表。缺省值为 ON。
- **DELIMITED BY 子句** DELIMITED BY 子句仅用于 TEXT 输出格式。分隔符字符串位于各列之间。缺省符号为逗号。
- **ENCODING 子句** ENCODING 子句允许指定用于写入文件的编码。ENCODING 子句只能与 TEXT 格式配合使用。

  有关如何获得 SQL Anywhere 支持的编码列表的详细信息，请参见“支持的字符集”一节《SQL Anywhere 服务器 - 数据库管理》。

  使用 Interactive SQL 时，如果未指定 ENCODING 子句，则按以下顺序确定用于写入文件的编码：

  ○ 用 default_isql_encoding 选项指定的编码（如果设置此选项）

  ○ 运行 Interactive SQL 的计算机上操作系统字符集的缺省编码

  遇到不能用操作系统字符集表示的数据时，ENCODING 子句很有用。在这种情况下，如果不使用 ENCODING 子句，那些不能用缺省编码表示的字符在输出时会丢失（即发生有损耗的转换)。

  如果在 OUTPUT 语句中指定了编码，则应指定与使用 INPUT 语句插入数据时相同的 ENCODING 子句。

  有关 Interactive SQL 和编码的详细信息，请参见“default_isql_encoding 选项 [Interactive SQL]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **ESCAPE CHARACTER 子句** 以十六进制代码和符号存储的字符的缺省转义字符是反斜线 (\)。例如，\x0A 是换行符。

  使用 ESCAPE CHARACTER 子句可以对此进行更改。例如，要将感叹号用作转义字符，可按如下方式指定：

  ```
  ... ESCAPE CHARACTER '!'
  ```

  可将换行字符指定为 '\n'。其它字符可以使用十六进制 ASCII 代码（如使用 \x09 表示制表符）指定。两个连续的反斜线字符 (\\ ) 被解释为单个反斜线。任何除 n、x、X 或 \ 以外的字符及前面的反斜线都被解释为两个单独的字符。例如，\q 被解释为反斜线和字母 q。

- **ESCAPES 子句** 在 ESCAPES 设置为 ON（缺省值）的情况下，数据库服务器会识别反斜线字符后面的字符并将其解释为特殊字符。当 ESCAPES 设置为 OFF 时，会完全按照字符在源代码中出现的方式写入字符。
- **FORMAT 子句** FORMAT 子句允许指定输出的文件格式。允许的输出格式有：

  ○ **TEXT** 输出 TEXT 格式的文件，且文件中每行代表一数据行。所有的值都用逗号分隔，并且字符串括在撇号（单引号）中。可以使用 DELIMITED BY 和 QUOTE 子句更改分隔符和引号字符串。如果在 QUOTE 子句中指定 ALL，则所有的值（不仅是字符串）都将用引号引起来。TEXT 是缺省的输出类型

  还可以使用其它三种特殊序列。两个字符 \n 代表换行符；\\ 代表单个 \；而序列 \xDD 代表具有十六进制代码 DD 的字符。

  ○ **FIXED** 输出每列都有固定宽度的固定格式。使用 COLUMN WIDTHS 子句可以指定每一列的宽度。在该格式中不输出列标题。

如果省略 COLUMN WIDTHS 子句，则根据列的数据类型计算每列的宽度，该宽度足以容纳此数据类型的任何值。例外的是 LONG VARCHAR 和 LONG BINARY 数据缺省为 32 KB。

- **HTML** 输出结果是超文本标记语言格式。
- **SQL** 输出结果是 *.sql* 文件中的 Interactive SQL INPUT 语句（重新创建表中信息时需要使用此语句）。
- **XML** 输出结果是以 UTF-8 编码且包含嵌入式 DTD 的 XML 文件。二进制值以 CDATA 块进行编码，二进制数据显示为两位数十六进制字符串。

> **注意**
> INPUT 语句不接受 XML 作为文件格式。

- **HEXADECIMAL 子句** HEXADECIMAL 子句指定如何以 TEXT 格式输出二进制值。允许的值有：
  - **ON** 当设置为 ON 时，二进制值的写入形式为 Ox 前缀后跟一系列十六进制值对；例如，0xabcd。
  - **OFF** 当设置为 OFF 时，每次只写入一个字节的二进制值。每个字节前都有转义字符（如反斜线）作为前缀，后跟 x，然后是该字节的十六进制值对，例如 \xab\xcd。
  - **ASIS** 设置为 ASIS 时，值按原样写入而不发生任何转义，即使值中包含控制字符。ASIS 对于含有格式化字符（如制表符或回车符）的文本很有用。
- **QUOTE 子句** QUOTE 子句仅用于 TEXT 输出格式。引号字符串位于字符串值的两边。缺省值为单引号 (')。如果在 QUOTE 子句中指定 ALL，则引号字符串将被置于所有值的两边，而不仅仅是字符串的两边。
- **USING 子句** USING 子句将数据导出到 ODBC 数据源。可以使用 DSN 选项指定 ODBC 数据源名，或用 DRIVER 选项指定 ODBC 驱动程序名和连接参数。*Connection-parameter* 是数据库特定的连接参数的可选列表。

  Odbc-data-source 是用户名称或 ODBC 数据源名称。例如，SQL Anywhere demo 数据库的 *odbc-data-source* 为 SQL Anywhere 11 Demo。

  Odbc-driver-name 是 ODBC 驱动程序名。对于 SQL Anywhere 11 数据库，*odbc-driver-name* 是 SQL Anywhere 11；而对于 UltraLite 数据库，*odbc-driver-name* 为 UltraLite 11。
- **VERBOSE 子句** 当包括可选的 VERBOSE 关键字时，会将有关查询的错误消息、用于选择数据的 SQL 语句和数据本身写入输出文件。不包含数据的行带有两个连字符作为前缀。如果省略 VERBOSE（缺省设置），则只将数据写入文件。当输出格式为 TEXT、FIXED 或 SQL 时，VERBOSE 关键字有效。

注释

OUTPUT 语句将数据输出到文件或数据库。可在检索待输出数据的语句后面直接使用 OUTPUT 语句。如果前一个语句生成了多个结果集，则会返回错误。

可以使用可选的 FORMAT 子句指定输出格式。如果未指定 FORMAT 子句，则使用 Interactive SQL output_format 选项设置（请参见“output_format 选项 [Interactive SQL]”一节 《SQL Anywhere 服务器 - 数据库管理》）。

由于 INPUT 语句是 Interactive SQL 命令，它不能用在任何复合语句（例如 IF）或存储过程中。请参见“过程、触发器、事件和批处理中允许使用的语句”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

无。

副作用

在 Interactive SQL 中，[结果] 选项卡显示当前查询的结果。

另请参见

- “SELECT 语句”一节第 689 页
- “INPUT 语句 [Interactive SQL]”一节第 610 页
- “UNLOAD 语句”一节第 731 页
- “导入和导出数据” 《SQL Anywhere 服务器 - SQL 的用法》
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

将 Employees 表的内容放在文本文件中：

```
SELECT *
FROM Employees;
OUTPUT TO 'Employees.txt'
FORMAT TEXT;
```

将 Employees 的内容放在现有文本文件的末尾，并在此文件中包括有关查询的任何消息：

```
SELECT *
FROM Employees;
OUTPUT TO 'Employees.txt'
APPEND VERBOSE;
```

假定需要导出包含嵌入式换行符的值。换行符的数值为 10，在 SQL 语句中可以用字符串 '\x0a' 代替该值。例如，执行以下语句，并将 HEXADECIMAL 设置为 ON：

```
SELECT CAST ('line1\x0aline2' AS VARBINARY);
OUTPUT TO 'file.txt' HEXADECIMAL ON;
```

将得到一个仅有一行的文件，其中包含以下文本：

```
0x6c696e65310a6c696e6532
```

但是，如果在 HEXADEMICAL 设置为 OFF 的情况下执行同一语句，则会获得以下文本：

```
'line1\x0Aline2'
```

最后，如果将 HEXADECIMAL 设置为 ASIS，则会获得含有两行的文件：

```
'line1
line2'
```

使用 ASIS 时获得两行是因为，已导出的嵌入式换行符没有转换成两位数的十六进制形式，也没有使用任何前缀。

以下示例将数据从 Customers 表输出到新表 Customers2 中：

```
SELECT * FROM Customers;
OUTPUT USING 'dsn=SQL Anywhere 11 Demo'
INTO "Customers2";
```

以下示例使用 DRIVER 选项将 demo 数据库中的 Customers 表复制到名为 *mydatabase.db* 的虚构数据库中。

```
SELECT * FROM Customers;
OUTPUT USING "DRIVER=SQL Anywhere 11;uid=dba;pwd=sql;dbf=c:\test
\mydatabase.db"
INTO "Customers";
```

以下示例使用 DRIVER 选项将 SQL Anywhere demo 数据库中的 Customers 表复制到虚构 UltraLite 数据库 *myULDatabase.db* 中名为 Customers 的表中。

```
SELECT * FROM Customers;
OUTPUT USING "DRIVER=Ultralite 11;dbf=c:\test\myULDatabase.udb"
INTO "Customers";
```

以下示例使用 DRIVER 选项将 Customers 表复制到名为 *mydatabase* 的虚构 MySQL 数据库中。

```
SELECT * FROM Customers;
OUTPUT USING "DRIVER=MySQL ODBC 5.1
Driver;DATABASE=mydatabase;SERVER=mySQLHost;UID=me;PWD=secret"
INTO "Customers";
```


SQL 语句 (P-Z)

以下各节定义了 SQL 语句 P-Z 部分的语法信息。

另请参见

- “SQL 语句 (A-D)”一节第 343 页
- “SQL 语句 (E-O)”一节第 565 页
- “SQL 语法中的常见元素”一节第 340 页
- “语法约定”一节第 341 页
- “语句适用性指示符”一节第 342 页

PARAMETERS 语句 [Interactive SQL]

此语句用于指定 Interactive SQL 命令文件的参数。

语法

PARAMETERS *parameter1*, *parameter2*, ...

注释

PARAMETERS 语句指定命令文件的参数，以便将来在命令文件中引用它们。

引用参数的方法是：将 `{parameter1}` 放到要替换指定参数的文件中。括号和参数名之间不能有空格。

如果使用少于所需个数的参数调用命令文件，则 Interactive SQL 会提示您提供缺少参数的值。

权限

无。

副作用

无。

另请参见

- “READ 语句 [Interactive SQL]”一节第 663 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下 Interactive SQL 命令文件带有两个参数。

```
PARAMETERS department_id, file;
SELECT Surname
FROM Employees
```

```
WHERE DepartmentID = {department_id}
>#{file}.dat;
```

如果将此脚本保存在名为 *test.sql* 的文件中，则可以在 Interactive SQL 中使用以下命令运行：

```
READ test.sql [100] [data]
```

PASSTHROUGH 语句 [SQL Remote]

此语句用于启动或停止 SQL Remote 管理所采用的直通模式。第一种和第二种形式启动直通模式，而第三种形式则停止直通模式。

语法 1

PASSTHROUGH [**ONLY**] **FOR** *userid*, ...

语法 2

PASSTHROUGH [**ONLY**] **FOR SUBSCRIPTION**
TO [*owner.*]*publication-name* [**(** *constant* **)**]

语法 3

PASSTHROUGH STOP

注释

在直通模式中，任何 SQL 语句都由数据库服务器执行，并且放在事务日志中以消息形式发送给预订者。如果使用 ONLY 关键字启动直通模式，则不在服务器上执行这些语句，而仅将它们发送给接收者。当直通会话包含对存储过程的调用时，这些过程必须位于发出直通命令的服务器中，即使它们不在本地服务器上执行。直通 SQL 语句的接收者可以是用户 ID 列表（语法 1），也可以是给定发布的所有预订者。直通模式可用于将统一数据库中的更改应用到远程数据库，或者将远程数据库中的语句发送到统一数据库。

语法 2 将语句发送到已启动预订的远程数据库，它不向已创建预订但未启动预订的远程数据库发送语句。

权限

必须具有 DBA 权限。

副作用

无。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

```
PASSTHROUGH FOR rem_db ;
...
( SQL statements to be executed at the remote database )
...
PASSTHROUGH STOP ;
```


PREPARE 语句 [ESQL]

此语句用于准备将在以后执行的语句，或者用于定义游标。

语法

PREPARE *statement-name*
  FROM *statement*
[**DESCRIBE** *describe-type* **INTO** [[**SQL**] **DESCRIPTOR**] *descriptor*]
[**WITH EXECUTE**]

statement-name : *identifier* or *hostvar*

statement : *string* or *hostvar*

describe-type :
 [**ALL** | **BIND VARIABLES** | **INPUT** | **OUTPUT** | **SELECT LIST**]
 [**LONG NAMES** [[[**OWNER.**]**TABLE.**]**COLUMN**]
    | **WITH VARIABLE RESULT**]

参数

- **statement-name**　语句名可以是标识符或主机变量。但在使用多个 SQLCA 时不能使用标识符。否则，两个预准备语句可能具有相同的语句号，而这可能导致执行或打开错误的语句。此外，建议不要将标识符用于多线程应用程序的语句名，因为语句名可能由多线程同时引用。
- **DESCRIBE 子句**　如果使用了 DESCRIBE INTO DESCRIPTOR，则会在指定的描述符中对准备好的语句进行说明。说明类型可能是 DESCRIBE 语句所允许的任何说明类型。
- **WITH EXECUTE 子句**　如果使用了 WITH EXECUTE 子句，则当且仅当此语句不是 CALL 或 SELECT 语句且不含任何主机变量时才会执行。成功执行后，此语句立即被删除。如果 PREPARE 和 DESCRIBE（如果有）成功执行，但该语句却无法执行，则会设置警告 SQLCODE 111, SQLSTATE 01W08，但不会删除该语句。

  DESCRIBE INTO DESCRIPTOR 和 WITH EXECUTE 子句可以提高性能，因为它们减少了所需的客户/服务器通信。
- **WITH VARIABLE RESULT 子句**　WITH VARIABLE RESULT 子句用于描述可能有多个结果集（具有不同列数或列类型）的过程。

  如果使用 WITH VARIABLE RESULT，则数据库服务器在 DESCRIBE 子句之后将 SQLCOUNT 值设置为下列值之一：

  - **0**　结果集可能更改：应该在执行每个 OPEN 语句后再次描述过程调用。
  - **1**　结果集是固定的。不需要重新进行描述。

静态和动态
出于兼容性考虑，仍然支持准备 COMMIT、PREPARE TO COMMIT 和 ROLLBACK 语句。但是，建议用静态嵌入式 SQL 执行所有的事务管理操作，因为某些应用程序环境可能需要它。此外，其它嵌入式 SQL 系统不支持动态事务管理操作。

注释

PREPARE 语句准备 *statement* 中指定的 SQL 语句，并使准备好的语句与 *statement-name* 关联。在执行语句或者打开游标（如果是 SELECT 语句的话）时，引用上述的语句名称。*statement-name* 可以是在自动包含的 *sqlca.h* 头文件中定义的 a_sql_statement_number 类型的主机变量。如果将标识符用于 *statement-name*，则每个模块只有一个语句可使用此 *statement-name* 进行准备。

如果将主机变量用作 *statement-name*，则它必须是 SHORT INT 类型。*sqlca.h* 中含有此类型的类型定义，名为 a_sql_statement_number。SQL 预处理器识别此类型，因此可在 DECLARE 节中使用。在执行 PREPARE 语句期间，主机变量由数据库填充，不必由程序员进行初始化。

权限

无。

副作用

此前使用同一名称准备的所有语句都会丢失。

仅当使用了 WITH EXECUTE 且执行成功时，才会在使用后删除此语句。在其它情况下，应当确保每次使用后都删除语句。如果未执行此操作，则不回收与此语句关联的内存。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “DESCRIBE 语句 [ESQL]”一节第 534 页
- “OPEN 语句 [ESQL] [SP]”一节第 647 页
- “EXECUTE 语句 [ESQL]”一节第 567 页
- “DROP STATEMENT 语句 [ESQL]”一节第 554 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

下面的语句准备一个简单查询：

```
EXEC SQL PREPARE employee_statement FROM
'SELECT Surname FROM Employees';
```

PREPARE TO COMMIT 语句

此语句用于检查 COMMIT 能否成功执行。

语法

PREPARE TO COMMIT

注释

PREPARE TO COMMIT 语句测试能否成功执行 COMMIT。如果不违反数据库完整性就无法执行 COMMIT，则此语句将导致错误。

在存储过程、触发器、事件或批中不能使用 PREPARE TO COMMIT 语句。

权限

无。

副作用

无。

另请参见

- “COMMIT 语句”一节第 408 页
- “ROLLBACK 语句”一节第 684 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在以下语句序列中，由于对 Employees 表进行了外键检查，因此导致了错误。

```
EXECUTE IMMEDIATE
   "SET OPTION wait_for_commit = 'On'";
EXECUTE IMMEDIATE "DELETE FROM Employees
   WHERE EmployeeID = 160";
EXECUTE IMMEDIATE "PREPARE TO COMMIT";
```

以下语句序列在执行删除语句时（即使它导致完整性违规）不会导致错误。PREPARE TO COMMIT 语句返回错误。

```
SET OPTION wait_for_commit= 'On';
DELETE
FROM Departments
WHERE DepartmentID = 100;
PREPARE TO COMMIT;
```

PRINT 语句 [T-SQL]

此语句用于将消息返回到客户端，或者在数据库服务器的消息窗口中显示消息。

语法

PRINT *format-string* [, *arg-list*]

注释

如果从 Open Client 应用程序或 jConnect 应用程序连接，则 PRINT 语句会将消息返回到客户端窗口。如果是从嵌入式 SQL 或 ODBC 应用程序连接，则消息显示在数据库服务器消息窗口中。

对于可选参数列表中的参数，格式字符串可以包含占位符。这些占位符的形式为 *%nn!*，其中 *nn* 是介于 1 和 20 之间的整数。

权限

无。

副作用

无。

另请参见

- “MESSAGE 语句”一节第 645 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

下面的语句显示一条消息：

```
PRINT 'Display this message';
```

以下语句说明了如何在 PRINT 语句中使用占位符：

```
DECLARE @var1 INT, @var2 INT
SELECT @var1 = 3, @var2 = 5
PRINT 'Variable 1 = %1!, Variable 2 = %2!', @var1, @var2
```

PUT 语句 [ESQL]

此语句用于在指定的游标中插入行。

语法

PUT *cursor-name*
{ **USING DESCRIPTOR** *sqlda-name* | **FROM** *hostvar-list* }
[**INTO** { **DESCRIPTOR** *sqlda-name* | *hostvar-list* }]
[**ARRAY** :*row-count*]

cursor-name : *identifier* or *hostvar*

sqlda-name : *identifier*

hostvar-list : may contain indicator variables

row-count : *integer* or *hostvar*

注释

在指定的游标中插入行。列的值从第一个 SQLDA 或主机变量列表中获取，与 INSERT 语句中的列（对于 INSERT 游标）或选择列表中的列（对于 SELECT 游标）是一一对应的关系。

PUT 语句只能在满足以下条件的 INSERT 或 SELECT 语句的游标上使用：此语句的 FROM 子句引用单个表，或引用由单个基表组成的可更新视图。

如果 SQLDA 中的 sqldata 指针为空指针，则表示没有为该列指定任何值。如果有 DEFAULT VALUE 与该列关联，则使用缺省值；否则，使用 NULL 值。

第二个 SQLDA 或主机变量列表包含 PUT 语句的结果。

可选的 ARRAY 子句可用于执行宽放置，即一次插入多行，这样可以提高性能。整数值是插入的行数。SQLDA 中，每个条目必须有一个变量（行数 * 列数）。第一行放在 SQLDA 变量 0 和（每行的列数）-1 之间，依此类推。

插入到游标中
对于滚动（对值敏感）游标，如果新行与 WHERE 子句匹配且键集游标未完成填充，则显示插入的行。对于动态游标，如果插入的行与 WHERE 子句匹配，则可能显示该行。不敏感游标不能更新。

有关将 LONG VARCHAR 或 LONG BINARY 值放入数据库的信息，请参见“SET 语句”一节 第 696 页。

权限

必须有 INSERT 权限。

副作用

当将行插入到对值敏感的游标（即，由键集决定的游标）时，插入的行会出现在结果集的末尾处，即使它们和查询的 WHERE 子句不匹配，或者当 ORDER BY 子句以正常方式将它们放置在结果集的其它位置时也是如此。请参见“通过游标修改行”一节 《SQL Anywhere 服务器 - 编程》。

另请参见

- “UPDATE 语句”一节第 735 页
- “UPDATE（定位）语句 [ESQL] [SP]”一节第 740 页
- “DELETE 语句”一节第 530 页
- “DELETE（定位）语句 [ESQL] [SP]”一节第 533 页
- “INSERT 语句”一节第 616 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

以下语句说明了如何在嵌入式 SQL 中使用 PUT：

```
EXEC SQL PUT cur_employee FROM :employeeID, :surname;
```

RAISERROR 语句

此语句用于发出错误信号和向客户端发送消息。

语法

RAISERROR *error-number* [*format-string*] [, *arg-list*]

参数

- **error-number** *error-number* 是大于 17000 的五位整数。错误号存储在全局变量 @@error 中。
- **format-string** 如果未提供 *format-string* 或为空，则使用错误号在系统表中查找错误消息。Adaptive Server Enterprise 从 SYSMESSAGES 表中获取的消息范围为 17000-19999。在 SQL Anywhere 中，此表为空视图，因此这个范围内的错误消息应该提供格式字符串。错误号大于或等于 20000 的消息从 ISYSUSERMESSAGE 表中获取。

  在 SQL Anywhere 中，*format-string* 的长度最多可达 255 字节。

  SQL Anywhere 不支持 Adaptive Server Enterprise 中的 RAISERROR 语句所支持的扩展值。

  对于可选参数列表中的参数，格式字符串可以包含占位符。这些占位符的形式为 %*nn*!，其中 *nn* 是介于 1 和 20 之间的整数。

  中间 RAISERROR 状态和代码信息在过程终止后会丢失。如果在返回时伴随 RAISERROR 发生了错误，则返回错误信息，而 RAISERROR 信息将丢失。应用程序可以通过在不同的执行点检查 @@error 全局变量来查询中间 RAISERROR 状态。

注释

RAISERROR 语句允许发出用户定义的错误并向客户端发送消息。

权限

无。

副作用

无。

另请参见

- “CREATE TRIGGER 语句 [T-SQL]”一节第 516 页
- “CREATE TRIGGER 语句”一节第 511 页
- “on_tsql_error 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “continue_after_raiserror 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句引发错误 23000（该错误在用户定义的错误范围内），并向客户端发送消息。请注意，*error-number* 和 *format-string* 参数之间没有逗号。逗号后的第一项被解释为参数列表中的第一项。

```
RAISERROR 23000 'Invalid entry for this column: %1!', @val
```

下一个示例使用 RAISERROR 禁止连接。

```
CREATE PROCEDURE DBA.login_check()
BEGIN
    // Allow a maximum of 3 concurrent connections
    IF( DB_PROPERTY('ConnCount') > 3 ) THEN
         RAISERROR 28000
```



```
        'User %1! is not allowed to connect -- there are ' ||
                        'already %2! users logged on',
        Current User,
        CAST( DB_PROPERTY( 'ConnCount' ) AS INT )-1;
    ELSE
        CALL sp_login_environment;
    END IF;
END
go
GRANT EXECUTE ON DBA.login_check TO PUBLIC
go
SET OPTION PUBLIC.login_procedure='DBA.login_check'
go
```

有关禁止连接的其它方法，请参见“login_procedure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

READ 语句 [Interactive SQL]

此语句用于从文件中读取 Interactive SQL 语句。

语法

READ [**ENCODING** *encoding*] *filename* **[** *parameter* **]** ...

encoding : *identifier* or *string*

注释

READ 语句从指定的文件中读取 Interactive SQL 语句序列。此文件可以包含任何有效的 Interactive SQL 语句，其中包括其它 READ 语句。READ 语句可以嵌套到任何深度。如果文件名中不包含绝对路径，则 Interactive SQL 将搜索该文件。Interactive SQL 将首先搜索调用文件（如果存在）的目录，然后搜索在环境变量 SQLPATH 中指定的目录，接着再搜索在环境变量 PATH 中指定的目录。如果指定的文件没有文件扩展名，则 Interactive SQL 会在每个目录中搜索带扩展名 *.sql* 的同名文件。

ENCODING 子句允许您指定用于读取文件的编码。READ 语句在读取文件时不处理转义字符。它假定整个文件都使用指定的编码。

对于 Interactive SQL，如果未指定 ENCODING 子句，则按以下顺序确定用于读取文件的编码：

- 用 default_isql_encoding 选项指定的编码（如果设置此选项）
- 运行 Interactive SQL 的计算机上操作系统字符集的缺省编码

有关 Interactive SQL 和编码的详细信息，请参见“default_isql_encoding 选项 [Interactive SQL]”一节 《SQL Anywhere 服务器 - 数据库管理》。

参数可以在命令文件名之后列出。这些参数对应于在语句文件开头的 PARAMETERS 语句中指定的参数。请参见“PARAMETERS 语句 [Interactive SQL]”一节第 655 页。

参数名必须用方括号括起来。只要源文件中包含 `{parameter-name}`（其中 *parameter-name* 是相应参数的名称），Interactive SQL 就替换相应参数。

传递给命令文件的参数可以是标识符、数字、带引号的标识符或字符串。如果用引号将参数括起来，则在替换时引号也放到文本中。不是标识符、数字或字符串（包含空格或制表符）的参数都必须用方括号 ([ ]) 括起来。这样可以在命令文件中进行任意文本替换。

如果传递给命令文件的参数不够，Interactive SQL 会提示您提供缺少的参数的值。

通过 Interactive SQL 执行 *reload.sql* 文件时，必须将加密密钥指定为一个参数。如果未在 READ 语句中提供密钥，Interactive SQL 会提示您输入密钥。请参见“Interactive SQL 实用程序 (dbisql)”一节 《SQL Anywhere 服务器 - 数据库管理》。

filename 的位置根据 READ 语句的位置确定，如下所示：

- 如果直接在 Interactive SQL 中执行 READ 语句，则相对于运行 Interactive SQL 的目录解析 *filename* 的路径。
- 如果 READ 语句在外部文件中（例如，*.sql* 文件），则 Interactive SQL 会首先尝试相对于外部文件所在位置来解析 *filename* 的路径。如果未成功，则 Interactive SQL 会在相对于正运行 Interactive SQL 的目录的路径中查找 *filename*。

权限

无。

副作用

无。

另请参见

- “PARAMETERS 语句 [Interactive SQL]”一节第 655 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下是 READ 语句示例。

```
READ status.rpt '160';
READ birthday.sql [>= '1988-1-1'] [<= '1988-1-30'];
```

READTEXT 语句 [T-SQL]

此语句用于从数据库中读取文本值和图像值，从指定的偏移处开始读取指定的字节数。

语法

READTEXT *table-name.column-name*
text-pointer-offset-size
[**HOLDLOCK**]

注释

READTEXT 用于从数据库中读取图像和文本值。不能对视图执行 READTEXT 操作。

权限

表的 SELECT 权限。

副作用

无。

另请参见

- “WRITETEXT 语句 [T-SQL]”一节第 751 页
- “GET DATA 语句 [ESQL]”一节第 590 页
- “TEXTPTR 函数 [Text and image]”一节第 310 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

REFRESH MATERIALIZED VIEW 语句

通过执行查询定义来初始化或刷新实例化视图中的数据。

语法

REFRESH MATERIALIZED VIEW *view-list*
[**WITH** {
    ISOLATION LEVEL *isolation-level*
    | { **EXCLUSIVE** | **SHARE** } **MODE** }]
[**FORCE BUILD**]

view-list :
[*owner*.]*materialized-view-name* [, ...]

isolation-level :
READ UNCOMMITTED
| **READ COMMITTED**
| **SERIALIZABLE**
| **REPEATABLE READ**
| **SNAPSHOT**

参数

- **WITH 子句** WITH 子句用于指定刷新过程中在基础性基表上使用的锁定类型。锁定类型决定如何填充实例化视图以及如何影响事务的并发性。WITH 语句设置不影响实例化视图本身的锁定类型，它始终为独占锁。可指定的锁定子句为：
  - **ISOLATION LEVEL isolation-level** WITH ISOLATION LEVEL 用于更改执行刷新操作的隔离级别。语句执行结束时会恢复该连接的原始隔离级别。

    对于快速视图，*isolation-level* 只能是 SERIALIZABLE。

对于快照隔离，只支持快照级别（指定 SNAPSHOT）；不支持语句级和只读语句快照。

有关隔离级别的信息，请参见“使用事务和隔离级别”《SQL Anywhere 服务器 - SQL 的用法》和“隔离级别和一致性”一节《SQL Anywhere 服务器 - SQL 的用法》。

- **EXCLUSIVE MODE** 如果不想更改隔离级别，但希望保证更新后的数据与基础表中已提交数据保持一致，则使用 WITH EXCLUSIVE MODE。使用 WITH EXCLUSIVE MODE 时，会在所有基础基表上放置独占表锁，在刷新操作完成前，其它事务无法对基础表执行查询、更新或其它任何操作。如果无法获取独占表锁，则刷新操作失败并返回错误。请参见“表锁”一节《SQL Anywhere 服务器 - SQL 的用法》。
- **SHARE MODE** WITH SHARE MODE 用于在执行刷新操作时为其它事务赋予对基础表的读权限。指定此子句后，在执行刷新操作之前，所有基础性基表都获得了共享表锁，并持续到刷新操作完成。请参见“表锁”一节《SQL Anywhere 服务器 - SQL 的用法》。

- **FORCE BUILD 子句** 缺省情况下，执行 REFRESH MATERIALIZED VIEW 语句时，数据库服务器会检查实例化视图是否失效（即自上次刷新实例化视图后基础表已发生更改）。如果未失效，则不进行刷新。指定 FORCE BUILD 子句以强制刷新实例化视图，而无论实例化视图是否已失效。

注释

此语句用于初始化或刷新 *view-list* 中列出的实例化视图。

如果对未失效的实例化视图执行 REFRESH MATERIALIZED VIEW 语句，则除非指定了 FORCE BUILD 子句，否则不会执行刷新操作。

锁定和数据并发的缺省刷新行为如下：

- 如果视图是快速视图，则无论是否启用快照隔离，缺省刷新行为都是 WITH SHARE MODE。
- 如果视图是手动视图，且*正在使用*快照隔离，则缺省为 WITH ISOLATION LEVEL SNAPSHOT。
- 如果视图是手动视图，但*未使用*快照隔离，则缺省为 WITH SHARE MODE。

有关隔离级别和启用快照隔离的详细信息，请参见“隔离级别和一致性”一节《SQL Anywhere 服务器 - SQL 的用法》和“allow_snapshot_isolation 选项 [数据库]”一节《SQL Anywhere 服务器 - 数据库管理》。

为了 REFRESH MATERIALIZED VIEW 能够成功执行以及能够在优化中使用视图，某些选项需要具有特定的值。另外，创建每个实例化视图时都为其存储了相应的选项设置。要刷新视图，或在优化中使用视图，这些选项设置必须与当前选项相匹配。请参见“实例化视图的限制”一节《SQL Anywhere 服务器 - SQL 的用法》。

如果在执行完部分工作后刷新失败，则视图会处于未初始化状态，并且数据无法恢复到刷新开始前的状态。请检查刷新失败时出现的错误，解决导致失败的问题，然后再次执行 REFRESH MATERIALIZED VIEW 语句。

还可以使用 ALTER MATERIALIZED VIEW 语句的 IMMEDIATE REFRESH 子句，以将视图更改为在基础数据发生更改时立即刷新。请参见“ALTER MATERIALIZED VIEW 语句”一节 第 358 页。

当存在使用 WITH HOLD 子句打开的使用语句快照或事务快照的游标时，不能执行此语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

必须具有实例化视图的 INSERT 权限，以及实例化视图定义中的表的 SELECT 权限。

副作用

关闭任何引用实例化视图的打开游标。

在执行开始时执行检查点操作。

在执行开始和结束时执行自动提交。

在执行时，使用连接 blocking 选项在正在刷新的实例化视图上放置独占模式锁，并在实例化视图所引用的所有表上放置共享模式锁（不阻塞）。如果指定了 WITH 子句，可以在基础表上获得更多锁。此外，在刷新完成之前，实例化视图会一直处于未初始化状态，从而使得数据库服务器或优化程序无法使用它。

另请参见

- “使用实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE MATERIALIZED VIEW 语句”一节第 455 页
- “ALTER MATERIALIZED VIEW 语句”一节第 358 页
- “隔离级别和一致性”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “blocking 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “表锁”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “模式锁”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_refresh_materialized_views 系统过程”一节第 878 页
- “sa_materialized_view_info 系统过程”一节第 852 页
- “sa_materialized_view_can_be_immediate 系统过程”一节第 850 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

假设您将创建一个实例化视图 EmployeeConfid99，然后使用以下语句向该实例化视图填充数据：

```
CREATE MATERIALIZED VIEW EmployeeConfid99 AS
   SELECT EmployeeID, Employees.DepartmentID, SocialSecurityNumber, Salary,
ManagerID,
      Departments.DepartmentName, Departments.DepartmentHeadID
   FROM Employees, Departments
   WHERE Employees.DepartmentID=Departments.DepartmentID;
REFRESH MATERIALIZED VIEW EmployeeConfid99;
```

随后，在该视图已投入使用后，您想要使用 READ COMMITTED 隔离级别（隔离级别 1）来刷新该视图并想重建该视图。可以执行以下语句：

```
REFRESH MATERIALIZED VIEW EmployeeConfid99
   WITH ISOLATION LEVEL READ COMMITTED
   FORCE BUILD;
```

小心
处理该示例时，应删除所创建的实例化视图。否则，在试验其它示例时，将无法对其基础表 Employees 和 Departments 执行模式更改。无法变更具有已启用相关实例化视图的表的模式。请参见“删除实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

REFRESH TEXT INDEX 语句

刷新文本索引。

语法

REFRESH TEXT INDEX *text-index-name* **ON** [*owner.*]*table-name*
[**WITH** {
    ISOLATION LEVEL *isolation-level*
    | **EXCLUSIVE MODE**
    | **SHARE MODE** }]
[**FORCE** { **BUILD** | **INCREMENTAL** }]

参数

- **WITH 子句** WITH 子句用于指定刷新过程中在基础性基表上获取的锁的类型。获取的锁类型决定了填充文本索引的方式以及事务并发性受影响的方式。如果不指定 WITH 子句，则无论为连接设置何种隔离级别，缺省值均为 WITH ISOLATION LEVEL READ UNCOMMITTED。

  可以指定以下 WITH 子句选项：

  - **ISOLATION LEVEL isolation-level** WITH ISOLATION LEVEL 用于更改执行刷新操作的隔离级别。有关隔离级别的信息，请参见“使用事务和隔离级别” 《SQL Anywhere 服务器 - SQL 的用法》和“隔离级别和一致性”一节 《SQL Anywhere 服务器 - SQL 的用法》。

    语句执行结束时会恢复连接的原始隔离级别。

  - **EXCLUSIVE MODE** 如果不想更改隔离级别，但希望保证更新后的数据与基础表中已提交数据保持一致，则使用 WITH EXCLUSIVE MODE。使用 WITH EXCLUSIVE MODE 时，会在基础性基表上放置独占表锁，在刷新操作完成前，其它事务无法对基础表执行查询、更新或其它任何操作。如果无法获取表锁，则刷新操作将失败并返回错误。请参见“表锁”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  - **SHARE MODE** WITH SHARE MODE 用于在执行刷新操作时为其它事务赋予对基础表的读权限。指定此子句后，在执行刷新操作之前，基础性基表将获得共享表锁，并持续到刷新操作完成。请参见“表锁”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **FORCE 子句** 此子句用于指定刷新方法。如果未指定此子句，则数据库服务器会根据表的更改程度来决定是执行增量更新还是完整重建。请参见“文本索引刷新类型”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  - **FORCE BUILD 子句** 此子句用于强制文本索引进行完整重建。

  - **FORCE INCREMENTAL 子句** 此子句用于强制文本索引进行增量更新。

注释

只能对定义为 MANUAL REFRESH 或 AUTO REFRESH 的文本索引使用此语句。

使用 FORCE 子句时，可以检查 sa_text_index_stats 系统过程的结果，以确定完整重建 (FORCE BUILD) 和增量更新 (FORCE INCREMENTAL) 哪个最合适。请参见“sa_text_index_stats 系统过程”一节第 906 页。

不能对定义为 IMMEDIATE REFRESH 的文本索引执行 REFRESH TEXT INDEX 语句。

对于 MANUAL REFRESH 文本索引，使用 sa_text_index_stats 系统过程确定是否应刷新文本索引。以 doc_length 除 pending_length，并将所得百分比作为确定是否需要进行刷新的标准。要确定需要的重建类型，请对 deleted_length 和 doc_count 应用相同方法。

权限

必须是基础表的所有者，或者具有 DBA 权限或 REFERENCES 权限。

当存在使用 WITH HOLD 子句打开的使用语句快照或事务快照的游标时，不能执行此语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

副作用

自动提交。

另请参见

- “文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE TEXT INDEX 语句”一节第 509 页
- “ALTER TEXT INDEX 语句”一节第 383 页
- “DROP TEXT INDEX 语句”一节第 560 页
- “TRUNCATE TEXT INDEX 语句”一节第 728 页
- “sa_refresh_text_indexes 系统过程”一节第 877 页
- “sa_text_index_stats 系统过程”一节第 906 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句刷新名为 MarketingTextIndex 的文本索引，强制其重建。

```
REFRESH TEXT INDEX MarketingTextIndex ON MarketingInformation
   FORCE BUILD;
```

REFRESH TRACING LEVEL 语句

REFRESH TRACING LEVEL 语句用于在跟踪会话进行期间从 sa_diagnostic_tracing_level 表重装跟踪级别。

语法

REFRESH TRACING LEVEL

注释

此语句用于从 sa_diagnostic_tracing_level 表重装跟踪级别信息。它必须从正在被分析的数据库中调用。

第一次启动跟踪会话时，会将行从 sa_diagnostic_tracing_level 表装载到服务器内存中，以控制所跟踪信息的种类。如果希望更改正在被跟踪的数据的类型，但不希望以停止或重新启动跟踪会话的方式来实现，则可以通过在 sa_diagnostic_tracing_level 表中手工删除或插入相应行，然后执行 REFRESH TRACING LEVEL 语句来重装这些设置。

要查看当前跟踪级别，请按如下方式查询 sa_diagnostic_tracing_level 表：

```
SELECT * FROM sa_diagnostic_tracing_level WHERE enabled = 1;
```

有关 sa_diagnostic_tracing_level 系统表的详细信息，请参见“sa_diagnostic_tracing_level 表”一节第 780 页。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “ATTACH TRACING 语句”一节第 388 页
- “DETACH TRACING 语句”一节第 539 页
- “使用诊断跟踪进行高级应用程序分析”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

假定正在解决一个性能问题。您为整个数据库打开高级别跟踪，以捕获导致出现问题的查询。启动跟踪会话之后，您发现捕获系统上所有用户的所有查询使数据库明显变慢，因此您决定宁愿将跟踪限制为一个用户，并等待该用户报告问题。但是，您不希望通过停止跟踪会话来更改这些设置。

在 Sybase Central 中，通过使用 [数据库跟踪向导] 可以完成此操作，这也是建议使用的方法。但是，通过将 sa_diagnostic_tracing_level 表中 scope=DATABASE 和 enabled=1 的行替换为 scope=USER、identifier=*userid*、enabled=1 等的相应行，您也可以完成此操作。然后执行 REFRESH TRACING LEVEL 语句以使用新设置继续跟踪。

RELEASE SAVEPOINT 语句

此语句用于释放当前事务中的保存点。

语法

RELEASE SAVEPOINT [*savepoint-name*]

注释

释放保存点。*savepoint-name* 是在当前事务内的 SAVEPOINT 语句上指定的标识符。如果省略 *savepoint-name*，则释放最近的保存点。

释放保存点时并不执行任何类型的 COMMIT 操作。它只是从当前的活动保存点列表中删除保存点。

权限

当前事务中必须有相应的 SAVEPOINT。

副作用

无。

另请参见

- “BEGIN TRANSACTION 语句 [T-SQL]”一节第 398 页
- “COMMIT 语句”一节第 408 页
- “ROLLBACK 语句”一节第 684 页
- “ROLLBACK TO SAVEPOINT 语句”一节第 685 页
- “SAVEPOINT 语句”一节第 688 页
- “事务内的保存点”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

REMOTE RESET 语句 [SQL Remote]

在自定义数据库抽取过程中，此语句在单个事务中为远程用户启动所有预订。

语法

REMOTE RESET *userid*

注释

此语句在单个事务中为远程用户启动所有预订。它将 SYSREMOTEUSER 表中的 log_sent 和 confirm_sent 值设置为事务日志中的当前位置。它还将此远程用户所有预定的 SYSSUBSCRIPTION 中的创建值和启动值设置为事务日志中的当前位置。此语句不执行提交操作。在此调用后，必须执行显式提交操作。

要编写在活动数据库上安全的抽取进程，必须在与启动预订的同一事务中以隔离级别 3 抽取数据。

此语句是启动预订的替代方法。START SUBSCRIPTION 的副作用是隐式提交，这样，如果远程用户有多个预订，则不可能使用 START SUBSCRIPTION 在一个事务中全部启动它们。

权限

必须具有 DBA 权限。

副作用

此语句不执行任何自动提交。

另请参见

- “START SUBSCRIPTION 语句 [SQL Remote]”一节第 716 页
- “ISYSREMOTEUSER 系统表”一节第 763 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句重置远程用户 SamS 的预订：

```
REMOTE RESET SamS;
COMMIT;
```

REMOVE EXTERNAL OBJECT 语句

此语句用于从数据库中删除外部对象。

语法

REMOVE EXTERNAL OBJECT *object-name*

参数

- **object-name** 外部对象的名称。

注释

有关外部环境的详细信息，请参见“外部环境概述”一节 《SQL Anywhere 服务器 - 编程》。

权限

必须具有 DBA 权限。

副作用

无

另请参见

- “外部环境概述”一节 《SQL Anywhere 服务器 - 编程》
- “ALTER EXTERNAL ENVIRONMENT 语句”一节第 353 页
- “INSTALL EXTERNAL OBJECT 语句”一节第 620 页
- “START EXTERNAL ENVIRONMENT 语句”一节第 713 页
- “STOP EXTERNAL ENVIRONMENT 语句”一节第 720 页
- “SYSEXTERNENV 系统视图”一节第 947 页
- “SYSEXTERNENVOBJECT 系统视图”一节第 948 页

标准和兼容性

- **SQL/2003** 服务商扩充。

REMOVE JAVA 语句

此语句用于从数据库中删除类或 jar 文件。删除类后，它不能再用作列类型或变量类型。

要删除的类或 jar 文件必须已安装。

语法

REMOVE JAVA *classes-to-remove*

classes-to-remove :
CLASS *java-class-name*, ... | **JAR** *jar-name*, ...

参数

- **CLASS 子句** *java-class-name* 参数是要删除的一个或多个 Java 类的名称。这些类必须是当前数据库中已安装的类。
- **JAR 子句** *jar-name* 是最大长度为 255 的字符串值。

  每个 *jar-name* 都必须与当前数据库中某个保留 jar 的 *jar-name* 相等。*jar-name* 的相等性由 SQL 系统的字符串比较规则确定。

注释

从数据库中删除类或 jar 文件。

权限

- 必须具有 DBA 权限。
- Windows Mobile 上不支持。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句从当前数据库中删除名为 Demo 的 Java 类。

```
REMOVE JAVA CLASS Demo;
```

REORGANIZE TABLE 语句

当由于需要连续访问数据库而不可能完全重建数据库时，此语句对表做碎片整理。

语法

REORGANIZE TABLE [*owner*.]*table-name*
[{ **PRIMARY KEY**
| **FOREIGN KEY** *foreign-key-name*
| **INDEX** *index-name* }]

参数

根据下面其中一项的值重组表：

- **PRIMARY KEY 子句** 重组表的主键索引。
- **FOREIGN KEY 子句** 重组指定的外键。
- **INDEX 子句** 重组指定的索引。

注释

表碎片可能会降低性能。使用该语句可对一个表中的行进行碎片整理，或压缩由于 DELETE 变得稀疏的索引。它还可以减少用于存储表及其索引的总页数，并且可以减少索引树中的级别数。但是，它不会使数据库文件的总大小减少。建议使用 sa_table_fragmentation 和 sa_index_density 系统过程选择值得处理的表。

如果未指定索引或键，则重组过程通过删除和重新插入行组来整理表中的行碎片。要为每个组获取表的独占锁。处理完组后，立即释放并重新获取锁（必要时需等待），为其它连接提供访问表的机会。处理组时，检查点操作挂起；处理完组后，可能会发生检查点操作。行按主键的顺序进行处理；如果表没有主键，则产生错误。经过处理的行重新插入到表的末尾，这使得行在处理结束时按主键聚簇。注意，无论行最初的破碎程度如何，所需的工作量是相同的。

如果指定了索引或键，则处理指定的索引。在操作期间，保持表的独占锁并挂起检查点操作。其它连接对表的任何访问尝试都将阻塞或失败，具体取决于它们的 blocking 选项设置。通过在获取独占锁之前预读取索引页，可以尽可能缩短锁定期限。

由于重组操作可能会修改许多页，检查点日志可能会变得很大。这可能导致数据库文件大小的增加。但是，这只是暂时增大，因为检查点日志在关闭时会被删除，且文件此时也会被截断。

此语句不会记录到事务日志中。

当存在使用 WITH HOLD 子句打开的使用语句快照或事务快照的游标时，不能执行此语句。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

- 必须是表的所有者，或是具有 DBA 权限的用户。
- Windows Mobile 上不支持。

副作用

在开始重组之前，会执行一个检查点操作以设法得到尽可能多的空闲页。此外，当执行 REORGANIZE TABLE 语句时，大约每 100 行会执行一次隐式提交，因此重组大表时会发生多次提交。

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句重组 Employees 表的主键索引：

```
REORGANIZE TABLE Employees
PRIMARY KEY;
```

以下语句重组 Employees 表的表页：

```
REORGANIZE TABLE Employees;
```

以下语句重组 Products 表的 IX_product_name 索引：

```
REORGANIZE TABLE Products
   INDEX IX_product_name;
```

以下语句重组 Employees 表的 FK_DepartmentID_DepartmentID 外键：

```
REORGANIZE TABLE Employees
   FOREIGN KEY FK_DepartmentID_DepartmentID;
```

RESIGNAL 语句

此语句用于重新发出异常情况通知。

语法

RESIGNAL [*exception-name*]

注释

在异常处理程序中，RESIGNAL 允许在异常仍处于活动状态时退出复合语句，或停止报告其它指定的异常。异常由其它异常处理程序处理或返回给应用程序。异常处理程序在 RESIGNAL 之前的任何操作都被撤消。

权限

无。

副作用

无。

另请参见

- “SIGNAL 语句”一节第 709 页
- “BEGIN 语句”一节第 395 页
- “在过程和触发器中使用异常处理程序”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “RAISERROR 语句”一节第 661 页

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

下面这段代码向应用程序返回除 [未找到列] 外的所有异常。

```
...
DECLARE COLUMN_NOT_FOUND EXCEPTION
   FOR SQLSTATE '52003';
...
EXCEPTION
WHEN COLUMN_NOT_FOUND THEN
SET message='Column not found';
WHEN OTHERS THEN
RESIGNAL;
```

RESTORE DATABASE 语句

此语句用于从档案中恢复已备份的数据库。

语法

RESTORE DATABASE *filename*
FROM *archive-root*
[**CATALOG ONLY**
  | [**RENAME** *dbspace-name* **TO** *new-dbspace-name*] ...]
[**HISTORY** { **ON** | **OFF** }]

filename : *string* | *variable*
archive-root : *string* | *variable*
new-dbspace-name : *string* | *variable*

参数

- **CATALOG ONLY 子句** 检索有关指定档案的信息，并将其放到备份历史记录文件 (*backup.syb*) 中，但不从档案中恢复任何数据。
- **RENAME 子句** 可以为每个 dbspace 指定新位置。不能使用 RENAME 子句更改 dbspace 名称。但是，可以使用 RENAME 子句更改文件名。
- **HISTORY 子句** 可以控制是否在历史记录文件 *backup.syb* 中记录 RESTORE DATABASE 操作。

注释

除非指定了 HISTORY OFF，否则每个 RESTORE DATABASE 操作都会更新名为 *backup.syb* 的备份历史记录文件。此文件记录已在数据库服务器上执行的 BACKUP 和 RESTORE 操作。在下列情况下，您可能不希望在 *backup.syb* 文件中记录 RESTORE DATABASE 操作：

- 经常进行 RESTORE DATABASE 操作
- 没有定期存档或删除 *backup.syb* 文件的过程
- 磁盘空间非常有限

RESTORE DATABASE 替换要恢复的数据库。如果需要增量备份，请使用 BACKUP 命令的映像格式并仅保存事务日志；但是不支持到磁带的映像备份。

权限

执行此语句所需的权限是在服务器命令行上用 -gu 选项来设置的。缺省设置为要求具有 DBA 权限。请参见“-gu 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》 。

Windows Mobile 上不支持此语句。

副作用

无。

另请参见

- “预定义 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》
- “BACKUP 语句”一节第 390 页
- “备份和数据恢复” 《SQL Anywhere 服务器 - 数据库管理》
- “SALOGDIR 环境变量”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。
- **Windows Mobile** Windows Mobile 上不支持。

示例

以下示例从磁带驱动器中恢复数据库。所需的反斜线数取决于您在执行 RESTORE DATABASE 时连接的数据库。此数据库影响 escape_character 选项的设置。该选项通常设置为 On，但在 utility_db 中设置为 Off。当连接除 utility_db 之外的任何数据库时，需要额外的反斜线。

```
RESTORE DATABASE 'd:\\dbhome\\mydatabase.db'
FROM '\\\\.\\tape0';
```

RESUME 语句

此语句用于重新开始执行返回结果集的游标。

语法

RESUME *cursor-name*

cursor-name : *identifier* | *hostvar*

注释

此语句重新执行返回结果集的过程。该过程一直执行，直到遇见下一个结果集（不带 INTO 子句的 SELECT 语句）为止。如果该过程结束时未找到任何结果集，则会设置 SQLSTATE_PROCEDURE_COMPLETE 警告。使用 RESUME 重新开始 SELECT 语句的游标时，也会设置此警告。

在 Interactive SQL 中不支持 RESUME 语句。如果希望在 Interactive SQL 查看多个结果集，可以将 isql_show_multiple_result_sets 选项设置为 ON，或者选择 [工具] » [选项]，然后在 [结果] 选项卡上选择 [显示多个结果集]。

权限

游标在此前必须已打开。

副作用

无。

另请参见

- “DECLARE CURSOR 语句 [ESQL] [SP]”一节第 524 页
- “FETCH 语句 [ESQL] [SP]”一节第 574 页
- “从过程返回结果”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下是嵌入式 SQL 的示例。

```
1. EXEC SQL RESUME cur_employee;
2. EXEC SQL RESUME :cursor_var;
```

RETURN 语句

此语句用于从函数、过程或批处理中无条件退出，并可以选择提供返回值。

语法

RETURN [*expression*]

注释

RETURN 语句导致从 SQL 块中立即退出。如果提供了 *expression*，则 *expression* 的值作为函数或过程的值返回。

如果 RETURN 出现在内部 BEGIN 块中，则所终止的是外部 BEGIN 块。

RETURN 语句之后的语句不再执行。

在函数中，表达式的数据类型应当与函数的 RETURNS 数据类型相同。

在过程中，使用 RETURN 是为了与 Transact-SQL 兼容，它用于返回整型错误代码。

权限

无。

副作用

无。

另请参见

- “CREATE FUNCTION 语句（Web 服务）”一节第 445 页
- “CREATE PROCEDURE 语句（Web 服务）”一节第 471 页
- “BEGIN 语句”一节第 395 页

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

下面的函数返回三个数字的乘积：

```
CREATE FUNCTION product (
   a NUMERIC,
   b NUMERIC,
   c NUMERIC )
RETURNS NUMERIC
BEGIN
   RETURN ( a * b * c );
END;
```

计算三个数字的乘积：

```
SELECT product(2, 3, 4);
```

product(2, 3, 4)
24

下面的过程使用 RETURN 语句避免执行复杂查询（如果这样做没有意义的话）：

```
CREATE PROCEDURE customer_products
( in customer_ID integer DEFAULT NULL)
RESULT ( ID integer, quantity_ordered integer )
BEGIN
   IF customer_ID NOT IN (SELECT ID FROM Customers)
   OR customer_ID IS NULL THEN
      RETURN
   ELSE
      SELECT Products.ID,sum(
         SalesOrderItems.Quantity )
      FROM  Products,
            SalesOrderItems,
            SalesOrders
      WHERE SalesOrders.CustomerID=customer_ID
      AND SalesOrders.ID=SalesOrderItems.ID
      AND SalesOrderItems.ProductID=Products.ID
      GROUP BY Products.ID
   END IF
END;
```

REVOKE 语句

此语句用于删除用户的权限。

语法 1

REVOKE *permission*, ... **FROM** *userid*, ...

permission :
CONNECT
| **DBA**

| **BACKUP**
| **CREATE ON** *dbspace*
| **GROUP**
| **INTEGRATED LOGIN**
| **KERBEROS LOGIN**
| **MEMBERSHIP IN GROUP** *userid*, ...
| **PROFILE**
| **RESOURCE**
| **VALIDATE**

语法 2

REVOKE *table-permission*, ...
ON [*owner*.]*table-name*
FROM *userid*, ...

table-permission :
ALL [**PRIVILEGES**]
| **ALTER**
| **DELETE**
| **INSERT**
| **REFERENCES** [**(** *column-name*, ... **)**]
| **SELECT** [**(** *column-name*, ... **)**]
| **UPDATE** [**(** *column-name*, ... **)**]

语法 3

REVOKE EXECUTE
ON [*owner*.]*procedure-name*
FROM *userid*, ...

注释

REVOKE 语句删除使用 GRANT 语句授予的权限。语法 1 撤消特殊用户权限。语法 2 撤消表权限。语法 3 撤消执行过程的权限。

REVOKE CONNECT 从数据库中删除用户 ID，同时取消该用户所拥有的任何对象（表、视图、过程等）以及由该用户授予的任何权限。如果正在删除的用户拥有由其他用户拥有的视图引用的表，则您无法针对此用户执行 REVOKE CONNECT。

REVOKE GROUP 从组的所有成员自动撤消 MEMBERSHIP IN GROUP。

将某个用户添加到某一组后，该用户将继承指派给该组的所有权限。SQL Anywhere 不允许您撤消用户以组成员身份继承的权限的子集，因为您只能撤消由 GRANT 语句显式授予的权限。如果需要让不同的用户拥有不同的权限，则可创建具有适当权限的不同的组，或者为每位用户显式授予他们所需的权限。

授予或撤消一个组对于表、视图或过程的权限时，该组的所有成员都会继承这些更改。但不继承 DBA、RESOURCE 和 GROUP 权限：必须为需要这些权限的每个单独的用户 ID 分别指派这些权限。

如果授予用户 WITH GRANT OPTION 权限，之后又撤消该权限，则还会撤消该用户在具有 WITH GRANT OPTION 权限时授予其他用户的任何权限。

权限

必须是要撤消的权限的授予者，或者是有 DBA 权限的用户。

如果正撤消其他用户的连接权限或表权限，则此时该用户不能与数据库相连。无法撤消 DBO 的连接权限。

当连接到实用程序数据库时，如果执行 REVOKE CONNECT FROM DBA，则会禁用将来与实用程序数据库的连接。这意味着，除非您使用在执行 REVOKE CONNECT 之前便已存在的连接或重新启动数据库服务器，否则以后将无法连接到实用程序数据库。

副作用

自动提交。

另请参见

- “GRANT 语句”一节第 595 页

标准和兼容性

- **SQL/2003** 语法 1 是服务商扩充。语法 2 是核心特性。语法 3 是持久存储模块特性。

示例

禁止用户 Dave 更新 Employees 表。

```
REVOKE UPDATE ON Employees FROM Dave;
```

撤消用户 Jim 的资源权限。

```
REVOKE RESOURCE FROM Jim;
```

撤消名为 Administrator 的用户配置文件的集成登录映射。

```
REVOKE INTEGRATED LOGIN FROM Administrator;
```

禁止 Finance 组执行 ShowCustomers 过程。

```
REVOKE EXECUTE ON ShowCustomers FROM Finance;
```

从数据库中删除用户 ID FranW。

```
REVOKE CONNECT FROM FranW;
```

REVOKE CONSOLIDATE 语句 [SQL Remote]

此语句用于使统一数据库停止从此数据库接收 SQL Remote 消息。

语法

REVOKE CONSOLIDATE FROM *userid*

注释

必须在远程数据库中为代表统一数据库的用户 ID 授予 CONSOLIDATE 权限。REVOKE CONSOLIDATE 语句从接收当前数据库中的消息的用户列表中删除统一数据库的用户 ID。

权限

必须具有 DBA 权限。

副作用

自动提交。删除用户的所有预订。

另请参见

- “REVOKE PUBLISH 语句 [SQL Remote]”一节第 682 页
- “REVOKE REMOTE 语句 [SQL Remote]”一节第 683 页
- “REVOKE REMOTE DBA 语句 [SQL Remote]”一节第 684 页
- “GRANT CONSOLIDATE 语句 [SQL Remote]”一节第 599 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

- 以下语句撤消 condb 用户 ID 的统一状态：

```
REVOKE CONSOLIDATE FROM condb;
```

REVOKE PUBLISH 语句 [SQL Remote]

此语句用于终止将指定用户 ID 标识为 CURRENT 发布者的操作。

语法

REVOKE PUBLISH FROM *userid*

注释

在外发消息中，SQL Remote 安装中的每个数据库都由发布者用户 ID 标识。使用 CURRENT PUBLISHER 特殊常量可得到当前的发布者用户 ID。以下的查询标识当前发布者：

```
SELECT CURRENT PUBLISHER;
```

REVOKE PUBLISH 语句会终止将指定用户 ID 标识为发布者的操作。

当数据库含有活动的 SQL Remote 发布或预订时，不要执行 REVOKE PUBLISH 从数据库中撤消发布。

在数据库中发出 REVOKE PUBLISH 语句对 SQL Remote 安装有以下几个方面的影响：

- 不能在任何以 CURRENT PUBLISHER 列作为主键的一部分的表中插入数据。任何外发的消息不再用发布者用户 ID 标识，因此不会被接收者数据库接受。

如果在 SQL Remote 安装中更改了任何统一或远程数据库中的发布者用户 ID，必须确保在所有从该数据库接收消息的数据库上向新的发布者用户 ID 授予 REMOTE 权限。这通常需要删除并重新创建所有预订。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “GRANT PUBLISH 语句 [SQL Remote]”一节第 601 页
- “REVOKE REMOTE 语句 [SQL Remote]”一节第 683 页
- “REVOKE REMOTE DBA 语句 [SQL Remote]”一节第 684 页
- “REVOKE CONSOLIDATE 语句 [SQL Remote]”一节第 681 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

```
REVOKE PUBLISH FROM publisher_ID;
```

REVOKE REMOTE 语句 [SQL Remote]

此语句用于禁止用户接收来自此数据库的 SQL Remote 消息。

语法

REVOKE REMOTE FROM *userid*, ...

注释

用户 ID 必须具有 REMOTE 权限，才能接收 SQL Remote 复制安装中的消息。REVOKE REMOTE 语句从接收当前数据库中的消息的用户列表中删除用户 ID。

权限

必须具有 DBA 权限。

副作用

自动提交。删除用户的所有预订。

另请参见

- “REVOKE PUBLISH 语句 [SQL Remote]”一节第 682 页
- “GRANT REMOTE 语句 [SQL Remote]”一节第 602 页
- “REVOKE REMOTE DBA 语句 [SQL Remote]”一节第 684 页
- “REVOKE CONSOLIDATE 语句 [SQL Remote]”一节第 681 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

```
REVOKE REMOTE FROM SamS;
```

REVOKE REMOTE DBA 语句 [SQL Remote]

此语句撤消用户 ID 的 REMOTE DBA 权限。

语法 1

REVOKE REMOTE DBA
FROM *userid*, ...

注释

在 MobiLink 中，REMOTE DBA 权限是 SQL Anywhere 同步客户端 (dbmlsync) 所需的权限级别。

在 SQL Remote 中，REMOTE DBA 权限使消息代理能够完全访问数据库以使所有更改包含在消息中，同时避免与分发 DBA 用户 ID 和口令相关的安全问题。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “REVOKE PUBLISH 语句 [SQL Remote]”一节第 682 页
- “REVOKE REMOTE 语句 [SQL Remote]”一节第 683 页
- “GRANT REMOTE DBA 语句 [MobiLink] [SQL Remote]”一节第 603 页
- “REVOKE CONSOLIDATE 语句 [SQL Remote]”一节第 681 页
- “启动同步”一节 《MobiLink - 客户端管理》
- “授予 REMOTE DBA 权限”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句撤消用户 S_Beaulieu 的 REMOTE DBA 权限。

```
REVOKE REMOTE DBA FROM S_Beaulieu;
```

ROLLBACK 语句

此语句用于结束事务，并撤消自上次执行 COMMIT 或 ROLLBACK 以来所做的任何更改。

语法

ROLLBACK [**WORK**]

注释

事务是指在两个 COMMIT 或 ROLLBACK 语句之间，对一个数据库连接完成的逻辑工作单元。ROLLBACK 语句可结束当前事务，并撤消自上一个 COMMIT 或 ROLLBACK 以来对该数据库所做的所有更改。

权限

无。

副作用

关闭所有不以 WITH HOLD 方式打开的游标。

另请参见

- “COMMIT 语句”一节第 408 页
- “ROLLBACK TO SAVEPOINT 语句”一节第 685 页

标准和兼容性

- **SQL/2003** 核心特性。

ROLLBACK TO SAVEPOINT 语句

取消自 SAVEPOINT 以来所做的所有更改。

语法

ROLLBACK TO SAVEPOINT [*savepoint-name*]

注释

ROLLBACK TO SAVEPOINT 语句撤消自 SAVEPOINT 建立以来所做的所有更改。不撤消在 SAVEPOINT 之前所做的更改；它们仍然处于待执行状态。

savepoint-name 是在当前事务内的 SAVEPOINT 语句中指定的标识符。如果省略 *savepoint-name*，则使用最近的保存点。指定保存点之后的所有保存点都自动释放。

权限

当前事务中必须有相应的 SAVEPOINT。

副作用

无。

另请参见

- “BEGIN TRANSACTION 语句 [T-SQL]”一节第 398 页
- “COMMIT 语句”一节第 408 页
- “RELEASE SAVEPOINT 语句”一节第 670 页
- “ROLLBACK 语句”一节第 684 页
- “SAVEPOINT 语句”一节第 688 页
- “事务内的保存点”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL/基础特性。

ROLLBACK TRANSACTION 语句 [T-SQL]

此语句用于取消在 SAVE TRANSACTION 后所做的所有更改。

语法

ROLLBACK TRANSACTION [*savepoint-name*]

注释

ROLLBACK TRANSACTION 语句撤消使用 SAVE TRANSACTION 建立保存点后所做的所有更改。不撤消在 SAVE TRANSACTION 之前所做的更改；它们仍然处于待执行状态。

savepoint-name 是在当前事务内的 SAVE TRANSACTION 语句中指定的标识符。如果省略 *savepoint-name*，则回退所有未完成的更改。指定保存点之后的所有保存点都自动释放。

权限

当前事务中必须有相应的 SAVE TRANSACTION。

副作用

无。

另请参见

- “ROLLBACK TO SAVEPOINT 语句”一节第 685 页
- “BEGIN TRANSACTION 语句 [T-SQL]”一节第 398 页
- “COMMIT 语句”一节第 408 页,
- “SAVE TRANSACTION 语句 [T-SQL]”一节第 687 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例显示五行，值分别为 10、20，等等。ROLLBACK TRANSACTION 语句撤消 DELETE（而不是先前的 INSERT 或 UPDATE）的结果。


```
BEGIN
    SELECT row_num INTO #tmp
    FROM sa_rowgenerator( 1, 5 )
    UPDATE #tmp SET row_num=row_num*10
    SAVE TRANSACTION before_delete
    DELETE FROM #tmp WHERE row_num >= 3
    ROLLBACK TRANSACTION before_delete
    SELECT * FROM #tmp
END
```

ROLLBACK TRIGGER 语句

此语句用于撤消在触发器中所做的所有更改。

语法

ROLLBACK TRIGGER [**WITH** *raiserror-statement*]

注释

ROLLBACK TRIGGER 语句回退在触发器中完成的工作，包括导致触发器被触发的数据修改。

另外，还可以发出 RAISERROR 语句。如果发出 RAISERROR 语句，则会向应用程序返回错误。如果不发出 RAISERROR 语句，则不返回任何错误。

如果在嵌套触发器中使用 ROLLBACK TRIGGER 语句，并且未使用 RAISERROR 语句，则只撤消最里面的触发器以及导致触发器被触发的语句。

权限

无。

副作用

None

另请参见

- “CREATE TRIGGER 语句”一节第 511 页
- “ROLLBACK 语句”一节第 684 页
- “ROLLBACK TO SAVEPOINT 语句”一节第 685 页
- “RAISERROR 语句”一节第 661 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

SAVE TRANSACTION 语句 [T-SQL]

此语句用于在当前事务中建立保存点。

语法

SAVE TRANSACTION *savepoint-name*

注释

在当前事务中建立保存点。*savepoint-name* 是可以用在 ROLLBACK TRANSACTION 语句中的标识符。事务结束后，所有保存点都自动释放。请参见“事务内的保存点”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

无。

副作用

无。

另请参见

- “SAVEPOINT 语句”一节第 688 页
- “BEGIN TRANSACTION 语句 [T-SQL]”一节第 398 页
- “COMMIT 语句”一节第 408 页
- “ROLLBACK TRANSACTION 语句 [T-SQL]”一节第 686 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例显示五行，值分别为 10、20，等等。ROLLBACK TRANSACTION 语句撤消 DELETE（而不是先前的 INSERT 或 UPDATE）的结果。

```
BEGIN
    SELECT row_num INTO #tmp
    FROM sa_rowgenerator( 1, 5 )
    UPDATE #tmp SET row_num=row_num*10
    SAVE TRANSACTION before_delete
    DELETE FROM #tmp WHERE row_num >= 3
    ROLLBACK TRANSACTION before_delete
    SELECT * FROM #tmp
END
```

SAVEPOINT 语句

此语句用于在当前事务中建立保存点。

语法

SAVEPOINT [*savepoint-name*]

注释

在当前事务中建立保存点。*savepoint-name* 是可以用在 RELEASE SAVEPOINT 或 ROLLBACK TO SAVEPOINT 语句中的标识符。事务结束后，所有保存点都自动释放。请参见“事务内的保存点”一节 《SQL Anywhere 服务器 - SQL 的用法》。

执行触发器或原子复合语句时建立的保存点在原子操作结束时自动释放。

不能从保存点中修改代理表中的数据。

权限

无。

副作用

无。

另请参见

- “RELEASE SAVEPOINT 语句”一节第 670 页
- “ROLLBACK TO SAVEPOINT 语句”一节第 685 页

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL/基础特性。

SELECT 语句

此语句用于从数据库中检索信息。

语法

[**WITH** *temporary-views*]
SELECT [**ALL** | **DISTINCT**] [*row-limitation*] *select-list*
[**INTO** { *hostvar-list* | *variable-list* | *table-name* }]
[**INTO LOCAL TEMPORARY TABLE** { *table-name* }]
[**FROM** *from-expression*]
[**WHERE** *search-condition*]
[**GROUP BY** *group-by-expression*]
[**HAVING** *search-condition*]
[**WINDOW** *window-expression*]
[**ORDER BY** { *expression* | *integer* } [**ASC** | **DESC**], ...]
[**FOR** { **UPDATE** [*cursor-concurrency*] | **READ ONLY** }]
[**FOR XML** *xml-mode*]
[**OPTION(** *query-hint*, ... **)**]

temporary-views :
regular-view, ...
| **RECURSIVE** { *regular-view* | *recursive-view* }, ...

regular-view :
view-name [**(** *column-name*, ... **)**]
AS (*subquery* **)**

recursive-view :
  view-name **(** *column-name*, ... **)**
  AS (*initial-subquery* **UNION ALL** *recursive-subquery* **)**

row-limitation :
  FIRST | **TOP** *n* [**START AT** *m*]

select-list :
expression [[**AS**] *alias-name*], ...
| *
| *window-function* **OVER** { *window-name* | *window-spec* }
  [[**AS**] *alias-name*]

from-expression：请参见“FROM 子句”一节第 582 页。

group-by-expression：请参见“GROUP BY 子句”一节第 604 页。

search-condition：请参见“搜索条件”一节第 33 页。

window-name：*identifier*

window-expression：请参见“WINDOW 子句”一节第 749 页。

window-spec：请参见“WINDOW 子句”一节第 749 页。

window-function :
RANK( )
| **DENSE_RANK( )**
| **PERCENT_RANK( )**
| **CUME_DIST( )**
| **ROW_NUMBER( )**
| *aggregate-function*

cursor-concurrency :
BY { **VALUES** | **TIMESTAMP** | **LOCK** }

xml-mode :
RAW [**, ELEMENTS**]
| **AUTO** [**, ELEMENTS**]
| **EXPLICIT**

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| **FORCE NO OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

参数

- **WITH 或 WITH RECURSIVE 子句** 定义一个或多个公用表表达式（也称为临时视图），用于该语句的其余部分。这些表达式可以是非递归的，也可以是自递归的。只要指定了 RECURSIVE 关键字，递归公用表表达式就可以单独显示，或与非递归表达式混杂显示。不支持相互递归公用表表达式。

只有 SELECT 语句出现在下列位置之一时，才允许使用该子句：

- 顶级 SELECT 语句中
- VIEW 定义的顶级 SELECT 语句中
- INSERT 语句内的顶级 SELECT 语句中

递归表达式由一个初始子查询和一个递归子查询组成。初始查询隐式定义视图的模式。递归子查询必须在 FROM 子句中包含对视图的引用。每次迭代中，该引用只能引用以前迭代中添加到视图的行。该引用不能出现在外连接的空值提供方中。递归公用表表达式不得使用集合函数，而且不得包含 GROUP BY、ORDER BY 或 DISTINCT 子句。请参见“公用表表达式”《SQL Anywhere 服务器 - SQL 的用法》。

不支持将 WITH 子句用于远程表。

- **ALL 或 DISTINCT 子句** All（缺省值）返回满足 SELECT 语句的子句的所有行。如果指定 DISTINCT，则会消除重复的输出行。当指定 DISTINCT 时，很多语句的执行时间显著延长，因此只有在必要时才应使用 DISTINCT。
- ***row-limitation* 子句** 行限制子句允许您只返回满足 WHERE 子句的行的子集。TOP 和 START AT 值可以是主机变量、整型常量或整型变量。TOP 值必须大于等于 0。START AT 值必须大于 0。通常，在指定这些子句时也指定 ORDER BY 子句，这样可以用一种有意义的方式对这些行进行排序。请参见“显式限制查询返回的行数”一节《SQL Anywhere 服务器 - SQL 的用法》。
- ***select-list* 子句** *select-list* 是一个由逗号分隔的表达式列表，用于指定从数据库中检索的内容。星号 (*) 表示选择 FROM 子句中的所有表的所有列。

  select-list 中允许使用集合函数。*select-list* 中还可以使用子查询。每个子查询都必须用括号括起来。

  在整个查询中都可以使用别名来表示带别名的表达式。

  Interactive SQL 还在 SELECT 语句的每个输出列的顶部显示别名。如果表达式后面未指定可选别名，则 Interactive SQL 将显示表达式本身。
- **INTO 子句** 下面是 INTO 子句的三种用法：
  - **INTO *hostvar-list* 子句** 此子句仅用于嵌入式 SQL。它指定 SELECT 语句结果的位置。*select-list* 中的每一项都必须有一个主机变量项。*select-list* 中的项依次放入主机变量中。每个主机变量还可以有一个指示符主机变量，以便程序可以判定 *select-list* 项是否为 NULL。
  - **INTO *variable-list* 子句** 此子句仅用于过程和触发器。它指定 SELECT 语句结果的位置。*select-list* 中的每一项都必须有一个变量。*select-list* 中的项依次放入变量中。
  - **INTO *table-name* 子句** 该子句用于创建表并用数据填充表。

    对于所要创建的永久表，查询必须满足以下条件之一：

    - *select-list* 包含多个项，而 INTO 目标为单个 *table-name* 标识符。
    - *select-list* 包含一个 *，而 INTO 目标指定为 *owner.table*。

    要创建一个列的永久表，表名必须指定为 *owner.table*。

作为创建表的副作用，此语句会在执行前导致 COMMIT。执行此语句需要具有 RESOURCE 权限。新表未被授予任何权限：该语句是后接 INSERT ... SELECT 的 CREATE TABLE 的简写形式。

使用此子句创建的表没有定义主键。可以使用 ALTER TABLE 添加主键。在对表应用任何 UPDATE 或 DELETE 之前应添加主键；否则，这些操作会使受影响的行的所有列值记录在事务日志中。

- **INTO LOCAL TEMPORARY TABLE** 此子句用于创建局部临时表，并用查询结果填充该表。使用此子句时，临时表名不必以 # 开头。
- **FROM 子句** 行是从 *table-expression* 指定的表和视图中检索的。不带 FROM 子句的 SELECT 语句可用于显示不是从表中派生的表达式的值。例如，以下两个语句是等同的，都显示全局变量 @@version 的值。

```
SELECT @@version;
SELECT @@version FROM DUMMY;
```

请参见“FROM 子句”一节第 582 页。

- **WHERE 子句** 此子句指定将从 FROM 子句指定的表中选择的行。该子句可用于在多个表之间建立连接，它是 ON 短语（FROM 子句的组成部分）的替代方法。请参见“搜索条件”一节第 33 页和“FROM 子句”一节第 582 页。
- **GROUP BY 子句** 可以按列、别名或函数进行分组。查询结果对于指定列、别名或函数的每个不同的值集均包含一行。与 DISTINCT 和集合操作 UNION、INTERSECT 及 EXCEPT 一样，GROUP BY 子句处理 NULL 值的方式与处理每个域中任何其它值的方式相同。换句话说，分组属性内的多个 NULL 值将构成一个单个的组。然后可将集合函数应用于这些组以获得有意义的结果。

  当使用 GROUP BY 时，*select-list*、HAVING 子句和 ORDER BY 子句不能引用 GROUP BY 子句中未指定的任何标识符。*select-list* 和 HAVING 子句例外，它们可以包含集合函数。
- **HAVING 子句** 此子句根据组值而不是各行值来选择行。只有当该语句有 GROUP BY 子句或 *select-list* 完全由集合函数组成时，才能使用 HAVING 子句。在 HAVING 子句中引用的任何列名必须存在于 GROUP BY 子句中，或被用作 HAVING 子句中集合函数的参数。
- **WINDOW 子句** 此子句定义全部或部分用于与窗口函数一起使用的窗口，例如 AVG 和 RANK。请参见“WINDOW 子句”一节第 749 页。
- **ORDER BY 子句** 此子句将对查询的结果进行排序。ORDER BY 列表中的每一项均可标记为 ASC 以按升序排序（缺省值），或标记为 DESC 以按降序排序。如果表达式是整数 *n*，则查询结果按 *select-list* 中的第 *n* 项排序。

  确保以特定的顺序返回行的唯一方法是使用 ORDER BY。当缺少 ORDERBY 子句时，SQL Anywhere 以最有效的顺序返回行。这意味着结果集的外观可能因您上次访问行的时间和其它因素而异。

  在嵌入式 SQL 中，SELECT 语句用于从数据库中检索结果，并通过 INTO 子句将值放入主机变量。SELECT 语句必须只返回一行。对于多行查询，必须使用游标。
- **FOR UPDATE 或 FOR READ ONLY 子句** 这些子句指定是否允许通过在查询中打开的游标进行更新，以及当允许时，将使用哪种并发语义。此子句不能与 FOR XML 子句一起使用。

当指定 FOR UPDATE BY TIMESTAMP 或 FOR UPDATE BY VALUES 时，该数据库服务器通过使用由键集决定的游标来使用优化的并发。在这种情况下，可发生更新丢失。

如果 SELECT 语句中未使用 FOR 子句，则游标的可更新性取决于游标的声明（请参见“DECLARE 语句”一节第 523 页和“FOR 语句”一节第 577 页），并由 API 指定游标如何并发。在 ODBC、JDBC 和 OLE DB 中，语句的可更新性是显式和只读的，除非被应用程序替换，否则将使用只进游标。在 Open Client、嵌入式 SQL 和存储过程中，无需指定游标的可更新性，缺省值为 FOR UPDATE。

为确保语句获得意图锁，必须执行以下步骤之一：

- 在查询中指定 FOR UPDATE BY LOCK
- 在查询的 FROM 子句中指定 HOLDLOCK、WITH (HOLDLOCK)、WITH (UPDLOCK) 或 WITH (XLOCK)
- 使用指定 CONCUR_LOCK 的 API 调用打开游标
- 读取带有指示读取更新的属性的行

除了游标的可更新性之外，语句的可更新性还取决于 ansi_update_constraints 数据库选项的设置和语句的专有特性，包括语句是包含 ORDER BY、DISTINCT、GROUP BY、HAVING、UNION、集合函数、连接还是不可更新的视图。

有关游标敏感性的详细信息，请参见“SQL Anywhere 游标”一节 《SQL Anywhere 服务器 - 编程》。

有关 ODBC 并发性的详细信息，请参见“选择 ODBC 游标特性”一节 《SQL Anywhere 服务器 - 编程》中对 SQLSetStmtAttr 的讨论。

有关 ansi_update_constraints 数据库选项的详细信息，请参见“ansi_update_constraints 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关游标可更新性的详细信息，请参见“了解可更新语句”一节 《SQL Anywhere 服务器 - 编程》。

- **FOR XML 子句** 此子句指定以 XML 文档格式返回结果集。XML 的格式取决于所指定的模式。此子句不能与 FOR UPDATE 或 FOR READ ONLY 子句一起使用。

  当指定 RAW 模式时，结果集中的每一行表示为一个 XML <row> 元素，每一列表示为 <row> 元素的一个属性。

  AUTO 模式返回嵌套 XML 元素格式的查询结果。*select-list* 中引用的每一个表都会表示为 XML 中的一个元素。元素的嵌套顺序基于 *select-list* 中引用表的顺序。

  使用 EXPLICIT 模式可以控制生成的 XML 文档的格式。在命名元素和指定嵌套结构方面，使用 EXPLICIT 模式比 RAW 或 AUTO 模式提供了更大的灵活性。请参见“使用 FOR XML EXPLICIT”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  有关使用 FOR XML 子句的详细信息，请参见“使用 FOR XML 子句以 XML 格式检索查询结果”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **OPTION 子句** 此子句提供关于如何处理查询的提示。支持以下查询提示：

  - **MATERIALIZED VIEW OPTIMIZATION 子句** MATERIALIZED VIEW OPTIMIZATION 子句用于指定在处理查询时优化程序应如何利用实例化视图。指定的 *option-value* 将替换仅用

于此查询的 materialized_view_optimization 数据库选项。*option-value* 可能的值同样可用于 materialized_view_optimization 数据库选项。请参见“materialized_view_optimization 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **FORCE OPTIMIZATION 子句** 当查询说明仅包含简单的查询（在 WHERE 子句中包含相等条件且唯一地标识特定行的单块、单表查询）时，它通常会在处理过程中跳过基于开销的优化。在某些情况下，您可能需要出现基于开销的优化。例如，如果在处理查询的过程中您想要考虑实例化视图，则视图匹配必须发生。但视图匹配只有在基于开销的优化过程中发。如果想要针对某个查询发生基于开销的优化，而您的查询说明仅包括简单的查询，则可指定 FORCE OPTIMIZATION 选项以确保优化程序在查询上执行基于开销的优化。

  同样，在过程中指定 SELECT 语句中的 FORCE OPTIMIZATION 选项会强制将优化程序用于对该过程的任何调用。这种情况下，不对该语句的计划进行高速缓存。

  有关简单查询和视图匹配的详细信息，请参见“查询处理阶段”一节 《SQL Anywhere 服务器 - SQL 的用法》和“跳过查询处理阶段的资格”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **FORCE NO OPTIMIZATION 子句** 如果希望语句跳过优化程序，可指定 FORCE NO OPTIMIZATION 子句。如果语句过于复杂以至于无法以这种方式进行处理（可能是由于数据库选项的设置，也可能是由于模式或查询的特性而导致），此语句会失败并且数据库服务器将返回错误。有关可跳过优化程序的语句的详细信息，请参见“跳过查询处理阶段的资格”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- ***option-name* = *option-value*** 仅针对此语句指定优先于任何有效的公共或临时选项设置的选项设置。所支持的选项为：

  - “isolation_level 选项 [数据库] [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “max_query_tasks 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “optimization_goal 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “optimization_level 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “optimization_workload 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
  - “user_estimates 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

注释

SELECT 语句可用于：

- 从数据库中检索结果。
- 可在 Interactive SQL 中使用，以浏览数据库中的数据或将数据从数据库导出到外部文件。
- 可在过程和触发器中使用，或在嵌入式 SQL 中使用。带 INTO 子句的 SELECT 语句用于在 SELECT 语句只返回一行时从数据库中检索结果。对于多行查询，必须使用游标。
- 从一个过程返回结果集。

注意
当 SELECT 语句中使用 GROUP BY 表达式时，*select-list*、HAVING 子句和 ORDER BY 子句只能引用 GROUP BY 子句中指定的标识符。*select-list* 和 HAVING 子句例外，它们可以包含集合函数。

权限

必须有指定表和视图的 SELECT 权限。

副作用

无。

另请参见

- “表达式”一节第 16 页
- “FROM 子句”一节第 582 页
- “搜索条件”一节第 33 页
- “UNION 子句”一节第 729 页
- “EXCEPT 子句”一节第 565 页
- “INTERSECT 子句”一节第 623 页
- “连接：从多个表检索数据”《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 核心特性。SELECT 语句的复杂性意味着应该根据标准检查各子句。例如，ROLLUP 关键字是特性 T431 的一部分。

  FOR UPDATE、FOR READ ONLY 和 FOR UPDATE (*column-list*) 都是核心特性。

  FOR UPDATE BY [LOCK | TIMESTAMP | VALUES] 是一个 SQL Anywhere 服务商扩充。

示例

此示例返回 Employees 表中的雇员总数。

```
SELECT COUNT(*)
FROM Employees;
```

此示例列出了所有客户及其订单总值。

```
SELECT CompanyName,
   CAST( SUM( SalesOrderItems.Quantity *
   Products.UnitPrice ) AS INTEGER ) VALUE
FROM Customers
   JOIN SalesOrders
   JOIN SalesOrderItems
   JOIN Products
GROUP BY CompanyName
ORDER BY VALUE DESC;
```

以下语句显示了一个嵌入式 SQL SELECT 语句，其中 Employees 表中的雇员数被选择到 :size 主机变量中：

```
SELECT count(*) INTO :size
FROM Employees;
```

以下语句进行了优化，能够快速返回结果集中的第一行：

```
SELECT Name
FROM Products
GROUP BY Name
HAVING COUNT( * ) > 1
```



```
AND MAX( UnitPrice ) > 10
OPTION( optimization_goal = 'first-row' );
```

SET 语句

此语句用于向 SQL 变量赋值。

语法

SET *identifier* **=** *expression*

注释

SET 语句为变量赋新值。必须是以前用 CREATE VARIABLE 语句或 DECLARE 语句所创建的变量，或者必须是过程的 OUTPUT 参数。变量名称可以使用以下 Transact-SQL 约定：在名称前使用 @ 符号。例如：

```
SET @localvar = 42
```

在 SQL 语句中，只要是允许使用列名的位置，就可以使用变量。如果列名和变量名相同，则使用变量值。

变量是当前连接的本地对象，当您与数据库断开连接或使用 DROP VARIABLE 语句时消失。它们不受 COMMIT 或 ROLLBACK 语句的影响。

从嵌入式 SQL 程序中为 INSERT 或 UPDATE 语句创建大文本或二进制对象时需要使用变量，因为嵌入式 SQL 主机变量仅限于 32,767 个字节。

权限

无。

副作用

无。

另请参见

- “CREATE VARIABLE 语句”一节第 517 页
- “DECLARE 语句”一节第 523 页
- “DROP VARIABLE 语句”一节第 563 页
- “表达式”一节第 16 页

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

此简单示例显示了名为 'birthday' 的变量的创建，并使用 SET 来将日期设置为 CURRENT DATE。

```
CREATE VARIABLE @birthday DATE;
    SET @birthday = CURRENT DATE;
```

下面这段代码向数据库中插入一个大文本值。


```
EXEC SQL BEGIN DECLARE SECTION;
DECL_VARCHAR( 500 ) buffer;
/* Note: maximum DECL_VARCHAR size is 32765 */
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE VARIABLE hold_blob LONG VARCHAR;
EXEC SQL SET hold_blob = '';
for(;;) {
   /* read some data into buffer ... */
   size = fread( buffer, 1, 5000, fp );
   if( size <= 0 ) break;
   /* Does not work if data contains null chars */
   EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES( 1, hold_blob );
EXEC SQL DROP VARIABLE hold_blob;
```

下面这段代码向数据库中插入一个大二进制值。

```
EXEC SQL BEGIN DECLARE SECTION;
DECL_BINARY( 5000 ) buffer;
EXEC SQL END DECLARE SECTION;

EXEC SQL CREATE VARIABLE hold_blob LONG BINARY;
EXEC SQL SET hold_blob = '';
for(;;) {
   /* read some data into buffer ... */
   size = fread( &(buffer.array), 1, 5000, fp );
   if( size <= 0 ) break;
   buffer.len = size;
   /* add data to blob using concatenation */
   EXEC SQL SET hold_blob = hold_blob || :buffer;
}
EXEC SQL INSERT INTO some_table VALUES ( 1, hold_blob );
EXEC SQL DROP VARIABLE hold_blob;
```

SET 语句 [T-SQL]

此语句用于为当前连接设置与 Adaptive Server Enterprise 兼容的数据库选项。

语法

SET *option-name option-value*

注释

可用选项如下：

选项名称	选项值
ansinull	On 或 Off
ansi_permissions	On 或 Off
close_on_endtrans	On 或 Off

选项名称	选项值
datefirst	1、2、3、4、5、6 或 7 此选项的设置在获取工作日值时会影响 DATEPART 函数。 有关指定每周第一天的详细信息，请参见“first_day_of_week 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》和“DATEPART 函数 [Date and time]”一节第 169 页。
quoted_identifier	On | Off
rowcount	*integer*
self_recursion	On | Off
string_rtruncation	On | Off
textsize	*integer*
transaction isolation level	0、1、2、3、snapshot、statement snapshot 或 read only statement snapshot

在 SQL Anywhere 中，用 SET OPTION 语句设置数据库选项。不过，SQL Anywhere 还支持用 Adaptive Server Enterprise 的 SET 语句设置对兼容性有用的选项。

在 SQL Anywhere 和 Adaptive Server Enterprise 中，可以用 Transact-SQL SET 语句设置以下选项：

- **SET ansinull { On | Off }** 在 SQL Anywhere 和 Adaptive Server Enterprise 中，比较值与 NULL 的缺省行为是不同的。将 ansinull 设置为 Off 可提供与 Transact-SQL 兼容的 NULL 比较。

  SQL Anywhere 还支持以下语法：

  SET ansi_nulls { **On** | **Off** }

  有关详细信息，请参见“ansinull 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **SET ansi_permissions { On | Off }** 在 SQL Anywhere 和 Adaptive Server Enterprise 中，关于执行含有列引用的 UPDATE 或 DELETE 所需权限的缺省行为是不同的。将 ansi_permissions 设置为 Off 可提供与 Transact-SQL 兼容的 UPDATE 和 DELETE 权限。请参见“ansi_permissions 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **SET close_on_endtrans { On | Off }** 在 SQL Anywhere 和 Adaptive Server Enterprise 中，在事务结束时关闭游标的缺省行为是不同的。将 close_on_endtrans 设置为 Off 可提供与 Transact-SQL 兼容的行为。请参见“close_on_endtrans 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **SET datefirst { 1 | 2 | 3 | 4 | 5 | 6 | 7 }** 缺省值为 7，即在缺省情况下，每周的第一天是星期日。要永久设置此选项，请参见“first_day_of_week 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **SET quoted_identifier { On | Off }** 控制用双引号括起来的字符串是被解释为标识符 (On) 还是文字字符串 (Off)。请参见“为实现 Transact-SQL 兼容性设置选项”一节 《SQL Anywhere 服务

器 - SQL 的用法》和“quoted_identifier 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **SET rowcount** *integer* Transact-SQL ROWCOUNT 选项将为所有游标读取的行数限制为指定的整数。这包括通过重新定位游标读取的行。超出此最大值的所有读取操作都返回警告。当应 OPEN 请求返回游标的行数估计值时应考虑该选项设置。

  SET ROWCOUNT 还将受搜索的 UPDATE 或 DELETE 语句影响的行数限制为 *integer*。例如，这可以用来允许 COMMIT 语句定期执行，以限制回退日志和锁定表的大小。应用程序（或过程）需要提供一个循环，为未受第一个操作影响的行重新发出更新/删除命令。下面给出了一个简单示例：

```
BEGIN
   DECLARE @count INTEGER
   SET rowcount 20
   WHILE(1=1) BEGIN
      UPDATE Employees SET Surname='new_name'
      WHERE Surname <> 'old_name'
      /* Stop when no rows changed */
      SELECT @count = @@rowcount
      IF @count = 0 BREAK
      PRINT string('Updated ',
               @count,' rows; repeating...')
      COMMIT
   END
   SET rowcount 0
END
```

  在 SQL Anywhere 中，如果 ROWCOUNT 设置比 Interactive SQL 可显示的行数大，则 Interactive SQL 可能会执行一些额外的读取以重新定位游标。因此，实际显示的行数可能比请求的数目少。另外，如果由于截断警告而重新读取任何行，计数可能不准确。

  如果值为零，则重置该选项以获取所有行。

- **SET self_recursion { On | Off }** self_recursion 选项用于在触发器中启用 (On) 或禁止 (Off) 与触发器关联的表的操作触发其它触发器。

- **SET string_rtruncation { On | Off }** 在 SQL Anywhere 和 Adaptive Server Enterprise 中，指派 SQL 字符串数据时截断非空格字符的缺省行为是不同的。将 string_rtruncation 设置为 On 可提供与 Transact-SQL 兼容的字符串比较。请参见“string_rtruncation 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **SET textsize** 指定使用 select 语句返回的文本或图像类型数据的最大大小（以字节为单位）。@@textsize 全局变量存储当前设置。要重置为缺省大小 (32 KB)，请使用下面的命令：

```
set textsize 0
```

- **SET transaction isolation level { 0 | 1 | 2 | 3 | snapshot | statement snapshot | read only statement snapshot }** 设置当前连接的锁定隔离级别，如“隔离级别和一致性”一节 《SQL Anywhere 服务器 - SQL 的用法》中所述。对于 Adaptive Server Enterprise，只有 1 和 3 是有效选项。对于 SQL Anywhere，0、1、2、3、snapshot、statement snapshot 和 read only statement snapshot 的任一项都是有效选项。请参见“isolation_level 选项 [数据库] [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

出于兼容性考虑，SQL Anywhere 允许用 SET 语句设置 prefetch 选项，但没有效果。

权限

无。

副作用

无。

另请参见

- “SET OPTION 语句”一节第 702 页
- “为实现 Transact-SQL 兼容性设置选项”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “兼容性选项”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

SET CONNECTION 语句 [Interactive SQL] [ESQL]

此语句用于更改活动数据库连接。

语法

SET CONNECTION [*connection-name*]

connection-name : *identifier*, *string*, or *hostvar*

注释

SET CONNECTION 语句将活动数据库连接更改为 connection-name。当前连接状态被保存起来，并将在重新成为活动连接时恢复。如果省略 connection-name，并且存在未命名的连接，则该连接成为活动连接。

当在嵌入式 SQL 中打开游标时，它们与当前连接关联。如果更改连接，前一个活动连接的游标名将变得不可访问。这些游标在原地保持活动，并在关联的连接再次变为活动时恢复可访问性。

权限

无。

副作用

无。

另请参见

- “CONNECT 语句 [ESQL] [Interactive SQL]”一节第 410 页
- “DISCONNECT 语句 [ESQL] [Interactive SQL]”一节第 540 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** Interactive SQL 是服务商扩充。嵌入式 SQL 是核心特性。

示例

以下是嵌入式 SQL 中的示例。

```
EXEC SQL SET CONNECTION :conn_name;
```

在 Interactive SQL 中，将当前连接设置为名为 conn1 的连接。

```
SET CONNECTION conn1;
```

SET DESCRIPTOR 语句 [ESQL]

此语句用于描述 SQL 描述符区中的变量，并将数据放入描述符区。

语法

SET DESCRIPTOR *descriptor-name*
{ **COUNT** = { *integer* | *hostvar* }
| **VALUE** { *integer* | *hostvar* } *assignment*, ... }

assignment :
{ **TYPE** | **SCALE** | **PRECISION** | **LENGTH** | **INDICATOR** }
  = { *integer* | *hostvar* }
| **DATA** = *hostvar*

descriptor-name : identifier

注释

SET DESCRIPTOR 语句用于描述描述符区中的变量，并将数据放入描述符区。

SET ...COUNT 语句设置描述符区中描述的变量数。此数值不得超过分配描述符区时指定的变量数。

值 { *integer* | *hostvar* } 指定描述符区中要赋值的变量。

执行 SET ...DATA 时会进行类型检查，以确保描述符区中的变量类型与主机变量类型相同。LONG VARCHAR 和 LONG BINARY 不受 SET DESCRIPTOR ... DATA 支持。

如果发生错误，SQLCA 中会返回代码。

权限

无。

副作用

无。

另请参见

- “ALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 343 页
- “DEALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 522 页
- “SQL 描述符区域 (SQLDA)”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 核心 SQL 之外的 SQL/基础特性。

示例

下面的示例设置 sqlda 中位置为 col_num 的列的类型。

```
void set_type( SQLDA *sqlda, int col_num, int new_type )
{
    EXEC SQL BEGIN DECLARE SECTION;
    INT new_type1 = new_type;
    INT col = col_num;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL SET DESCRIPTOR sqlda VALUE :col TYPE = :new_type1;
}
```

有关详细示例，请参见“ALLOCATE DESCRIPTOR 语句 [ESQL]”一节第 343 页。

SET OPTION 语句

此语句用于更改数据库选项的值。

语法

SET [**EXISTING**] [**TEMPORARY**] **OPTION**
[*userid*.| **PUBLIC**.]*option-name* = [*option-value*]

userid : *identifier*, *string*, or *hostvar*

option-name : *identifier*

option-value : *string literal*

嵌入式 SQL 语法

SET [**TEMPORARY**] **OPTION**
[*userid*.| **PUBLIC**.]*option-name* = [*option-value*]

userid : *identifier*, *string*, or *hostvar*

option-name : *identifier*, *string*, or *hostvar*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

注释

SET OPTION 语句用于更改影响数据库服务器行为的选项。设置选项的值可更改所有用户的行为或仅更改个别用户的行为。更改范围可以是临时的，也可以是永久的。

任何选项，不论是否是用户定义选项，在可以指派用户特定的值之前都必须具有公共设置。对于用户定义的选项，数据库服务器不支持设置 TEMPORARY 值。

选项的分类为：

- 数据库选项
- Transact-SQL 兼容性选项
- 复制数据库选项

有关所有可用选项的列表及说明，请参见“数据库选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

可以将选项设置为三种级别的作用域：公共、用户和临时。临时选项优先于其它选项，而用户选项优先于公共选项。如果为当前用户设置用户级选项，同样会设置相应的临时选项。

语法 1 不允许为 *option-value* 指定主机变量。但可以改用 EXECUTE IMMEDIATE 语句获得所需的结果。请参见“EXECUTE IMMEDIATE 语句 [SP]”一节第 570 页。

如果使用 EXISTING 关键字，则无法为个别用户 ID 设置选项值，除非已经有该选项的 PUBLIC 用户 ID 设置。

如果指定用户 ID，则选项值应用于该用户（或者，对于组用户 ID，则适用于该组的成员）。如果指定 PUBLIC，则选项值应用于没有单独设置该选项的所有用户。缺省情况下，该选项值应用于发出 SET OPTION 语句的当前登录用户 ID。

例如，如果 DBA 是发出 SQL 语句的用户，则下面的语句将选项更改应用于用户 DBA：

```
SET OPTION precision = 40;
```

而以下语句会将更改应用于 PUBLIC 用户 ID（所有用户都属于该用户组）。

```
SET OPTION Public.login_mode = Standard;
```

只有拥有 DBA 权限的用户才有权设置 PUBLIC 用户 ID 的选项。

用户可以使用 SET OPTION 语句更改他们自己的用户 ID 值。仅当您有 DBA 权限时，才能为其他用户 ID 设置选项值。

将 TEMPORARY 关键字添加到 SET OPTION 语句中会改变更改生效的持续时间。缺省情况下，选项值是永久性的；在使用 SET OPTION 语句显式进行更改之前，它不会变化。

如果 SET TEMPORARY OPTION 语句没有用用户 ID 限定，则新的选项值仅对当前连接生效。

如果对 PUBLIC 用户 ID 使用 SET TEMPORARY OPTION，则更改在数据库运行时间内一直生效。当数据库关闭时，PUBLIC 组的 TEMPORARY 选项恢复为其永久值。

为 PUBLIC 用户 ID 设置临时选项可提供安全性优点。例如，在启用 login_mode 选项时，数据库依赖于其运行时所在的系统的登录安全性。临时启用该选项意味着，对于依赖于 Windows 域的安全性的数据库，如果关闭该数据库并将它复制到本地计算机，它的安全不会受到威胁。在这种情况下，临时启用的 login_mode 选项将恢复为它的永久值，这个永久值可能是 [Standard]，即不允许集成登录的模式。

如果忽略 *option-value*，将从数据库中删除指定的选项设置。如果它是一个个人选项设置，则它的值会恢复为 PUBLIC 设置。如果删除 TEMPORARY 选项，则选项设置会恢复为永久设置。

小心
不支持从游标中读取行时更改选项设置，因为这会导致意外的行为。例如，在从游标读取时更改 date_format 设置会导致在结果集的行中出现不同的日期格式。不要在读取行时更改选项设置。

SET OPTION 语句为 SQL Flagger 所忽略。

权限

设置自己的选项不需要任何权限。

为其他用户或 PUBLIC 设置数据库选项需要 DBA 权限。

副作用

如果不指定 TEMPORARY，将执行自动提交。

另请参见

- “数据库选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “兼容性选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SQL Remote 选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SET OPTION 语句 [Interactive SQL]”一节第 704 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

将日期格式选项设置为 on：

```
SET OPTION public.date_format = 'Mmm dd yyyy';
```

将 wait_for_commit 选项设置为 On：

```
SET OPTION wait_for_commit = 'On';
```

以下是两个嵌入式 SQL 的示例。

```
1. EXEC SQL SET OPTION :user.:option_name = :value;
2. EXEC SQL SET TEMPORARY OPTION date_format = 'mm/dd/yyyy';
```

为当前连接的用户设置 date_format 选项。在以后连接时，同一用户 ID 将使用此选项值。

```
SET OPTION date_format = 'yyyy/mm/dd';
```

以下语句删除当前用户 ID 的 date_format 选项的设置。执行此语句后，将改用针对 PUBLIC 组的 date_format 设置。

```
SET OPTION date_format=;
```

SET OPTION 语句 [Interactive SQL]

该语句用于更改 Interactive SQL 选项的值。

语法 1 - 设置选项

SET OPTION *option-name* = [*option-value*]
| **SET TEMPORARY OPTION** *option-name* = [*option-value*]

option-name : *identifier*, *string*, or *hostvar*

option-value : *string*, *identifier*, or *number*

语法 2 - 永久保存当前选项

SET PERMANENT

语法 3 - 查看当前选项

SET

注释

缺省情况下，使用 SET OPTION 语法设置选项时，将永久存储选项设置，除非另一个 SET OPTION 语句对其进行更改，否则不会改变。

使用 SET TEMPORARY OPTION 语法可以临时更改选项设置。临时设置将一直有效，直至您关闭 Interactive SQL。在下次启动 Interactive SQL 时，选项会恢复为其永久设置。

使用 SET PERMANENT 语法永久保存所有当前 Interactive SQL 选项设置（将所有临时设置变成永久设置）。

使用语法 3 显示所有当前数据库选项设置。如果数据库服务器具有临时选项设置，将显示这些临时设置而不显示永久设置。

Interactive SQL 选项设置存储在客户端计算机上，而不是在数据库中。

有关对当前 Interactive SQL 选项的说明，请参见“Interactive SQL 选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

标准和兼容性

- **SQL/2003** 服务商扩充。

另请参见

- “Interactive SQL 选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

SET REMOTE OPTION 语句 [SQL Remote]

此语句用于设置 SQL Remote 消息链接的消息控制参数。

语法

SET REMOTE *link-name* **OPTION**
[*userid*.| **PUBLIC**.]*link-option-name* = *link-option-value*

link-name:
file

| **ftp**
| **smtp**

link-option-name:
common-option
| *file-option*
| *ftp-option*
| *smtp-option*

common-option:
debug
| **output_log_send_on_error**
| **output_log_send_limit**
| **output_log_send_now**

file-option:
directory
| **invalid_extensions**
| **unlink_delay**

ftp-option:
active_mode
| **host**
| **invalid_extensions**
| **password**
| **port**
| **root_directory**
| **user**
| **reconnect_retries**
| **reconnect_pause**

smtp-option:
local_host
| **pop3_host**
| **pop3_password**
| **pop3_userid**
| **smtp_host**
| **top_supported**

link-option-value : *string*

参数

- **userid** 如果不指定 *userid*，则假定为当前发布者。
- **options** 选项值与消息链接相关。有关详细信息，请参见：
  - “FILE 消息系统”一节 《SQL Remote》
  - “FTP 消息系统”一节 《SQL Remote》
  - “SMTP 消息系统”一节 《SQL Remote》

注释

如果是首次使用消息链接，则用户在消息链接窗口中输入消息链接参数时，消息代理会保存这些参数。在这种情况下，不必显式使用此语句。此语句在准备统一数据库以便抽取许多数据库时最为有用。

选项名称区分大小写。选项值是否区分大小写取决于具体的选项：布尔值区分大小写，而口令、目录名和其它字符串是否区分大小写则取决于文件系统（对于目录名）或数据库（对于用户 ID 和口令）是否区分大小写。

权限

必须具有 DBA 权限。发布者可以设置他们自己的选项。

副作用

自动提交。

另请参见

- “从远程数据库收集错误”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

对于用户 myuser 的 FTP 链接，以下语句将 FTP 主机设置为 *ftp.mycompany.com*：

```
SET REMOTE FTP OPTION myuser.host = 'ftp.mycompany.com';
```

以下语句使 SQL Remote 停止使用生成消息的指定文件扩展名：

```
SET REMOTE ftp OPTION "Public"."invalid_extensions" =
'exe,pif,dll,bat,cmd,vbs';
```

SET SQLCA 语句 [ESQL]

此语句用于指示 SQL 预处理器使用缺省全局 *sqlca* 以外的 SQLCA。

语法

SET SQLCA *sqlca*

sqlca : *identifier* or *string*

注释

SET SQLCA 语句指示 SQL 预处理器使用缺省全局 *sqlca* 以外的 SQLCA。sqlca 必须是作为对 SQLCA 指针的 C 语言引用的标识符或字符串。

当前 SQLCA 指针隐式传递给各嵌入式 SQL 语句中的数据库接口库。C 源文件中位于此语句后的所有嵌入式 SQL 语句将使用新的 SQLCA。

仅当编写重入代码时才需要使用此语句（请参见“多线程代码或重入代码的 SQLCA 管理”一节 《SQL Anywhere 服务器 - 编程》）。sqlca 应该引用局部变量。任何全局变量或模块静态变量都可能会由另一线程修改。

权限

无。

副作用

无。

另请参见

- “多线程代码或重入代码的 SQLCA 管理”一节 《SQL Anywhere 服务器 - 编程》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在 Windows DLL 中可以找到拥有的函数。每个使用 DLL 的应用程序都有自己的 SQLCA。

```
an_sql_code FAR PASCAL ExecuteSQL( an_application *app, char *com )
{
   EXEC SQL BEGIN DECLARE SECTION;
   char *sqlcommand;
   EXEC SQL END DECLARE SECTION;
   EXEC SQL SET SQLCA "&app->.sqlca";
   sqlcommand = com;
   EXEC SQL WHENEVER SQLERROR CONTINUE;
   EXEC SQL EXECUTE IMMEDIATE :sqlcommand;
return( SQLCODE );
}
```

SETUSER 语句

此语句用于允许数据库管理员模拟其他用户和启用连接池。

语法

{ **SET SESSION AUTHORIZATION** | **SETUSER** }
[[**WITH OPTION**] *userid*]

参数

- **WITH OPTION 子句** 缺省情况下，只变更权限（包括组成员资格）。如果指定 WITH OPTION，生效的数据库选项更改为 *userid* 的当前数据库选项。
- **userid** 用户 ID 是标识符（SETUSER 语法）或字符串（SET SESSION AUTHORIZATION 语法）。请参见“标识符”一节第 8 页和“字符串”一节第 9 页。

注释

提供 SETUSER 语句是为了使数据库管理变得更容易。它使具有 DBA 权限的用户能够模拟数据库的其他用户。当运行 SETUSER 语句后，可以通过运行以下命令之一来检查模拟的是哪一个用户：

- SELECT USER
- SELECT CURRENT USER

也可以从应用程序服务器使用 SETUSER 以利用连接池。连接池可减少需要建立的非重复连接的数量，从而提高性能。

不带用户 ID 的 SETUSER 撤消前面所有的 SETUSER 语句。

过程、触发器、事件处理程序和批处理中不能使用 SETUSER 语句。

SETUSER 语句有以下几种用途：

- **创建对象** 可以使用 SETUSER 创建由其他用户拥有的数据库对象。
- **权限检查** 通过充当其他用户（具有他们的权限和组成员资格），数据库管理员可以测试查询、过程、视图等的权限和名称解析。
- **为管理员提供更安全的环境** 数据库管理员具有在数据库中执行任何操作的权限。如果想要确保不会在无意中执行错误的操作，可使用 SETUSER 切换为具有较少权限的另一用户 ID。

注意
过程、触发器、事件或批处理中不能使用 SETUSER 语句。

权限

必须具有 DBA 权限。

另请参见

- “EXECUTE IMMEDIATE 语句 [SP]”一节第 570 页
- “GRANT 语句”一节第 595 页
- “REVOKE 语句”一节第 679 页
- “SET OPTION 语句”一节第 702 页

标准和兼容性

- **SQL/2003** SET SESSION AUTHORIZATION 是核心特性。SETUSER 是服务商扩充。

示例

下面的语句由名为 DBA 的用户执行，它将用户 ID 更改为 Joe，然后更改为 Jane，最后改回 DBA。

```
SETUSER "Joe"
// ... operations...
SETUSER WITH OPTION "Jane"
// ... operations...
SETUSER
```

以下语句将用户设置为 Jane。该用户 ID 作为字符串提供（而不是标识符）。

```
SET SESSION AUTHORIZATION 'Jane';
```

SIGNAL 语句

此语句用于发出异常情况的信号。

语法

SIGNAL *exception-name*

注释

SIGNAL 允许您引发异常。有关如何处理异常的说明，请参见“在过程和触发器中使用异常处理程序”一节 《SQL Anywhere 服务器 - SQL 的用法》。

使用 *exception-name* 来指定在当前复合语句的开头使用 DECLARE 语句声明的异常的名称。该异常必须对应于系统定义的 SQLSTATE 或用户定义的 SQLSTATE。用户定义的 SQLSTATE 值必须介于 99000 和 99999 之间。

权限

无。

副作用

无。

另请参见

- “RESIGNAL 语句”一节第 675 页
- “BEGIN 语句”一节第 395 页
- “在过程和触发器中使用异常处理程序”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 持久存储模块特性。

示例

下面的复合语句声明一个用户定义的异常并发出用户定义异常的信号。如果在 Interactive SQL 中执行此示例，则消息 [**My exception signaled**] 会显示在 [**结果**] 区域的 [**消息**] 选项卡中。

```
BEGIN
   DECLARE myexception EXCEPTION
   FOR SQLSTATE '99001';
   SIGNAL myexception;
   EXCEPTION
      WHEN myexception THEN
         MESSAGE 'My exception signaled'
         TO CLIENT;
END
```

START DATABASE 语句

此语句用于启动当前数据库服务器上的数据库。

语法

START DATABASE *database-file* [*start-options* ...]

start-options :
[**AS** *database-name*]

[**ON** *database-server-name*]
[**WITH TRUNCATE AT CHECKPOINT**]
[**FOR READ ONLY**]
[**AUTOSTOP** { **ON** | **OFF** }]
[**KEY** *key*]
[**WITH SERVER NAME** *alternative-database-server-name*]
[**DIRECTORY** *dbspace-directory*]

参数

- **database-file** *database-file* 参数是一个字符串。如果在 *database-file* 中提供了相对路径，则该路径相对于数据库服务器的起始目录。
- **start-options 子句** *start-options* 可按任何顺序列出：
  - **AS 子句** 如果未指定 *database-name*，则为数据库分配一个缺省名称。此缺省名称是数据库文件的根。例如，会为文件 *C:\Database Files\demo.db* 中的数据库指定缺省名称 demo。*database-name* 参数是标识符。
  - **ON 子句** 此子句仅受 Interactive SQL 的支持。在 Interactive SQL 中，如果未指定 *server-name*，则缺省服务器是当前正在运行的服务器中最先启动的服务器。*server-name* 参数是标识符。
  - **WITH TRUNCATE AT CHECKPOINT 子句** 通过启用检查点处日志截断的选项来启动数据库。
  - **FOR READ ONLY 子句** 以只读模式启动数据库。在需要恢复的数据库上使用此语句时，此语句会失败并返回错误消息。
  - **AUTOSTOP 子句** AUTOSTOP 子句的缺省设置为 ON。如果 AUTOSTOP 设置为 ON，则在删除连接数据库的最后一个连接时卸载数据库。如果 AUTOSTOP 设置为 OFF，则不卸载数据库。

    在 Interactive SQL 中，可以使用 YES 或 NO 代替 ON 和 OFF。
  - **KEY 子句** 如果数据库是高度加密的，使用该子句输入 KEY 值（口令）。
  - **WITH SERVER NAME 子句** 此子句用于指定在连接到此数据库时的数据库服务器的备用名称。如果您使用的是数据库镜像，则主服务器和镜像服务器必须具有相同的数据库服务器名称，因为客户端不知道它们将连接到哪台服务器上。

    有关替代服务器名和数据库镜像的详细信息，请参见“-sn 数据库选项”一节 《SQL Anywhere 服务器 - 数据库管理》和“数据库镜像简介”一节 《SQL Anywhere 服务器 - 数据库管理》。
  - **DIRECTORY 子句** 使用此子句指定为要启动的数据库放置 dbspace 文件的目录。例如，如果数据库服务器在与 dbspace 相同的目录中启动，并且您包括了 `DIRECTORY '.'` 子句，则这会指示数据库服务器在当前目录中查找全部 dbspace。请参见“-ds 数据库选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

注释

在当前数据库服务器上启动指定的数据库。

如果没有与数据库连接而想要使用 START DATABASE 语句，则必须首先连接到某个数据库，例如实用程序数据库。

有关实用程序数据库的信息，请参见“使用实用程序数据库”一节 《SQL Anywhere 服务器 - 数据库管理》。

START DATABASE 语句不将当前应用程序连接到指定的数据库：仍需要显式连接。

Interactive SQL 支持 ON 子句，它允许数据库在非当前数据库服务器上启动。

只能使用数据库名 utility_db 连接 SQL Anywhere 实用程序数据库。请参见“使用实用程序数据库”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

所需权限由数据库服务器的 -gd 选项指定。此选项在个人数据库服务器上缺省为 all，在网络服务器上缺省为 DBA。

副作用

None

另请参见

- “STOP DATABASE 语句”一节第 718 页
- “CONNECT 语句 [ESQL] [Interactive SQL]”一节第 410 页
- “-gd 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

在当前服务器中启动数据库文件 *C:\Database Files\sample_2.db*。

```
START DATABASE 'c:\database files\sample_2.db';
```

在 Interactive SQL 中，在名为 sample 的服务器上将数据库文件 *c:\Database Files\sample_2.db* 作为 sam2 启动。

```
START DATABASE 'c:\database files\sample_2.db'
AS sam2
ON sample;
```

START ENGINE 语句 [Interactive SQL]

此语句用于启动数据库服务器。

语法

START ENGINE AS *database-server-name* [**STARTLINE** *command-string*]

注释

START ENGINE 语句启动数据库服务器。如果想要为数据库服务器指定一组选项，请将 STARTLINE 关键字与命令字符串一起使用。符合“SQL Anywhere 数据库服务器”一节 《SQL Anywhere 服务器 - 数据库管理》中的数据库服务器说明的命令字符串即为有效。

权限

None

副作用

None

另请参见

- “STOP ENGINE 语句”一节第 719 页
- “SQL Anywhere 数据库服务器”一节 《SQL Anywhere 服务器 - 数据库管理》
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

启动名为 sample 的数据库服务器，但不启动其中的任何数据库。

```
START ENGINE AS sample;
```

以下示例显示 STARTLINE 子句的用法。

```
START ENGINE AS eng1 STARTLINE 'dbeng11 -c 8M';
```

START EXTERNAL ENVIRONMENT 语句

此语句用于启动外部环境。

语法

START EXTERNAL ENVIRONMENT *environment-name*

environment-name :
JAVA
| PERL
| PHP
| CLR
| C_ESQL32
| C_ESQL64
| C_ODBC32
| C_ODBC64

参数

- **environment-name** 要停止的外部环境的名称。

注释

有关外部环境的详细信息，请参见“外部环境概述”一节 《SQL Anywhere 服务器 - 编程》。

权限

必须具有 DBA 权限。

副作用

None

另请参见

- “外部环境概述”一节 《SQL Anywhere 服务器 - 编程》
- “ALTER EXTERNAL ENVIRONMENT 语句”一节第 353 页
- “STOP EXTERNAL ENVIRONMENT 语句”一节第 720 页
- “INSTALL EXTERNAL OBJECT 语句”一节第 620 页
- “REMOVE EXTERNAL OBJECT 语句”一节第 672 页
- “SYSEXTERNENV 系统视图”一节第 947 页
- “SYSEXTERNENVOBJECT 系统视图”一节第 948 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

启动 Perl 外部环境。

```
START EXTERNAL ENVIRONMENT PERL;
```

START JAVA 语句

此语句用于启动 Java VM。

语法

START JAVA

注释

START JAVA 语句启动 Java VM。主要用途是在方便的时候装载 Java VM，这样当用户开始使用 Java 功能时，不会出现装载 Java VM 时的初始暂停。

必须设置数据库服务器才能查找 Java VM。由于可为每个数据库指定不同的 Java VM，因此，可使用 java_location 选项指明 Java VM 的位置（路径）。请参见“java_location 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关启动 Java VM 的详细信息，请参见“启动和停止 Java VM”一节 《SQL Anywhere 服务器 - 编程》。

权限

必须安装 Java VM，而且数据库必须支持 Java。

Windows Mobile 上不支持此语句。

副作用

None

另请参见

- “STOP JAVA 语句”一节第 721 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

启动 Java VM。

```
START JAVA;
```

START LOGGING 语句 [Interactive SQL]

此语句用于开始将执行的 SQL 语句记录到日志文件中。

语法

START LOGGING *filename*

注释

START LOGGING 语句开始将所有后续执行的 SQL 语句复制到您指定的日志文件中。如果该文件不存在，则 Interactive SQL 将创建该日志文件。记录将一直进行，直到使用 STOP LOGGING 语句显式停止记录过程或结束当前的 Interactive SQL 会话。也可以通过单击 [SQL] » [开始记录] 和 [SQL] » [停止记录] 来开始和停止记录。

权限

无。

副作用

无。

另请参见

- “STOP LOGGING 语句 [Interactive SQL]”一节第 721 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

开始记录到 c: 目录下名为 *filename.sql* 的文件。

```
START LOGGING 'c:\filename.sql';
```

START SUBSCRIPTION 语句 [SQL Remote]

此语句用于启动用户对发布的预订。

语法

START SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*, ...

参数

- **publication-name** 用户所预订的发布的名称。这可以包括发布的所有者。
- **subscription-value** 与发布的预订表达式进行比较的字符串。因为每个预订者可能会有多个发布预订，则此处必须使用该值。
- **subscriber-id** 发布预订者的用户 ID。此用户必须具有发布的预订。

注释

发布更新从统一数据库发送到远程数据库时，即称启动了 SQL Remote 预订。

START SUBSCRIPTION 语句是一组管理预订的语句之一。CREATE SUBSCRIPTION 语句定义预订者要接收的数据。SYNCHRONIZE SUBSCRIPTION 语句确保在统一数据库与远程数据库之间保持一致。需要使用 START SUBSCRIPTION 语句才能启动发送给预订者的消息。

在预订启动之前，预订两端的数据必须保持一致。建议使用数据库抽取实用程序管理预订的创建、同步及启动。如果使用数据库抽取实用程序，则不需要执行显式 START SUBSCRIPTION 语句。此外，一旦预订同步，消息代理将立即启动预订。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE SUBSCRIPTION 语句 [SQL Remote]”一节第 492 页
- “REMOTE RESET 语句 [SQL Remote]”一节第 671 页
- “SYNCHRONIZE SUBSCRIPTION 语句 [SQL Remote]”一节第 724 页
- “STOP SUBSCRIPTION 语句 [SQL Remote]”一节第 722 页
- “抽取实用程序 (dbxtract)”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句启动用户 SamS 对 pub_contact 发布的预订。


```
START SUBSCRIPTION TO pub_contact
FOR SamS;
```

START SYNCHRONIZATION DELETE 语句 [MobiLink]

此语句可用于为 MobiLink 同步重新启动有关删除操作的日志记录。

语法

START SYNCHRONIZATION DELETE

注释

通常，SQL Anywhere 和 UltraLite 自动记录对属于同步的表或列所作的任何更改，并在下一次同步期间将这些更改上载到统一数据库。使用 STOP SYNCHRONIZATION DELETE 语句可以暂时停止对删除操作的自动记录。START SYNCHRONIZATION DELETE 语句允许您重新启动自动记录。

执行 STOP SYNCHRONIZATION DELETE 语句之后，不会同步在此连接上执行的任何删除操作。效果一直持续到执行 START SYNCHRONIZATION DELETE 语句为止。重复执行 STOP SYNCHRONIZATION DELETE 不会有额外的效果。

单个 START SYNCHRONIZATION DELETE 语句可重新启动记录，无论在它前面执行了多少个 STOP SYNCHRONIZATION DELETE 语句。

如果您的应用程序不同步数据，则不要使用 START SYNCHRONIZATION DELETE。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “STOP SYNCHRONIZATION DELETE 语句 [MobiLink]”一节第 723 页
- “StartSynchronizationDelete 方法”一节 《UltraLite - .NET 编程》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的 SQL 语句序列阐释了如何使用 START SYNCHRONIZATION DELETE 和 STOP SYNCHRONIZATION DELETE：

```
-- Prevent deletes from being sent
-- to the consolidated database
STOP SYNCHRONIZATION DELETE;

-- Remove all records older than 1 month
-- from the remote database,
-- NOT the consolidated database
DELETE FROM PROPOSAL
```



```
WHERE last_modified < months( CURRENT TIMESTAMP, -1 )

-- Re-enable all deletes to be sent
-- to the consolidated database
-- DO NOT FORGET to start this
START SYNCHRONIZATION DELETE;

-- Commit the entire operation,
-- otherwise rollback everything
-- including the stopping of the deletes
commit;
```

STOP DATABASE 语句

此语句用于停止当前数据库服务器上的数据库。

语法

STOP DATABASE *database-name*
[**ON** *database-server-name*]
[**UNCONDITIONALLY**]

参数

- **STOP DATABASE 子句** *database-name* 是在当前服务器上运行的数据库（不是当前数据库）的名称。
- **ON 子句** 此子句仅受 Interactive SQL 的支持。如果未在 Interactive SQL 中指定 *database-server-name*，则将搜索所有运行的服务器，查找具有指定名称的数据库。

  如果不在 Interactive SQL 中使用此语句，则仅停止在当前数据库服务器上启动的数据库。
- **UNCONDITIONALLY 子句** 停止数据库，即使存在到数据库的连接。缺省情况下，如果存在到数据库的连接，则不停止该数据库。

注释

STOP DATABASE 语句停止当前数据库服务器上的指定数据库。

权限

所需权限由数据库服务器的 -gk 选项指定。此选项在个人数据库服务器上缺省为 all，在网络服务器上缺省为 DBA。

对当前连接到的数据库不能使用 STOP DATABASE。

副作用

None

另请参见

- “START DATABASE 语句”一节第 710 页
- “DISCONNECT 语句 [ESQL] [Interactive SQL]”一节第 540 页
- “-gd 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

停止当前服务器上名为 *sample* 的数据库。

```
STOP DATABASE sample;
```

STOP ENGINE 语句

此语句用于停止数据库服务器。

语法

STOP ENGINE [*database-server-name*] [**UNCONDITIONALLY**]

参数

- **STOP ENGINE 子句** *database-server-name* 只能在 Interactive SQL 中使用。如果不是在 Interactive SQL 中运行此语句，则会停止当前数据库服务器。
- **UNCONDITIONALLY 子句** 如果您的连接是数据库服务器的唯一连接，则不需要使用 UNCONDITIONALLY。如果还有其它连接，则只有当您使用 UNCONDITIONALLY 关键字时，数据库服务器才会停止。

注释

STOP ENGINE 语句停止指定的数据库服务器。如果提供了 UNCONDITIONALLY 关键字，即使存在其它到数据库服务器的连接，该数据库服务器也会停止。缺省情况下，如果存在其它连接，则数据库服务器不会停止。

在存储过程、触发器、事件或批处理中不能使用 STOP ENGINE 语句。

权限

关闭服务器的权限取决于数据库服务器命令行中的 -gk 设置。个人服务器的缺省设置为 all，而网络服务器的缺省设置为 DBA。

副作用

None

另请参见

- “START ENGINE 语句 [Interactive SQL]”一节第 712 页
- “-gk 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

只要没有其它连接，就停止当前的数据库服务器。

```
STOP ENGINE;
```

STOP EXTERNAL ENVIRONMENT 语句

此语句用于停止外部环境。

语法

STOP EXTERNAL ENVIRONMENT *environment-name*

environment-name :
JAVA
| **PERL**
| **PHP**
| **CLR**
| **C_ESQL32**
| **C_ESQL64**
| **C_ODBC32**
| **C_ODBC64**

参数

- **environment-name** 要停止的外部环境的名称。

注释

有关外部环境的详细信息，请参见“外部环境概述”一节 《SQL Anywhere 服务器 - 编程》。

权限

必须具有 DBA 权限。

副作用

None

另请参见

- “外部环境概述”一节 《SQL Anywhere 服务器 - 编程》
- “ALTER EXTERNAL ENVIRONMENT 语句”一节第 353 页
- “START EXTERNAL ENVIRONMENT 语句”一节第 713 页
- “INSTALL EXTERNAL OBJECT 语句”一节第 620 页
- “REMOVE EXTERNAL OBJECT 语句”一节第 672 页
- “SYSEXTERNENV 系统视图”一节第 947 页
- “SYSEXTERNENVOBJECT 系统视图”一节第 948 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

停止 Perl 外部环境。

```
STOP EXTERNAL ENVIRONMENT PERL;
```

STOP JAVA 语句

此语句用于停止 Java VM。

语法

STOP JAVA

注释

STOP JAVA 语句卸载不在使用的 Java VM。主要用途是提高系统资源的使用效率。

权限

Windows Mobile 上不支持此语句。

副作用

None

另请参见

- “START JAVA 语句”一节第 714 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

停止 Java VM。

```
STOP JAVA;
```

STOP LOGGING 语句 [Interactive SQL]

此语句用于停止记录当前会话中的 SQL 语句。

语法

STOP LOGGING

注释

STOP LOGGING 语句阻止 Interactive SQL 将执行的每条 SQL 语句写入到日志文件中。您可以使用 START LOGGING 语句开始记录。也可以通过单击 [SQL] » [开始记录] 和 [SQL] » [停止记录] 来开始和停止记录。

权限

无。

副作用

无。

另请参见

- “START LOGGING 语句 [Interactive SQL]”一节第 715 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例停止当前记录会话。

```
STOP LOGGING;
```

STOP SUBSCRIPTION 语句 [SQL Remote]

此语句用于停止用户对发布的预订。

语法

STOP SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *subscriber-id*, ...

参数

- **publication-name** 用户所预订的发布的名称。这可以包括发布的所有者。
- **subscription-value** 与发布的预订表达式进行比较的字符串。因为每个预订者可能会有多个发布预订，则此处必须使用该值。
- **subscriber-id** 发布预订者的用户 ID。此用户必须具有发布的预订。

注释

发布更新从统一数据库发送到远程数据库时，即称启动了 SQL Remote 预订。

STOP SUBSCRIPTION 语句阻止继续向预订者发送消息。需要使用 START SUBSCRIPTION 语句才能重新启动发送给预订者的消息。但是，应确保在重新启动前已正确同步预订：即没有丢失任何消息。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “DROP SUBSCRIPTION 语句 [SQL Remote]”一节第 555 页
- “START SUBSCRIPTION 语句 [SQL Remote]”一节第 716 页
- “SYNCHRONIZE SUBSCRIPTION 语句 [SQL Remote]”一节第 724 页
- “抽取实用程序 (dbxtract)”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句停止用户 SamS 对 pub_contact 发布的预订。

```
STOP SUBSCRIPTION TO pub_contact
FOR SamS;
```

STOP SYNCHRONIZATION DELETE 语句 [MobiLink]

此语句可用于为 MobiLink 同步暂时停止有关删除操作的日志记录。

语法

STOP SYNCHRONIZATION DELETE

注释

通常，SQL Anywhere 和 UltraLite 远程数据库自动记录对同步中包含的表或列所作的任何更改，然后在下一次同步期间将这些更改上载到统一数据库。此语句允许您暂时停止记录对 SQL Anywhere 或 UltraLite 远程数据库的删除操作。

执行 STOP SYNCHRONIZATION DELETE 语句之后，不会同步在此连接上执行的所有后续删除操作。在执行 START SYNCHRONIZATION DELETE 语句之前始终有效。

重复执行 STOP SYNCHRONIZATION DELETE 不会有额外的效果。单个 START SYNCHRONIZATION DELETE 语句可重新启动记录，无论在它前面执行了多少个 STOP SYNCHRONIZATION DELETE 语句。

此语句对纠正远程数据库很有用，但要谨慎使用，因为它能有效地禁止 MobiLink 同步。

如果您的应用程序不同步数据，则不要使用 STOP SYNCHRONIZATION DELETE。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- UltraLite“StartSynchronizationDelete 方法”一节 《UltraLite - .NET 编程》
- UltraLite“StopSynchronizationDelete 方法”一节 《UltraLite - .NET 编程》
- “START SYNCHRONIZATION DELETE 语句 [MobiLink]”一节第 717 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

有关示例内容，请参见“START SYNCHRONIZATION DELETE 语句 [MobiLink]”一节第 717 页。

SYNCHRONIZE SUBSCRIPTION 语句 [SQL Remote]

此语句用于同步用户对发布的预订。

语法

SYNCHRONIZE SUBSCRIPTION
TO *publication-name* [**(** *subscription-value* **)**]
FOR *remote-user*, ...

参数

- **publication-name** 用户所预订的发布的名称。这可以包括发布的所有者。
- **subscription-value** 与发布的预订表达式进行比较的字符串。因为每个预订者可能会有多个发布预订，则此处必须使用该值。
- **remote-user** 发布预订者的用户 ID。此用户必须具有发布的预订。

注释

当远程数据库中的数据与统一数据库中的数据一致，使得从统一数据库发送到远程数据库的发布更新不会导致冲突和错误时，即称 SQL Remote 预订已**同步**。

要同步预订，需要将统一数据库的发布中的数据副本发送到远程数据库。SYNCHRONIZE SUBSCRIPTION 语句通过消息系统完成此操作。建议尽可能使用数据库抽取实用程序 (dbxtract) 来同步预订，而不使用消息系统。

权限

必须具有 DBA 权限。

副作用

自动提交。

另请参见

- “CREATE SUBSCRIPTION 语句 [SQL Remote]”一节第 492 页
- “START SUBSCRIPTION 语句 [SQL Remote]”一节第 716 页
- “STOP SUBSCRIPTION 语句 [SQL Remote]”一节第 722 页
- “抽取实用程序 (dbxtract)”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下语句同步用户 SamS 对 pub_contact 发布的预订。

```
SYNCHRONIZE SUBSCRIPTION
   TO pub_contact
   FOR SamS;
```

SYSTEM 语句 [Interactive SQL]

此语句用于从 Interactive SQL 内启动可执行文件。

语法

SYSTEM *'* [*path*] *filename '*

注释

启动指定的可执行文件。

- SYSTEM 语句必须全部位于一行中。
- SYSTEM 语句后面不能有注释。
- 将路径和文件名用单引号引起来。

权限

无。

副作用

无。

另请参见

- “CONNECT 语句 [ESQL] [Interactive SQL]”一节第 410 页
- “使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的语句启动记事本程序，假定记事本可执行程序位于您的 PATH 中。

```
SYSTEM 'notepad.exe';
```

TRIGGER EVENT 语句

此语句用于触发指定的事件。事件可以是为事件触发器定义的事件，或是调度事件。

语法

TRIGGER EVENT *event-name* [**(** *parm* = *value*, ... **)**]

参数

- **parm = value**　当触发条件导致执行事件处理程序时，数据库服务器可以使用 event_parameter 函数为事件处理程序提供上下文信息。TRIGGER EVENT 语句使您可以显式提供这些参数，以模拟事件处理程序的上下文。

注释

操作与特定的触发器条件或 CREATE EVENT 语句的调度有关。即使没有到达调度时间或出现触发器条件，您也可以使用 TRIGGER EVENT 语句强制执行事件处理程序。TRIGGER EVENT 不执行已禁用的事件处理程序。

每个 *value* 都是一个字符串。每个 *value* 的最大长度受到 -gp 服务器选项所指定的最大页面大小的限制。如果 *value* 的长度超过页面大小，则字符串将在页面填满的位置被截断。

权限

必须具有 DBA 权限。

副作用

无。

另请参见

- “-gp 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “ALTER EVENT 语句”一节第 351 页
- “CREATE EVENT 语句”一节第 429 页
- “EVENT_PARAMETER 函数 [System]”一节第 191 页

标准和兼容性

- **SQL/2003**　服务商扩充。

示例

以下示例说明如何将字符串参数传递给一个事件。该事件显示其在数据库服务器消息窗口中触发的时间。

```
CREATE EVENT ev_PassedParameter
HANDLER
```

```
BEGIN
  MESSAGE 'ev_PassedParameter - was triggered at ' ||
event_parameter( 'time' );
END;
TRIGGER EVENT ev_PassedParameter( "Time"=string( current timestamp ) );
```

TRUNCATE 语句

此语句用于删除表中的所有行，但不删除表定义。

语法

TRUNCATE
TABLE [*owner.*]*table-name*
| **MATERIALIZED VIEW** [*owner.*]*materialized-view-name*

注释

TRUNCATE 语句会删除表或实例化视图中的所有行。

> **注意**
> 在同步或复制所涉及的数据库上应谨慎使用 TRUNCATE TABLE 语句，因为该语句与没有 WHERE 子句的 DELETE 语句相似，它会删除表中的所有行。但是 TRUNCATE 语句不会导致触发任何触发器。此外，行的删除操作不记入事务日志，因此不会进行同步或复制。这样可导致出现不一致的情况，这种不一致会引起同步或复制失败。

执行 TRUNCATE 语句后，对象的模式及所有索引仍然存在，直到发出 DROP 语句为止。模式定义和约束保持不变，触发器和权限仍然有效。

table-name 可以是基表的名称，也可以是临时表的名称。

使用 TRUNCATE TABLE 时，如果满足以下所有条件，则执行快速表截断：

- 没有到表的外键，也没有来自表的外键。
- TRUNCATE TABLE 语句不在触发器内执行。
- TRUNCATE TABLE 语句不在原子语句内执行。

如果执行快速截断，事务日志中不会记录单独的 DELETE 操作，而且在截断操作之前和之后都会执行 COMMIT 操作。快速截断不能在快照事务内使用。请参见“快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》。

如果试图对构建了快速文本索引的表或快速视图所引用的表使用 TRUNCATE TABLE，则截断会失败。对于非快速文本索引或实例化视图则不会出现这种情况；但强烈建议您先截断相关索引和实例化视图中的数据，再对表执行 TRUNCATE TABLE 语句，然后刷新索引和实例化视图。请参见“TRUNCATE 语句”一节第 727 页和“TRUNCATE TEXT INDEX 语句”一节第 728 页。

权限

- 必须是表的所有者，具有 DBA 权限或表的 ALTER 权限。

- 对于基表和实例化视图，TRUNCATE 语句需要有表的独占访问权，因为操作是原子操作（要么删除所有行，要么不删除任何行）。这意味着必须将任何打开的引用了要截断的表的游标关闭，还必须发出 COMMIT 或 ROLLBACK 语句以释放对表的引用。
- 对于临时表，由于每个用户都有自己的数据副本，因此执行 TRUNCATE 语句时不需要独占访问。

副作用

- 截断实例化视图时，您会将视图的状态更改为未初始化。请参见“实例化视图状态和属性”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- TRUNCATE 语句不会触发 Delete 触发器。
- 执行 TRUNCATE 语句之前和之后都会执行 COMMIT。
- 各行删除操作不记入事务日志，因此不复制 TRUNCATE 操作。不要在 SQL Remote 复制或 MobiLink 远程数据库中使用此语句。
- 如果表包含的列被定义为 DEFAULT AUTOINCREMENT 或 DEFAULT GLOBAL AUTOINCREMENT，则截断操作将重置该列的下一个可用值。

另请参见

- “DELETE 语句”一节第 530 页
- “TRUNCATE TEXT INDEX 语句”一节第 728 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

删除 Departments 表的所有行：

```
TRUNCATE TABLE Departments;
```

TRUNCATE TEXT INDEX 语句

删除 MANUAL 或 AUTO REFRESH 文本索引中的数据。

语法

TRUNCATE TEXT INDEX *text-index-name*
ON [*owner*.]*table-name*

参数

- **ON 子句** 构建了文本索引的表的名称。

注释

如果您想要从手动文本索引中删除数据，但不想删除文本索引定义，则使用 TRUNCATE TEXT INDEX 语句。例如，如果要变更文本索引的文本配置对象以更改非索引字表，您必须首先截断文本索引，接着更改文本索引所引用的文本配置对象，然后刷新文本索引以使其填充新数据。

不能对定义为 IMMEDIATE REFRESH（缺省值）的文本索引执行 TRUNCATE TEXT INDEX 语句。而对于 IMMEDIATE REFRESH 文本索引，您必须将该索引删除。

权限

- 必须是构建了文本索引的表的所有者、具有 DBA 权限或对表拥有 ALTER 权限。
- TRUNCATE TEXT INDEX 需要有表的独占访问权。这意味着必须将所有打开的引用了要截断的表的游标关闭，还必须发出 COMMIT 或 ROLLBACK 语句以释放对表的引用。

副作用

自动提交

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE TEXT INDEX 语句”一节第 509 页
- “ALTER TEXT INDEX 语句”一节第 383 页
- “DROP TEXT INDEX 语句”一节第 560 页
- “REFRESH TEXT INDEX 语句”一节第 668 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

第一条语句创建 txt_index_manual 文本索引。第二条语句用数据填充该文本索引。第三条语句截断文本索引数据。

```
CREATE TEXT INDEX txt_index_manual ON MarketingInformation ( Description )
   MANUAL REFRESH;
REFRESH TEXT INDEX txt_index_manual ON MarketingInformation;
TRUNCATE TEXT INDEX txt_index_manual ON MarketingInformation;
```

截断的文本索引会在下次刷新时进行数据填充。

UNION 子句

此语句用于合并两个或更多选择语句的结果。

语法

[**WITH** *temporary-views*] *query-block*
UNION [**ALL** | **DISTINCT**] *query-block*
[**ORDER BY** [*integer* | *select-list-expression-name*] [**ASC** | **DESC**], ...]

[**FOR XML** *xml-mode*]
[**OPTION(** *query-hint*, ... **)**]

query-block：请参见“SQL 语法中的常见元素”一节第 340 页

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| *option-name* = *option-value*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

参数

- **OPTION 子句** 此子句用于指定执行语句时的提示。支持以下提示：
  - MATERIALIZED VIEW OPTIMIZATION *option-value*
  - FORCE OPTIMIZATION
  - *option-name* = *option-value*

  有关这些选项的说明，请参见“SELECT 语句”一节第 689 页的 OPTIONS 子句。

注释

使用 UNION 可将多个查询块的结果合并成一个较大的结果。每个 *query-block* 在选择列表中必须具有相同数目的项目。

UNION ALL 的结果是查询块的合并结果。UNION 的结果与 UNION ALL 的结果相同，但会删除重复的行。删除重复的行需要额外的处理，所以应尽量使用 UNION ALL 而不是 UNION。UNION DISTINCT 与 UNION 相同。

如果两个选择列表中的相应项具有不同的数据类型，则 SQL Anywhere 为结果中的相应列选择数据类型，并自动相应地转换各 *query-block* 中的列。

UNION 的第一个查询块用于确定要与 ORDER BY 子句匹配的名称。

显示的列名与为第一个 *query-block* 显示的列名相同。另一种自定义结果集列名的方法是在 *query-block* 上使用 WITH 子句。

权限

必须具有对各 *query-block* 的 SELECT 权限。

副作用

无。

另请参见

- “EXCEPT 子句”一节第 565 页
- “INTERSECT 子句”一节第 623 页
- “UNION 子句”一节第 729 页
- “SELECT 语句”一节第 689 页

标准和兼容性

- **SQL/2003** 核心特性。

示例

列出雇员和客户的所有不重复的姓。

```
SELECT Surname
FROM Employees
UNION
SELECT Surname
FROM Customers;
```

UNLOAD 语句

此语句用于将数据从数据源卸载到文件中。

语法

UNLOAD *data-source*
{ **TO** *filename*
  | **INTO FILE** *filename*
  | **INTO CLIENT FILE** *client-filename*
  | **INTO VARIABLE** *variable-name* }
[*unload-option* ...]

data-source
[**FROM**] [**TABLE**] [*owner.*]*table-name*
| [**FROM**] [**MATERIALIZED VIEW**] [*owner.*]*materialized-view-name*
| *select-statement*

filename : *string* | *variable*

client-filename : *string* | *variable*

语法

unload-option :
APPEND { **ON** | **OFF** }
| **BYTE ORDER MARK** { **ON** | **OFF** }
| { **COMPRESSED** | **NOT COMPRESSED** }
| { **COMPRESSED** | **NOT COMPRESSED** }
| **COLUMN DELIMITED BY** *string*
| **DELIMITED BY** *string*
| **ENCODING** *encoding*
| { **ENCRYPTED KEY** '*key*' [**ALGORITHM** '*algorithm*'] | **NOT ENCRYPTED** }
| **ESCAPE CHARACTER** *character*
| **ESCAPES** { **ON** | **OFF** }
| **FORMAT** { **TEXT** | **BCP** }
| **HEXADECIMAL** { **ON** | **OFF** }
| **ORDER** { **ON** | **OFF** }
| **QUOTE** *string*
| **QUOTES** { **ON** | **OFF** }
| **ROW DELIMITED BY** *string*

encoding : *string*

参数

- **TO 子句** 要将数据卸载到的文件的名称。*filename* 路径是相对于数据库服务器启动目录的路径。如果该文件不存在，则会进行创建。如果该文件已存在，则除非另外指定了 APPEND ON，否则该文件会被覆盖。
- **INTO FILE 子句** 语义上等效于 TO *filename*。
- **INTO CLIENT FILE 子句** 客户端计算机上用于存放所卸载数据的文件。如果该文件不存在，则会进行创建。如果该文件已存在，则除非另外指定了 APPEND ON，否则该文件会被覆盖。将相对于客户端应用程序的当前工作目录来解析此路径。

  要使用 SQL Remote 将数据卸载到客户端计算机，请参见“PASSTHROUGH 语句 [SQL Remote]”一节第 656 页。
- **INTO VARIABLE 子句** 要将数据卸载到的变量。该变量必须已存在，且必须为 CHAR、NCHAR 或 BINARY 类型。APPEND 选项可以使卸载的数据附加到变量的当前内容。
- **APPEND 子句** APPEND 为 ON 时，卸载的数据将附加到指定文件的末尾。APPEND 为 OFF 时，卸载的数据将替换指定文件的内容。缺省情况下，APPEND 为 OFF。如果指定了 COMPRESSED 或 ENCRYPTED 子句，则不能指定此子句，如果要附加的文件已进行压缩或加密，则也不能使用此子句。
- **BYTE ORDER MARK 子句** 此子句用于指定字节顺序标记 (BOM) 是否出现在编码中。缺省情况下，此选项为 ON，只要卸载的目标文件为本地文件或客户端文件。BYTE ORDER MARK 选项为 ON 时，UTF-8 和 UTF-16 数据包含 BOM。如果 BYTE ORDER MARK 为 OFF，则不卸载 BOM。
- **COMPRESSED 子句** 指定是否压缩数据。缺省值为 NOT COMPRESSED。如果想要附加到该数据 (APPEND ON)，则不能对其进行压缩。

  如果要附加到的文件已进行压缩，则必须指定 COMPRESSED 子句。
- **DELIMITED BY 子句** 用于列间的字符串。缺省列分隔符是逗号。通过提供字符串可指定其它列分隔符。只有字符串的第一个字节（字符）用作分隔符。
- **ENCODING 子句** 所有的数据库数据都从数据库字符编码转换为指定的字符编码。未指定 ENCODING 时，将使用数据库的字符编码且不执行转换。

  有关如何获得 SQL Anywhere 支持的编码列表的信息，请参见“支持的字符集”一节 《SQL Anywhere 服务器 - 数据库管理》。

  如果在卸载操作期间发生转换错误，将根据 on_charset_conversion_failure 选项的设置进行报告。请参见“on_charset_conversion_failure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  以下示例使用 UTF-8 字符编码卸载数据：

  ```
  UNLOAD TABLE mytable TO 'mytable_data_in_utf8.dat' ENCODING 'UTF-8';
  ```

  指定 BYTE ORDER 子句，使数据中包含字节顺序标记。
- **ENCRYPTED 子句** 指定是否对数据加密。如果指定 NOT ENCRYPTED（缺省值），则不对数据加密。如果指定带密钥但无算法的 ENCRYPTED KEY，则使用 AES128 和指定的密钥对数

据加密。如果指定带密钥和算法的 ENCRYPTED KEY，则使用指定的密钥和算法对数据加密。算法可以是 CREATE DATABASE 语句所接受的任何算法。您不能指定简单加密。请参见“CREATE DATABASE 语句”一节第 413 页。

如果想要附加到该数据 (APPEND ON)，则不能对其进行加密。

如果要附加到的文件已进行加密，则必须指定 ENCRYPTED 子句。

- **ESCAPES 子句** 在 ESCAPES 设置为 ON（缺省值）的情况下，数据库服务器会识别反斜线字符后面的字符并将其解释为特殊字符。换行符可以作为组合 \n 包含在数据中，其它字符可以作为十六进制 ASCII 代码（如使用 \x09 代替制表符）包含在数据中。两个连续的反斜线字符 (\ \ ) 被解释为单个反斜线。任何除 n、x、X 或 \ 以外的字符及前面的反斜线都被解释为两个单独的字符。例如，\q 插入反斜线和字母 q。
- **FORMAT 子句** 以 TEXT 格式或 BCP 输出格式输出数据。如果选择 TEXT，则假定输入行为文本字符，每个输入行占一行，值由列分隔符字符串分隔。选择 BCP 会允许导入 Adaptive Server Enterprise 生成的包含 BLOB 的 BCP 输出文件。
- **HEXADECIMAL 子句** 缺省情况下，HEXADECIMAL 为 ON。二进制列值以 0x*nnnnnn*... 格式写入，其中 0x 是零后跟一个 x，每个 *n* 都是一个十六进制数。处理多字节字符集时，使用 HEXADECIMAL ON 很重要。

  HEXADECIMAL 子句只能与 FORMAT TEXT 子句配合使用。
- **ORDER 子句** 如果 ORDER 设置为 ON（缺省值），则导出的数据按聚簇索引（如果有）排序。如果没有聚簇索引，则导出的数据按主键值排列。如果 ORDER 设置为 OFF，则数据导出顺序与不使用 ORDER BY 子句从表中选择数据时的顺序相同。ORDER 设置为 ON 时，导出速度较慢。不过，由于索引步骤的简单性，使用 LOAD TABLE 语句时重装速度较快。

  对于 UNLOAD *select-statement*，ORDER 子句会被忽略。但仍然可以在 SELECT 语句中指定 ORDER BY 子句来对数据进行排序。
- **QUOTE 子句** QUOTE 子句仅适用于 TEXT 数据；*string* 放置于字符串值的两边。缺省值为单引号（撇号）。
- **QUOTES 子句** QUOTES 设置为 ON（缺省值）时，所有导出的字符串都用单引号括起来。
- **ROW DELIMITED BY 子句** 此子句可用于指定表示记录结束的字符串。缺省的分隔符字符串是换行符 (\n)。然而，它可以是最大长度为 255 个字节的任何字符串；例如，`... ROW DELIMITED BY '###' ...`。相同的格式要求同样适用于其它 SQL 字符串。如果您要指定以制表符分隔的值，可使用制表符 (9) 的十六进制转义序列，`... ROW DELIMITED BY '\x09' ...`。如果分隔符字符串包含 \n，它将与 \r\n 或 \n 匹配。

注释

UNLOAD 语句允许将来自 SELECT 语句的数据导出到以逗号分隔的文件中。除非 SELECT 语句包含 ORDER BY 子句，否则结果集不排序。

UNLOAD TABLE 语句允许从数据库表或实例化视图到文件的高效批量导出。UNLOAD TABLE 语句比 Interactive SQL 的 OUTPUT 语句效率更高，并且可从任何客户端应用程序调用。

数据库服务器或客户端应用程序（分别取决于指定的是 TO FILE 还是 INTO CLIENT FILE），必须具有写入到指定文件的操作系统权限。

对于 UNLOAD TABLE，当卸载具有二进制数据类型的表列时，UNLOAD TABLE 以 \x*nnnn* 格式写入十六进制字符串，其中 *n* 是一个十六进制数。对于 UNLOAD *select-statement*，当卸载具有二进制数据类型的结果集列时，UNLOAD 以 *\0xnnnn* 格式写入十六进制字符串，其中 *n* 是一个十六进制数。

当卸载并重装有代理表的数据库时，您必须创建一个外部登录来将本地用户映射到远程用户，即使该用户在本地和远程数据库中的口令相同也是如此。如果您没有外部登录，那么重装有可能因为您无法连接到远程服务器而失败。请参见“使用外部登录”一节 《SQL Anywhere 服务器 - SQL 的用法》。

当卸载到变量 (INTO VARIABLE) 时，输出会按如下方式转换为一个字符集：

1. 使用 ENCODING 子句中指定的字符集。
2. 如果未指定 ENCODING 子句，若变量为 NCHAR 型，则使用数据库 NCHAR 字符集；否则使用数据库 CHAR 字符集。

此外，如果变量为 CHAR 型，所选编码必须与数据库 CHAR 编码匹配；如果变量为 NCHAR 型，所选编码必须与数据库 NCHAR 编码匹配；BINARY 变量可以使用任何编码

如果选择对卸载数据进行压缩和加密，则会首先进行压缩。

UNLOAD TABLE 会在整个表或实例化视图上放置一个独占锁。

权限

卸载到变量时，不需要任何权限（除访问数据源所需的常规权限外）。

执行 UNLOAD 语句所需的权限取决于数据库服务器的 -gl 选项，如下所示：

- 如果 -gl 选项设置为 ALL，则必须对 UNLOAD 语句中引用的表具有 SELECT 权限。
- 如果 -gl 选项设置为 DBA，则您必须具有 DBA 权限。
- 如果 -gl 选项设置为 NONE，则不允许执行 UNLOAD。

请参见“-gl 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

当写入到客户端计算机的文件中时：

- 需要 WRITECLIENTFILE 权限。请参见“WRITECLIENTFILE 特权”一节 《SQL Anywhere 服务器 - 数据库管理》。
- 需要对写入的目录具有写权限。
- 必须启用 allow_write_client_file 数据库选项。请参见“allow_write_client_file 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- 必须启用 write_client_file 受保护的功能。请参见“-sf 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

副作用

无。查询在当前隔离级别上执行。

另请参见

- “CREATE DATABASE 语句”一节第 413 页
- “LOAD TABLE 语句”一节第 626 页
- “使用聚簇索引”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “OUTPUT 语句 [Interactive SQL]”一节第 650 页
- “使用 UNLOAD 语句导出数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “使用 UNLOAD 语句导出数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “访问客户端计算机上的数据”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “导入和导出数据” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例将 Products 表中的内容卸载到 UTF-8 编码文件 *productsT.dat* 中：

```
UNLOAD TABLE Products TO 'productsT.dat' ENCODING 'UTF-8';
```

以下示例创建一个名为 @myProducts 的变量，然后将 Products.Name 列卸载到该变量：

```
CREATE VARIABLE @myProducts LONG VARCHAR;
UNLOAD SELECT NAME FROM Products INTO VARIABLE @myProducts;
```

UPDATE 语句

此语句用于修改数据库表中的现有行。

语法 1

UPDATE [*row-limitation*] *table-list*]
SET *set-item*, ...
[**FROM** *table-expression* [,...]]
[**WHERE** *search-condition*]
[**ORDER BY** *expression* [**ASC** | **DESC**] , ...]
[**OPTION(** *query-hint*, ... **)**]

table-list :
table-name [,...]

table-name :
[*owner*.]*table-name* [[**AS**] *correlation-name*]
| [*owner*.]*view-name* [[**AS**] *correlation-name*]
| *derived-table*

derived-table :
(*select-statement* **)**
[**AS**] *correlation-name* [**(** *column-name* [,...] **)**]

table-expression：
完整的表表达式可包含连接。请参见“FROM 子句”一节第 582 页。

语法 2

UPDATE *table-name*
SET *set-item*, ...
VERIFY (*column-name*, ... **) VALUES (** *expression*, ... **)**
[**WHERE** *search-condition*]
[**ORDER BY** *expression* [**ASC** | **DESC**], ...]
[**OPTION(** *query-hint*, ... **)**]

语法 3

UPDATE [*owner.*]*table-name*
PUBLICATION *publication*
{ **SUBSCRIBE BY** *expression*
| **OLD SUBSCRIBE BY** *expression* **NEW SUBSCRIBE BY** *expression*
}
WHERE *search-condition*

row-limitation :
FIRST
| **TOP** *n* [**START AT** *m*]

set-item :
[*correlation-name.*]*column-name* **=** { *expression* | **DEFAULT** }
| [*owner-name.*]*table-name.column-name* **=** { *expression* | **DEFAULT** }
| **@***variable-name* **=** *expression*

query-hint :
MATERIALIZED VIEW OPTIMIZATION *option-value*
| **FORCE OPTIMIZATION**
| **FORCE NO OPTIMIZATION**
| *option-name* = *option-value*

table-name :
[*owner.*]*base-table-name*
| *temporary-table-name*
| *derived-table-name*
| [*owner.*]*view-name*

option-name : *identifier*

option-value : *hostvar* (indicator allowed), *string*, *identifier*, or *number*

参数

- **UPDATE 子句** 对于语法 1，*table-list* 可包括临时表、派生表或视图。可对视图和派生表进行更新，除非它们是不可更新的。对于语法 2 和语法 3，*table-name* 必须为基表。

  仅当定义视图的查询说明为可更新时，才能对视图执行 UPDATE 语句。有关识别固有不可更新的视图的详细信息，请参见“使用视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **row-limitation 子句** 行限制子句允许您只返回满足 WHERE 子句的行的子集。TOP 和 START AT 值可以是主机变量、整型常量或整型变量。TOP 值必须大于等于 0。START AT 值必须大于 0。通常，在指定这些子句时也指定 ORDER BY 子句，这样可以用一种有意义的方式对这些行进行排序。请参见“显式限制查询返回的行数”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- **SET 子句** set 子句指定列，并指定如何更改值。

  可通过下列格式使用 SET 子句将列设置为计算列值：

  ```
  SET column-name = expression, ...
  ```

  每个指定的列被设置为等号右侧表达式的值。对*表达式*没有任何限制。如果表达式为 *column-name*，则使用旧值。

  如果某列定义了缺省值，则可以使用 SET 子句将该列设置为其缺省值。有关这一用法的示例，请参见示例部分。

  还可以通过下列格式使用 SET 子句为变量赋值：

  ```
  SET @variable-name = expression, ...
  ```

  为变量赋值时，变量必须已经声明，而且变量名必须以 "at" 符号 (@) 开头。变量和列赋值可以混合在一起，并且可以使用任意数字。如果 SET 列表中赋值号左边的名称不仅与变量名匹配，还与更新表中的某列匹配，此语句将更新该列。

  以下是部分 UPDATE 语句的示例。它不仅更新表，还为变量赋值：

  ```
  UPDATE T SET @var = expression1, col1 = expression2
  WHERE...
  ```

  这等效于：

  ```
  SELECT @var = expression1
  FROM T
  WHERE... ;
  UPDATE T SET col1 = expression2
  WHERE...
  ```

- **FROM 子句** 如果存在 FROM 子句，WHERE 子句将限定 FROM 子句的行。

  FROM *table-expression* 子句允许根据连接来更新表。*table-expression* 可以包含任意复杂表的表达式，如 KEY 和 NATURAL 连接。有关 FROM 子句和连接的完整说明，请参见“FROM 子句”一节第 582 页。

  如果使用 FROM 子句，在语句的两个部分必须以同样的方式限定表名。如果在一个位置使用了相关名，则在另一个位置也必须使用该相关名，否则将发生错误。

  下面的语句阐释 UPDATE 语句（有两个使用相关名的 FROM 子句）的表名中潜在的不明确性：

  ```
  UPDATE
  FROM table_1
  FROM table_1 AS alias_1, table_2 AS alias_2
  WHERE ...
  ```

  表 table_1 在第一个 FROM 子句中没有相关名标识，而在第二个 FROM 子句中具有相关名标识。在本例中，第一个子句中的 table_1 在第二个子句中用 alias_1 标识—此语句中只有一个 table_1 实例。这是一般规则的特例，即同一语句中，一个表一次用相关名标识另一次没有相关名，则将视为出现表的两个实例。

  但以下示例中，第二个 FROM 子句中有两个 table_1 的实例。此语句因语法错误而失败，因为不明确第二个 FROM 子句中的哪个 table_1 实例与第一个 FROM 子句中的第一个 table_1 实例匹配。

```
UPDATE
FROM table_1
FROM table_1 AS alias_1, table_1 AS alias_2
WHERE ...
```

仅当 ansi_update_constraints 设置为 Off 时，才能使用此子句。请参见“ansi_update_constraints 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关连接的完整说明，请参见“连接：从多个表检索数据” 《SQL Anywhere 服务器 - SQL 的用法》。

有关详细信息，请参见“FROM 子句”一节第 582 页。

- **WHERE 子句** 如果指定 WHERE 子句，则只更新满足搜索条件的行。如果未指定 WHERE 子句，则更新所有行。
- **ORDER BY 子句** 通常情况下，行以什么顺序更新并不重要。但与 FIRST 或 TOP 子句联用时，顺序就非常重要。

  不能在 ORDER BY 子句中使用列序号。

  除非将 ansi_update_constraints 选项设置为 Off，否则不得更新 ORDER BY 子句中出现的列。请参见“ansi_update_constraints 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **OPTION 子句** 此子句用于指定执行语句时的提示。支持以下提示：
  - MATERIALIZED VIEW OPTIMIZATION *option-value*
  - FORCE OPTIMIZATION
  - FORCE NO OPTIMIZATION
  - *option-name* = *option-value*

注释

插入到表中的字符串始终以它们输入时采用的大小写进行保存，而不论数据库是否区分大小写。用字符串 Street 更新的 CHAR 数据类型列在数据库中保存的形式始终是 S 为大写，其它字母均为小写。SELECT 语句返回的字符串为 Street。但是，如果数据库不区分大小写，所有比较都会将 Street 与 street、STREET 等不同大小写形式的字符串视为相同。而且，如果单列主键已经包含 Street 条目，则会拒绝对 street 执行 INSERT 操作，因为它会导致主键不唯一。

如果新值与旧值没有差别，则不更改数据。但是，每次行上发生 UPDATE（更新）时，无论新值是否与旧值相同，都触发 BEFORE UPDATE 触发器。AFTER UPDATE 触发器只在新值与旧值不同时才触发。

UPDATE 语句的语法 1 修改一个或多个表中的行值。语法 2 和语法 3 仅适用于 SQL Remote。

语法 2 仅用于 SQL Remote 中由消息代理执行单个表的单行更新。VERIFY 子句包含在要更新的行中应存在的值集。如果值不匹配，则继续 UPDATE 操作之前会触发所有 RESOLVE UPDATE 触发器。如果只是 VERIFY 子句不匹配，UPDATE 不会失败。

UPDATE 语句的语法 3 用于实现特定的 SQL Remote 功能，并且在 BEFORE 触发器内使用。每当列表更改时，它都提供 SUBSCRIBE BY 值的完整列表。它放在 SQL Remote 触发器中，以便数据库服务器能够计算 SUBSCRIBE BY 值的当前列表。这两个列表都放置在事务日志中。

消息代理使用这两个列表来确保将行移动到任何没有行且现在需要行的远程数据库。消息代理还从任何含有行且不再需要行的远程数据库中删除行。UPDATE 语句不影响含有行且仍然需要行的远程数据库。

对于在 SUBSCRIBE BY 子句中用子查询创建的发布，必须编写包含 UPDATE 语句的语法 3 的触发器，以确保各行保留在它们正确的预订中。

UPDATE 语句的语法 3 允许显式指定旧 SUBSCRIBE BY 列表和新 SUBSCRIBE BY 列表，从而使 SQL Remote 触发器的效率更高。如果缺少这些列表，数据库服务器将计算发布定义中的旧 SUBSCRIBE BY 列表。由于新 SUBSCRIBE BY 列表与旧 SUBSCRIBE BY 列表通常只是稍微有点差别，因此可能会执行两次旧列表的计算。通过既指定旧列表又指定新列表，可以避免这种额外的工作。

SUBSCRIBE BY 表达式是一个值或一个子查询。

UPDATE 语句的语法 3 在事务日志中创建项，但不更改数据库表。

使用 UPDATE 语句更新大量数据时，也会更新列的统计信息。

权限

必须具有所修改列的 UPDATE 权限。

副作用

更新列统计信息。

另请参见

- “DELETE 语句”一节第 530 页
- “INSERT 语句”一节第 616 页
- “FROM 子句”一节第 582 页
- “连接：从多个表检索数据” 《SQL Anywhere 服务器 - SQL 的用法》
- “UPDATE（定位）语句 [ESQL] [SP]”一节第 740 页
- “更新过程中的锁定”一节 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** 除 FROM 和 ORDER BY 子句是服务商扩充外，语法 1 是核心特性。语法 2 和语法 3 是仅用于 SQL Remote 的服务商扩充。

  要强制实现与 SQL/2003 的兼容，请确保将 ansi_update_constraints 选项设置为 Strict。请参见“ansi_update_constraints 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》。

示例

此示例使用示例数据库将雇员 Philip Chin（雇员 129）从销售部转移到市场部。

```
UPDATE Employees
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

利用示例数据库，以下示例通过从 ID 中减去 2000 来对所有现有销售订单重新编号。

```
UPDATE SalesOrders AS orders
SET orders.ID = orders.ID - 2000
ORDER BY orders.ID ASC;
```

仅当使用 ON UPDATE CASCADE 操作定义 SalesOrderItems 表（引用主键 SalesOrders.ID 的表）的外键时，才可能进行此更新。SalesOrderItems 表同样也会更新。

有关外键属性的详细信息，请参见“ALTER TABLE 语句”一节第 373 页和“CREATE TABLE 语句”一节第 497 页。

此示例使用示例数据库，以隔离级别 2 而不是以数据库的当前隔离级别设置更改产品的价格。

```
UPDATE Products
SET UnitPrice = 7.00
WHERE ID = 501
OPTION( isolation_level = 2 );
```

以下示例显示如何更新表以将某列设置为其缺省值。以下示例将创建一个表 MyTable，用数据填充该表，然后执行 UPDATE 语句，指定 SET 子句将列值更改为其缺省值。

```
CREATE TABLE MyTable(
   PK INT PRIMARY KEY  DEFAULT AUTOINCREMENT,
   TableName CHAR(128) NOT NULL,
   TableNameLen INT DEFAULT 20,
   LastUser CHAR(10) DEFAULT last user,
   LastTime TIMESTAMP DEFAULT TIMESTAMP,
   LastTimestamp TIMESTAMP DEFAULT @@dbts );

INSERT INTO MyTable WITH AUTO NAME
   SELECT
      LENGTH(t.table_name) AS TableNameLen,
      t.table_name AS TableName
   FROM SYS.SYSTAB t
   WHERE table_id<=10;

UPDATE MyTable SET  LastTime = DEFAULT, LastTimestamp = DEFAULT
   WHERE TableName LIKE '%sys%';
```

UPDATE（定位）语句 [ESQL] [SP]

此语句用于修改游标当前位置处的数据。

语法 1

UPDATE WHERE CURRENT OF *cursor-name*
{ **USING DESCRIPTOR** *sqlda-name* | **FROM** *hostvar-list* }

语法 2

UPDATE *update-table*, ...
SET *set-item*, ...
WHERE CURRENT OF *cursor-name*

hostvar-list : *indicator variables allowed*

update-table :
[*owner-name.*]*object-name* [[**AS**] *correlation-name*]

set-item :
[*correlation-name.*]*column-name* **=** { *expression* | **DEFAULT** }
| [*owner-name.*]*object-name.column-name* **=** { *expression* | **DEFAULT** }

object-name : *identifier* (a table or view name)

sqlda-name : *identifier*

参数

- **USING DESCRIPTOR 子句** 为变量赋值时，变量必须已经声明，而且变量名必须以 "at" 符号 (@) 开头。变量和列赋值可以混合在一起，并且可以使用任意数字。如果 SET 列表中赋值号左边的名称不仅与变量名匹配，还与更新表中的某列匹配，此语句将更新该列。
- **SET 子句** *set-item* 中引用的列必须位于更新的表或视图中。它们不能引用别名或其它表或视图中的列。如果要更新的表或视图在游标说明中被赋予了一个相关名，那么在 SET 子句中必须使用此相关名。

  每个 *set-item* 都与一个 *update-table* 关联，并会对游标查询中匹配表的相应列进行修改。*expression* 将引用 UPDATE 列表中标识的表的列，还可能使用常量、变量、查询选择列表中的表达式或以上各元素的组合（例如，使用 +、-、...、COALESCE、IF 等运算符）。*expression* 不能引用游标的查询中表达式的别名，也不能引用未在 UPDATE 列表中出现的游标的查询中其它表的列。子选择、子查询谓语和集合函数无法在 *set-items* 中使用。

  每个 *update-table* 都按照以下方式与游标查询中的表进行匹配：

  - 如果指定了相关名，则它将与游标的查询中具有相同 *table-or-view-name* 和相同 *correlation-name* 的表相匹配。
  - 否则，如果游标的查询中表具有相同 *table-or-view-name* 而没有指定的相关名，或具有与 *table-or-view-name* 相同的相关名时，则更新表将与游标的查询中的此表相匹配。
  - 否则，如果游标的查询中的单个表具有与更新表相同的 *table-or-view-name*，则更新表将与游标的查询中的此表相匹配。

  如果某列定义了缺省值，则可以使用 SET 子句将该列设置为其缺省值。有关这一用法的示例，请参见“UPDATE 语句”一节第 735 页的示例部分。

注释

这种形式的 UPDATE 语句更新指定游标的当前行。当前行被定义为从游标中成功读取的最后一行，对游标的最后一项操作不能是定位的 DELETE 语句。

对于语法 1，SQLDA 中的列或主机变量列表中的值与指定游标返回的列一一对应。如果 SQLDA 中的 sqldata 指针为空指针，则不更新相应的选择列表项。

在语法 2 中，在指定查询的当前行中，请求的列被设置为该行的指定值。这些列不必位于指定的打开游标的选择列表中。这种格式是可以准备的。

同样，为变量赋值时，变量必须已经声明，而且变量名必须以 "at" 符号 (@) 开头。变量和列赋值可以混合在一起，并且可以使用任意数字。如果 SET 列表中赋值号左边的名称不仅与变量名匹配，还与更新表中的某列匹配，此语句将更新该列。

USING DESCRIPTOR、FROM *hostvar-list* 和 *hostvar* 格式仅用于嵌入式 SQL。

权限

必须有所修改的列的 UPDATE 权限。

副作用

无。

另请参见

- “INSERT 语句”一节第 616 页
- “LOAD TABLE 语句”一节第 626 页
- “MERGE 语句”一节第 638 页
- “DELETE 语句”一节第 530 页
- “DELETE（定位）语句 [ESQL] [SP]”一节第 533 页
- “UPDATE 语句”一节第 735 页

标准和兼容性

- **SQL/2003** 核心特性。如果 ansi_update_constraints 选项设置为 Off，则可更新的游标的范围可能包含服务商扩充。
- **Sybase** Open Client/Open Server 支持使用嵌入式 SQL，SQL Anywhere 支持使用过程和触发器。

示例

以下是 UPDATE 语句 WHERE CURRENT OF 游标的示例：

```
UPDATE Employees
SET Surname = 'Jones'
WHERE CURRENT OF emp_cursor;
```

UPDATE 语句 [SQL Remote]

此语句用于修改数据库中的数据。

语法 1

UPDATE *table-list*
SET *column-name* = *expression*, ...
[**VERIFY (** *column-name*, ... **) VALUES (** *expression*, ... **)**]
[**WHERE** *search-condition*]
[**ORDER BY** *expression* [**ASC** | **DESC**], ...]

语法 2

UPDATE *table-name*
PUBLICATION *publication-name*
{ **SUBSCRIBE BY** *subscription-expression* |
  OLD SUBSCRIBE BY *old-subscription-expression*
  NEW SUBSCRIBE BY *new-subscription-expression* }
WHERE *search-condition*

expression: *value* | *subquery*

参数

- **table-name** *table-name* 指示在远程数据库中必须修改的表。

- **publication-name** *publication-name* 指示必须为其更改预订的发布。
- **subscription-expression** SQL Remote 使用 *subscription-expression* 的值确定行的新接收者和现有接收者。*subscription-expression* 可以是一个值或一个子查询。或者，也可以提供 OLD 和 NEW 预定表达式。
- **WHERE** WHERE 子句指定要在预订的数据库间传送哪些行。

注释

UPDATE 语句用于修改一个或多个表中的行。每个指定的列被设置为等号右侧表达式的值。对 *expression* 没有任何限制。表达式中甚至可以使用 *column-name*—此时将使用旧值。

如果未指定 WHERE 子句，则更新所有行。如果指定了 WHERE 子句，则仅更新满足搜索条件的行。

通常情况下，以什么顺序更新行并不重要。但是，与 NUMBER(*) 函数一起使用时，排序可用于按指定的顺序在行中添加递增的编号。此外，如果想要执行类似向表的主键值中添加 1 这样的操作，必须按主键的降序操作，以便避免从操作中得到重复的主键。

如果定义视图的 SELECT 语句不包含 GROUP BY 子句、集合函数或不涉及 UNION 子句，则可以更新视图。

插入到表中的字符串始终以它们输入时的大小写形式存储，无论数据库是否区分大小写。因此，用字符串 Value 更新的字符数据类型列在数据库中保存的形式始终是 V 为大写，其它字母均为小写。SELECT 语句返回的字符串为 Value。但是，如果数据库不区分大小写，所有比较都会将 Value 与 value、VALUE 等不同大小写形式的字符串视为相同。而且，如果单列主键已经包含 Value 条目，则会拒绝对 value 执行 INSERT，因为它会导致主键不唯一。

可选的 FROM 子句允许根据连接更新表。如果存在 FROM 子句，WHERE 子句将限定 FROM 子句的行。仅更新表列表中紧跟在 UPDATE 关键字后面的数据。

如果使用 FROM 子句，应以同样的方式在语句的两个部分限定要更新的表名，这点很重要。如果在一个位置使用了相关名，则在另一个位置也必须使用该相关名，否则将发生错误。

语法 1 和语法 2 仅适用于 SQL Remote。

不带 OLD 和 NEW SUBSCRIBE BY 表达式的语法 2 必须用于 BEFORE 触发器中。

带 OLD 和 NEW SUBSCRIBE BY 表达式的语法 2 可在任何地方使用。

语法 1 仅用于 SQL Remote 中由消息代理执行的单行更新。VERIFY 子句包含在要更新的行中应存在的值集。如果值不匹配，则继续 UPDATE 操作之前会触发所有 RESOLVE UPDATE 触发器。如果只是 VERIFY 子句不匹配，UPDATE 不会失败。当指定 VERIFY 子句时，一次只能更新一个表。

语法 2 仅用于 SQL Remote。如果不使用 OLD 和 NEW 表达式，它必须在 BEFORE 触发器内使用，以便能够访问相关的值。其目的是每当列表更改时都提供 SUBSCRIBE BY 值的完整列表。它放在 SQL Remote 触发器中，以便数据库服务器能够计算 SUBSCRIBE BY 值的当前列表。这两个列表都放置在事务日志中。

消息代理使用这两个列表来确保将行移动到任何没有行且现在需要行的远程数据库。消息代理还从任何含有行且不再需要行的远程数据库中删除行。UPDATE 语句不影响含有行且仍然需要行的远程数据库。

UPDATE 语句的语法 2 允许显式指定旧 SUBSCRIBE BY 列表和新 SUBSCRIBE BY 列表，从而使 SQL Remote 触发器的效率更高。如果缺少这些列表，数据库服务器将计算发布定义中的旧 SUBSCRIBE BY 列表。由于新 SUBSCRIBE BY 列表与旧 SUBSCRIBE BY 列表通常只是稍微有点差别，因此可能会执行两次旧列表的计算。通过既指定旧列表又指定新列表，可以避免这种额外的工作。

当表达式用相同的预订更新同一触发器内的许多表时，OLD 和 NEW SUBSCRIBE BY 语法尤其有用。这样可以动态地提高性能。

SUBSCRIBE BY 表达式是一个值或一个子查询。

UPDATE 语句的语法 2 用于实现特定的 SQL Remote 功能，并且在 BEFORE 触发器内使用。

对于在预订表达式中用子查询创建的发布，必须编写包含 UPDATE 语句语法 2 的触发器，以便确保行都保留在它们正确的预订中。

有关此功能的完整说明，请参见“使用 BEFORE UDPATE 触发器”一节 《SQL Remote》。

UPDATE 语句的语法 2 在事务日志中创建项，但不更改数据库表。

权限

必须具有所修改列的 UPDATE 权限。

副作用

无。

另请参见

- “使用 BEFORE UDPATE 触发器”一节 《SQL Remote》

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

以下示例将雇员 Philip Chin（雇员 129）从销售部转移到市场部。

```
UPDATE Employees
VERIFY( DepartmentID ) VALUES( 300 )
SET DepartmentID = 400
WHERE EmployeeID = 129;
```

VALIDATE 语句

此语句用于校验当前数据库，或校验当前数据库中的单个表或实例化视图。

语法 1 - 校验表和实例化视图。

VALIDATE {
TABLE [*owner.*]*table-name*
  | **MATERIALIZED VIEW** [*owner.*]*materialized-view-name* }
[**WITH EXPRESS CHECK**]

语法 2 - 校验数据库

VALIDATE { **CHECKSUM** | **DATABASE** }

语法 3 - 校验索引

VALIDATE {
INDEX *index-name*
| [**INDEX**] **FOREIGN KEY** *role-name*
| [**INDEX**] **PRIMARY KEY** }
ON [*owner.*]*object-name*
}

object-name :
table-name
| *materialized-view-name*

参数

- **WITH EXPRESS CHECK** 除缺省检查外，还检查表或实例化视图中的行数与索引中的条目数是否匹配。此选项不为每个行执行单独的索引查找，也不执行校验和校验。当使用较小的高速缓存校验大数据库时，此选项可以显著提高校验性能。

注释

表的校验包括校验和校验，以及表中的行数与该表关联的每个索引中的行数相匹配的校验。当指定 WITH EXPRESS CHECK 时，将不执行校验和校验。

VALIDATE DATABASE 语句校验数据库中的所有表页是否属于正确的对象。VALIDATE DATABASE 还执行校验和校验，但不校验索引或检查数据的正确性。

使用 VALIDATE CHECKSUM 语句在数据库上执行校验和校验。VALIDATE CHECKSUM 语句可确保数据库页没有在磁盘上被修改。创建启用了校验和的数据库时，在将每一数据库页写入磁盘前会为其计算校验和。VALIDATE CHECKSUM 从磁盘读取每个数据库页，然后为每页计算校验和。如果为页面计算出的校验和与存储的该页面的校验和不匹配，就会发生错误，并在数据库服务器消息窗口中显示关于该无效页的信息。VALIDATE CHECKSUM 语句在禁用了校验和的数据库上也很有用，因为重要数据库页仍然包括校验和。

VALIDATE CHECKSUM 语句用于在表或实例化视图上校验索引（包括索引统计信息）。VALIDATE INDEX 语句可确保索引中引用的各行确实存在。对于外键索引，还要确保主表中存在相应的行。这种检查补充了由 VALIDATE TABLE 语句执行的有效性检查。VALIDATE INDEX 语句还校验为指定索引报告的统计信息是否准确。如果不准确，将产生一个错误。

> **小心**
> 应在没有任何连接对数据库进行更改时对表或整个数据库进行校验；否则，可能会报告错误，指出某种形式的数据库损坏，而实际上并没有任何损坏。

权限

必须具有 DBA 或 VALIDATE 权限。

副作用

无。

另请参见

- “校验实用程序 (dbvalid)”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_validate 系统过程”一节第 912 页
- “校验数据库” 《SQL Anywhere 服务器 - 数据库管理》
- “CREATE DATABASE 语句”一节第 413 页
- “CREATE INDEX 语句”一节第 449 页

标准和兼容性

- **SQL/2003** 服务商扩充。

WAITFOR 语句

此语句用于将当前连接的处理推迟指定的时间长度，或推迟到特定的时间。

语法

WAITFOR {
DELAY *time*
| **TIME** *time* }
[**CHECK EVERY** *integer*]
[**AFTER MESSAGE BREAK**]

time : *string*

参数

- **DELAY 子句** 如果使用 DELAY，处理将暂停特定的时间长度。
- **TIME 子句** 如果指定 TIME，处理将一直暂停到数据库服务器时间到达指定的时间。如果当前服务器时间大于指定的时间，则处理将一直暂停到第二天的这个时间。
- **CHECK EVERY 子句** 此可选子句控制 WAITFOR 语句唤醒的频率。缺省情况下，它每 5 秒钟唤醒一次。该值以毫秒为单位，最小值为 250 毫秒。
- **AFTER MESSAGE BREAK 子句** WAITFOR 语句可用于等待来自另一连接的消息。大多数情况下，当接收到消息时，将消息转发到执行 WAITFOR 语句的应用程序，同时 WAITFOR 语句继续等待。如果指定了 AFTER MESSAGE BREAK 子句，当接收到来自另一连接的消息时，WAITFOR 语句完成。消息文本不会转发到应用程序，但是可以通过获取 MessageReceived 连接属性的值来进行访问。

  有关 MessageReceived 属性的详细信息，请参见“连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》。

注释

WAITFOR 语句定期唤醒（缺省情况下每 5 秒钟一次），以检查是否语句已取消或消息已接收。如果两种情况均未发生，则该语句继续等待。

WAITFOR 提供了以下语句的替代语句：

```
CALL java.lang.Thread.sleep( <time_to_wait_in_millisecs> );
```


因为调度事件在其自身的连接中执行，所以使用调度事件通常比使用 WAITFOR TIME 要好。

权限

None

副作用

此语句的实现在等待时使用工作线程。这会消耗由 -gn 数据库选项指定的线程（缺省值为 20 个线程）之一。

另请参见

- “CREATE EVENT 语句”一节第 429 页

标准和兼容性

- **SQL/2003** 服务商扩充。

示例

下面的示例等待 3 秒钟：

```
WAITFOR DELAY '00:00:03';
```

以下示例等待 0.5 秒钟（500 毫秒）：

```
WAITFOR DELAY '00:00:00:500';
```

以下示例一直等到晚上 8 点：

```
WAITFOR TIME '20:00';
```

在以下示例中，当收到来自连接 2 的消息时，连接 1 的 WAITFOR 语句完成：

```
// connection 1:
BEGIN
  DECLARE msg LONG VARCHAR;
  LOOP  // forever
    WAITFOR DELAY '00:05:00' AFTER MESSAGE BREAK;
    SET msg = CONNECTION_PROPERTY('MessageReceived');
    IF msg != '' THEN
      MESSAGE 'Msg: ' || msg TO CONSOLE;
    END IF;
  END LOOP
END;
// connection 2:
MESSAGE 'here it is' FOR connection 1
```

WHENEVER 语句 [ESQL]

此语句用于指定嵌入式 SQL 程序中的错误处理。

语法

WHENEVER {
SQLERROR

| SQLWARNING
| NOTFOUND }
GOTO
label
| STOP
| CONTINUE
| { *C-code*; }

label : *identifier*

注释

WHENEVER 语句用于捕获数据库在处理 SQL 语句时遇到的错误、警告和异常情况。可将此语句放置在嵌入式 SQL 程序中的任何位置，它不生成任何代码。预处理器将在每个连续的 SQL 语句后生成代码。错误操作对 WHENEVER 语句源行中的所有嵌入式 SQL 语句一直保持有效，直到发生下一个具有相同错误条件的 WHENEVER 语句或者直到源文件结束。

基于源文件位置的错误
错误条件的生效依据是 C 语言源文件的位置，而不是语句的执行时间。

缺省操作为 CONTINUE。

请注意，提供此语句是为了在简单程序中方便使用。多数情况下，检查错误的最简单方法是直接检查 SQLCA (SQLCODE) 的 sqlcode 字段。这种情况下不使用 WHENEVER 语句。事实上，所有 WHENEVER 语句均使预处理器在每个语句后生成 *if (SQLCODE)* 测试。

权限

无。

副作用

无。

标准和兼容性

- **SQL/2003** 核心特性。

示例

以下是 WHENEVER 语句的示例：

```
EXEC SQL WHENEVER NOTFOUND GOTO done;
EXEC SQL WHENEVER SQLERROR
   {
      PrintError( &sqlca );
      return( FALSE );
   };
```

WHILE 语句 [T-SQL]

此语句用于重复执行语句或复合语句。

语法

WHILE *search-condition-statement*

注释

除非各语句被组织成位于关键字 BEGIN 和 END 之间的复合语句，否则 WHILE 条件只影响单个 SQL 语句的执行。

BREAK 语句和 CONTINUE 语句可用于控制复合语句中的语句执行。BREAK 语句终止循环，执行在标记循环结束的 END 关键字后重新开始。CONTINUE 语句使 WHILE 循环重新开始，跳过 CONTINUE 后面的所有语句。

权限

无。

副作用

无。

另请参见

- “LOOP 语句”一节第 637 页
- “CONTINUE 语句 [T-SQL]”一节第 413 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

下面的代码阐释了 WHILE 的用法：

```
WHILE ( SELECT AVG(UnitPrice) FROM Products ) < $30
BEGIN
   UPDATE Products
   SET UnitPrice = UnitPrice + 2
   IF ( SELECT MAX(UnitPrice) FROM Products ) > $50
      BREAK
END
```

如果最贵产品的价格高于 $50，BREAK 语句将中断 WHILE 循环。否则继续循环，直到平均价格高于或等于 $30。

WINDOW 子句

在 SELECT 语句中，WINDOW 子句用于定义全部或部分与窗口函数一起使用的窗口，例如 AVG 和 RANK。

语法

WINDOW *window-expression*, ...

window-expression : *new-window-name* **AS (** *window-spec* **)**

window-spec :
[*existing-window-name*]
[**PARTITION BY** *expression*, ...]
[**ORDER BY** *expression* [**ASC** | **DESC**], ...]
[{ **ROWS** | **RANGE** } { *window-frame-start* | *window-frame-between* }]

window-frame-start :
{ **UNBOUNDED PRECEDING**
| *unsigned-integer* **PRECEDING**
| **CURRENT ROW** }

window-frame-between :
BETWEEN *window-frame-bound1* **AND** *window-frame-bound2*

window-frame-bound :
window-frame-start
| **UNBOUNDED FOLLOWING**
| *unsigned-integer* **FOLLOWING**

参数

- **PARTITION BY 子句** PARTITION BY 子句将结果集组织到基于指定表达式的唯一值的逻辑组中。当此子句与窗口函数一起使用时，函数将独立地应用于每个分区。例如，如果 PARTITION BY 后跟列名，则将按该列中的不同值对结果集进行分区。

  如果忽略此子句，则会将整个结果集考虑为一个分区。

  PARTITION BY *expression* 不能是整数值。

- **ORDER BY 子句** ORDER BY 子句定义如何对结果集的每个分区中的行进行排序。通过指定 ASC 按升序排序（缺省值）或 DESC 按降序排序，可进一步控制顺序。

  ORDER BY *expression* 不能是整数值。

  如果省略此子句，SQL Anywhere 会以最有效的顺序返回行，而且结果集的外观会根据用户上次访问行的时间而有所不同。

- **ROWS 子句和 RANGE 子句** ROWS 或 RANGE 子句均可用于表示窗口的大小。窗口的大小可以是分区的一行、多行或所有行。可根据当前行中值的数据值偏移范围 (RANGE) 或根据当前行中的物理行偏移数 (ROWS) 表示窗口的大小。

  使用 RANGE 子句时，还必须指定 ORDER BY 子句，因为范围计算需要存储值。范围的 ORDER BY 子句必须包含一个表达式，且该表达式必须产生一个日期或数字值。

  如果不指定 ROWS 或 RANGE 子句，数据库服务器将根据是否存在 ORDER BY 子句来使用缺省窗口大小。有关缺省值的信息，请参见“定义窗口”一节 《SQL Anywhere 服务器 - SQL 的用法》。

  - **PRECEDING 子句** PRECEDING 子句用于使用当前行作为参照点来定义窗口的第一行。起始行可根据当前行之前的行数来表示。例如，`5 PRECEDING` 设置窗口从当前行之前的第五行开始。

    UNBOUNDED PRECEDING 用于将窗口中的第一行设置为分区中的第一行。

- **BETWEEN 子句** BETWEEN 子句用于使用当前行作为参照点来定义窗口的第一行和最后一行。第一行和最后一行可分别根据当前行之前和之后的行数来表示。例如，`BETWEEN 3 PRECEDING AND 5 FOLLOWING` 设置窗口从当前行之前的第三行开始，结束于当前行之后的第五行。

  BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING 用于将窗口中的第一行和最后一行分别设置为分区中的第一行和最后一行。这等同于未指定 ROW 或 RANGE 子句时的缺省行为。

- **FOLLOWING 子句** FOLLOWING 子句用于使用当前行作为参照点来定义窗口的最后一行。最后一行可根据当前行之后的行数来表示。

  UNBOUNDED FOLLOWING 用于将窗口中的最后一行设置为分区中的最后一行。

注释

在 SELECT 语句中，WINDOW 子句必须在 ORDER BY 子句之前显示。

除 LIST 函数外，全部的集合函数都可用作窗口函数。但秩集合函数（RANK、DENSE_RANK、PERCENT_RANK、CUME_DIST 和 ROW_NUMBER）需要 ORDER BY 子句，且不允许 WINDOW 子句或内置定义中包含 ROW 或 RANGE 子句。对于所有其它窗口功能，可使用任何子句。

有关定义和使用窗口以获得所需结果的详细信息，请参见“定义窗口”一节 《SQL Anywhere 服务器 - SQL 的用法》和“窗口定义：内置与 WINDOW 子句”一节 《SQL Anywhere 服务器 - SQL 的用法》。

另请参见

- “SELECT 语句”一节第 689 页
- “OLAP 支持” 《SQL Anywhere 服务器 - SQL 的用法》

标准和兼容性

- **SQL/2003** SQL/2003 特性 T611、T612。

示例

以下示例返回雇员的薪水以及所选状态中全部雇员的平均薪水。结果先按照状态排序，然后按照姓排序。

```
SELECT EmployeeID, Surname, Salary, State,
  AVG( Salary ) OVER SalaryWindow
FROM Employees
WINDOW SalaryWindow AS ( PARTITION BY State )
ORDER BY State, Surname;
```

WRITETEXT 语句 [T-SQL]

允许对现有文本或图像列进行交互更新但不进行记录日志。

语法

WRITETEXT *table-name.column-name*
text-pointer [**WITH LOG**] *data*

注释

更新现有文本或图像值。除非提供 WITH LOG 选项，否则事务日志中不记录更新。对视图不能执行 WRITETEXT 操作。

权限

无。

副作用

WRITETEXT 不触发触发器，缺省情况下，事务日志中不记录 WRITETEXT 操作。

另请参见

- “READTEXT 语句 [T-SQL]”一节第 664 页
- “TEXTPTR 函数 [Text and image]”一节第 310 页

标准和兼容性

- **SQL/2003** Transact-SQL 扩充。

示例

下面这段代码阐释了 WRITETEXT 语句的用法。该示例中的 SELECT 语句返回单行。该示例用值 newdata 替换指定行中的 column_name 列的内容。

```
EXEC SQL create variable textpointer binary(16);
EXEC SQL set textpointer =
    (  SELECT textptr(column_name)
       FROM table_name WHERE ID = 5 );
EXEC SQL writetext table_name.column_name
   textpointer 'newdata';
```


系统对象

本部分介绍 SQL Anywhere 的表、视图和过程。

表

目录

系统表

每个数据库的结构都是以几个系统表来描述的。SYS 用户拥有系统表。这些表的内容只能通过数据库服务器进行更改。不能使用 UPDATE、DELETE 和 INSERT 命令来修改这些表的内容。此外，也不能使用 ALTER TABLE 和 DROP 命令来更改这些表的结构。

SQL Anywhere 中的系统表通过各自的对应视图来表示。

DUMMY 系统表

列名	列类型	列约束	表约束
dummy_col	INTEGER	NOT NULL	

DUMMY 表是以只读表的形式提供的，并且始终只有一行。它有利于从数据库中抽取信息，如，以下示例用于从数据库中获取当前用户 ID 和当前日期。

```
SELECT USER, today(*) FROM SYS.DUMMY;
```

在 FROM 子句中使用 SYS.DUMMY 是可选的。如果在 FROM 子句中没有指定任何表，将假定该表为 SYS.DUMMY。上例可写成下面的形式：

```
SELECT USER, today(*);
```

dummy_col　此列未被使用。它之所以存在，是因为如果没有任何列就无法创建表。

从 SYS.DUMMY 表中读取的开销要比从用户创建的类似表中读取的开销小，这是因为在 SYS.DUMMY 的表页中没有内部锁。

访问计划不是通过扫描 SYS.DUMMY 表构建的，而是用行构造函数算法来代替对 SYS.DUMMY 的引用，该算法会虚拟化表引用。这样便避免了与 SYS.DUMMY 相关的争用问题。请注意，DUMMY 仍旧会以表和/或相关名称的形式出现在简单计划、详细计划及图形式计划中。请参见“RowConstructor 算法 (ROWS)”一节 《SQL Anywhere 服务器 - SQL 的用法》。

ISYSARTICLE 系统表

ISYSARTICLE 系统表中的每一行都描述了发布中的一个项目。请参见“SYSARTICLE 系统视图”一节第 938 页。

ISYSARTICLECOL 系统表

ISYSARTICLECOL 系统表中的每一行都标识项目中的一个列。请参见“SYSARTICLECOL 系统视图”一节第 939 页。

ISYSATTRIBUTE 系统表

此表仅供内部使用。

ISYSATTRIBUTENAME 系统表

此表仅供内部使用。

ISYSCAPABILITY 系统表

ISYSCAPABILITY 系统表中的每一行都标识远程服务器的一种功能。请参见“SYSCAPABILITY 系统视图”一节第 939 页。

ISYSCHECK 系统表

ISYSCHECK 系统表中的每一行都标识表中的一个命名检查约束。请参见“SYSCHECK 系统视图”一节第 940 页。

ISYSCOLPERM 系统表

ISYSCOLPERM 系统表中的每一行都描述了列的 UPDATE、SELECT 或 REFERENCES 权限。请参见“SYSCOLPERM 系统视图”一节第 940 页。

ISYSCOLSTAT 系统表

ISYSCOLSTAT 系统表包含优化程序所使用的列统计信息。请参见“SYSCOLSTAT 系统视图”一节第 941 页。

注意
对于使用 SQL Anywhere 11.0.0 版及更高版本创建的数据库，会始终加密此表，以防止他人对数据进行未经授权的访问。

ISYSCONSTRAINT 系统表

ISYSCONSTRAINT 系统表中的每一行都描述了除系统表之外的所有其它表的命名约束。请参见“SYSCONSTRAINT 系统视图”一节第 942 页。

ISYSDEPENDENCY 系统表

ISYSDEPENDENCY 系统表中的每一行都描述了一个表或视图依赖性。请参见“SYSDEPENDENCY 系统视图”一节第 945 页。

ISYSDBFILE 系统表

ISYSDBFILE 系统表中的每一行都描述了一个 dbspace。请参见“SYSDBFILE 系统视图”一节第 943 页。

ISYSDBSPACE 系统表

ISYSDBSPACE 系统表中的每一行都描述了一个 dbspace。请参见“SYSDBSPACE 系统视图”一节第 943 页。

ISYSDBSPACEPERM 系统表

ISYSDBSPACEPERM 系统表中的每一行都描述了一个 dbspace 的权限。请参见“SYSDBSPACEPERM 系统视图”一节第 944 页。

ISYSDOMAIN 系统表

每种预定义的数据类型（也称为**域**）都有一个指定的唯一编号。提供 ISYSDOMAIN 表是为了参考，以显示这些编号与相应数据类型之间的关联。此表永远不会更改。请参见“SYSDOMAIN 系统视图”一节第 945 页。

ISYSEVENT 系统表

ISYSEVENT 系统表中的每一行都描述一个使用 CREATE EVENT 创建的事件。请参见“SYSEVENT 系统视图”一节第 946 页。

ISYSEXTERNLOGIN 系统表

ISYSEXTERNLOGIN 系统表的每一行都描述了远程数据访问的一个外部登录。请参见“SYSEXTERNLOGIN 系统视图”一节第 948 页。

注意
对于使用 SQL Anywhere 11.0.0 版及更高版本创建的数据库，会始终加密此表，以防止他人对数据进行未经授权的访问。

ISYSFILE 系统表

ISYSFILE 系统表中的每一行都描述了数据库的一个 dbspace。每个数据库都由一个或多个 dbspace 组成；每个 dbspace 都对应一个操作系统文件。请参见“SYSFILE 兼容性视图（不建议使用）”一节第 1005 页。

ISYSFKEY 系统表

ISYSFKEY 系统表中的每一行都描述了数据库中的一个外键。请参见“SYSFKEY 系统视图”一节第 949 页。

ISYSGROUP 系统表

ISYSGROUP 系统表中的每一行都定义了一个组成员。此表介绍了组与成员之间的多对多关系。请参见“SYSGROUP 系统视图”一节第 950 页。

ISYSHISTORY 系统表

ISYSHISTORY 系统表中的每一行都指出了数据库以不同的软件版本和/或在不同的平台上启动的时间。请参见“SYSHISTORY 系统视图”一节第 951 页。

ISYSIDX 系统表

ISYSIDX 系统表中的每一行都描述了数据库中的一个索引。请参见“SYSIDX 系统视图”一节第 952 页。

ISYSIDXCOL 系统表

ISYSIDXCOL 系统表中的每一行都描述索引中的一个列。请参见“SYSIDXCOL 系统视图”一节第 953 页。

ISYSJAR 系统表

ISYSJAR 系统表中的每一行都定义了系统中的一个 JAR 文件。请参见“SYSJAR 系统视图”一节第 954 页。

ISYSJARCOMPONENT 系统表

ISYSJAR 系统表中的每一行都定义了一个 JAR 文件组件。请参见“SYSJARCOMPONENT 系统视图”一节第 955 页。

ISYSJAVACLASS 系统表

ISYSJAVACLASS 系统表中的每一行都描述了一个 Java 类。请参见“SYSJAVACLASS 系统视图”一节第 955 页。

ISYSLOGINMAP 系统表

ISYSLOGINMAP 系统表包含可用于使用集成登录或 Kerberos 登录连接到数据库的所有用户配置文件名。作为安全措施，只有具有 DBA 权限的用户才能查看该表的内容。请参见“SYSLOGINMAP 系统视图”一节第 956 页。

ISYSLOGINPOLICY 系统表

ISYSLOGINPOLICY 系统表中的每一行都描述了一个登录策略。请参见“SYSLOGINPOLICY 系统视图”一节第 957 页。

ISYSLOGINPOLICYOPTION 系统表

ISYSLOGINPOLICYOPTION 系统表中的每一行都描述了登录策略的一个选项。请参见“SYSLOGINPOLICYOPTION 系统视图”一节第 957 页。

ISYSMVOPTION 系统表

ISYSMVOPTION 系统表中的每一行都描述了实例化视图的一个选项。请参见“SYSMVOPTION 系统视图”一节第 958 页。

ISYSMVOPTIONNAME 系统表

ISYSMVOPTIONNAME 系统表中的每一行都提供了在 ISYSMVOPTION 中列出的一个实例化视图的名称。请参见“SYSMVOPTIONNAME 系统视图”一节第 958 页。

ISYSOBJECT 系统表

ISYSOBJECT 系统视图中的每一行都描述了数据库中的一个对象。数据库对象的示例包括表、视图、列、索引和过程。请参见“SYSOBJECT 系统视图”一节第 959 页。

ISYSOPTION 系统表

ISYSOPTION 系统表中的每一行都描述了针对某个用户 ID 的一个选项的设置。选项设置通过 SET 命令存储在 ISYSOPTION 表中，并且每位用户针对每个选项都有其自己的设置。请参见“SYSOPTION 系统视图”一节第 960 页。

ISYSOPTSTAT 系统表

ISYSOPTSTAT 系统表存储开销模型校准信息（由 ALTER DATABASE CALIBRATE 语句计算）。请参见“SYSOPTSTAT 系统视图”一节第 960 页。

ISYSPHYSIDX 系统表

ISYSPHYSIDX 系统表中的每一行都描述了数据库中的一个物理索引。请参见“SYSPHYSIDX 系统视图”一节第 960 页。

ISYSPROCEDURE 系统表

ISYSPROCEDURE 系统表中的每一行都描述了数据库中的一个过程。请参见“SYSPROCEDURE 系统视图”一节第 961 页。

ISYSPROCPARM 系统表

ISYSPROCPARM 系统表中的每一行都描述了针对数据库中某个过程的一个参数。请参见“SYSPROCPARM 系统视图”一节第 962 页。

ISYSPROCPERM 系统表

ISYSPROCPERM 系统表中的每一行都描述了一个被授予调用某一过程的权限的用户。请参见“SYSPROCPERM 系统视图”一节第 964 页。

ISYSPROXYTAB 系统表

ISYSPROXYTAB 系统表中的每一行都描述了一个代理表。请参见“SYSPROXYTAB 系统视图”一节第 964 页。

ISYSPUBLICATION 系统表

ISYSPUBLICATION 系统表中的每一行都描述了一个 SQL Remote 或 MobiLink 发布。请参见“SYSPUBLICATION 系统视图”一节第 965 页。

ISYSREMARK 系统表

ISYSREMARK 系统表中的每一行都描述了某个对象的一个注释。请参见“SYSREMARK 系统视图”一节第 965 页。

ISYSREMOTEOPTION 系统表

ISYSREMOTEOPTION 系统表中的每一行都描述了一个 SQL Remote 消息链接参数的值。请参见“SYSREMOTEOPTION 系统视图”一节第 966 页。

ISYSREMOTEOPTIONTYPE 系统表

ISYSREMOTEOPTIONTYPE 系统表中的每一行都描述了一个 SQL Remote 消息链接参数。请参见“SYSREMOTEOPTIONTYPE 系统视图”一节第 966 页。

ISYSREMOTETYPE 系统表

ISYSREMOTETYPE 系统表包含有关 SQL Remote 的信息。请参见“SYSREMOTETYPE 系统视图”一节第 967 页。

ISYSREMOTEUSER 系统表

ISYSREMOTEUSER 系统表中的每一行都描述了一个具有 REMOTE 权限的用户 ID（预订者），以及发送自/至该用户的 SQL Remote 消息的状态。请参见“SYSREMOTEUSER 系统视图”一节第 967 页。

ISYSSCHEDULE 系统表

ISYSSCHEDULE 系统表中的每一行都描述事件的触发时间（由 CREATE EVENT 的 SCHEDULE 子句指定）。请参见“SYSSCHEDULE 系统视图”一节第 969 页。

ISYSSERVER 系统表

ISYSSERVER 系统表中的每一行都描述一个远程服务器。请参见“SYSSERVER 系统视图”一节第 970 页。

ISYSSOURCE 系统表

ISYSSOURCE 系统视图的每行包含在 ISYSOBJECT 系统表中列出的一个对象的源对象。请参见“SYSSOURCE 系统视图”一节第 970 页。

ISYSSQLSERVERTYPE 系统表

ISYSSQLSERVERTYPE 系统表包含有关与 Adaptive Server Enterprise 兼容的信息。请参见“SYSSQLSERVERTYPE 系统视图”一节第 971 页。

ISYSSUBSCRIPTION 系统表

ISYSSUBSCRIPTION 系统表中的每一行都描述一个用户 ID（必须拥有 REMOTE 权限）对一个发布的预订。请参见“SYSSUBSCRIPTION 系统视图”一节第 971 页。

ISYSSYNC 系统表

此表包含与 MobiLink 同步相关的信息。该表中的一些列包含潜在敏感的数据。出于这个原因，仅限具有 DBA 权限的用户能够访问该表。SYSSYNC2 视图提供了访问此表中除潜在的敏感列以外的数据的公共方法。请参见“SYSSYNC 系统视图”一节第 972 页。

ISYSSYNCSCRIPT 系统表

此表包含与 MobiLink 同步脚本相关的信息。请参见“SYSSYNCSCRIPT 系统视图”一节第 973 页。

ISYSTAB 系统表

ISYSTAB 系统表中的每一行都描述数据库中的一个表。请参见“SYSTAB 系统视图”一节第 973 页。

ISYSTABCOL 系统表

ISYSTABCOL 系统表中的每一行都描述了数据库中某个表的一个列。请参见“SYSTABCOL 系统视图”一节第 976 页。

ISYSTEXTCONFIG 系统表

ISYSTEXTCONFIG 系统表中的每一行都描述了一个文本配置，以与全文搜索功能配合使用。请参见“SYSTEXTCONFIG 系统视图”一节第 979 页。

ISYSTEXTIDX 系统表

ISYSTEXTIDX 系统表中的每一行都描述了一个文本索引，以与全文搜索功能配合使用。请参见“SYSTEXTIDX 系统视图”一节第 980 页。

ISYSTEXTIDXTAB 系统表

ISYSTEXTIDXTAB 系统表中的每一行都描述了一个文本索引，以与全文搜索功能配合使用。请参见“SYSTEXTIDX 系统视图”一节第 980 页。

ISYSTABLEPERM 系统表

ISYSTABLEPERM 系统表中的每一行都与一个表、一个授予权限的用户 ID（**授予者**）和一个被授予权限的用户 ID（**被授予者**）相对应。请参见“SYSTABLEPERM 系统视图”一节第 977 页。

ISYSTRIGGER 系统表

ISYSTRIGGER 系统表中的每一行描述数据库中的一个触发器。请参见“SYSTRIGGER 系统视图”一节第 981 页。

ISYSTYPEMAP 系统表

ISYSTYPEMAP 系统表包含 ISYSSQLSERVERTYPE 系统表的兼容性映射值。请参见“SYSTYPEMAP 系统视图”一节第 983 页。

ISYSUSER 系统表

ISYSUSER 系统表中的每一行都描述了系统中的一位用户。请参见“SYSUSER 系统视图”一节第 983 页。

注意
对于使用 SQL Anywhere 11.0.0 版及更高版本创建的数据库，会始终加密此表，以防止他人对数据进行未经授权的访问。

ISYSUSERAUTHORITY 系统表

ISYSUSERAUTHORITY 系统表中的每一行都描述了已授予某位用户的权限。请参见“SYSUSERAUTHORITY 系统视图”一节第 984 页。

ISYSUSERMESSAGE 系统表

ISYSUSERMESSAGE 系统表中的每一行都包含有关错误情况的用户定义消息。请参见“SYSUSERMESSAGE 系统视图”一节第 985 页。

ISYSUSERTYPE 系统表

ISYSUSERTYPE 系统表中的每一行都描述了一种用户定义的数据类型。请参见“SYSUSERTYPE 系统视图”一节第 985 页。

ISYSVIEW 系统表

ISYSVIEW 系统表中的每一行描述了数据库中的一个视图。请参见“SYSVIEW 系统视图”一节第 986 页。

ISYSWEBSERVICE 系统表

ISYSWEBSERVICE 系统表中的每一行都描述了一项 Web 服务。请参见“SYSWEBSERVICE 系统视图”一节第 987 页。

诊断跟踪表

下面是一些用于应用程序分析和诊断跟踪的主表。这些表属于 dbo 用户。对于其中的许多表来说，它们都有一个具有相似名称和模式的全局共享临时表。例如，sa_diagnostic_blocking 表有一个全局临时表对等项，即 sa_tmp_diagnostic_blocking 表，后者与前者具有相同的模式。在跟踪会话期间，诊断数据会写入到这些临时表中。由于不对临时表进行记录，因此它们能够在跟踪会话期间提供优良的性能，其中重要的是它们能够将对服务器的影响降到最低。

另请参见

- “应用程序分析”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “使用诊断跟踪进行高级应用程序分析”一节 《SQL Anywhere 服务器 - SQL 的用法》

sa_diagnostic_auxiliary_catalog 表

sa_diagnostic_auxiliary_catalog 表属于 dbo 用户，它用于在生产数据库和跟踪数据库之间映射数据库对象。这些对象包括用户表、过程和函数。此表主要由索引顾问和 TRACED_PLAN 函数使用。

列

列名	列类型	列约束	表约束
original_object_id	UNSIGNED BIGINT	NOT NULL	主键。
local_object_id	UNSIGNED BIGINT	NOT NULL	唯一。
pages_if_table	UNSIGNED INT		
rows_if_table	UNSIGNED BIGINT		

original_object_id 此对象在主跟踪数据库中的对象 ID。

local_object_id 此对象在辅助跟踪数据库中的对象 ID。

pages_if_table 如果对象是一个表，则它是表的页数。如果对象不是表，则此值为 NULL。

rows_if_table 如果对象是一个表，则它是表的行数。如果对象不是表，则此值为 NULL。

另请参见

- “TRACED_PLAN 函数 [Miscellaneous]”一节第 314 页
- “索引顾问”一节 《SQL Anywhere 服务器 - SQL 的用法》

sa_diagnostic_blocking 表

sa_diagnostic_blocking 表属于 dbo 用户，它用于记录阻塞事件。如果启用了记录阻塞事件，则每当在尝试访问资源时连接受到阻塞，就会在此表中插入一行。通常情况下，该操作是由表锁或行锁所引起的。大量的阻塞可能表明您应当检查应用程序中的并发操作，以减少表和行的争用问题。

此表有两个版本：sa_diagnostic_blocking 和 sa_tmp_diagnostic_blocking。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	外键引用 sa_diagnostic_cursor。 外键引用 sa_diagnostic_request。
lock_id	UNSIGNED BIGINT	NOT NULL	
request_id	UNSIGNED BIGINT		外键引用 sa_diagnostic_request。
cursor_id	UNSIGNED BIGINT		外键引用 sa_diagnostic_cursor。
original_table_object_id	UNSIGNED BIGINT		
rowid	UNSIGNED BIGINT		
block_time	TIMESTAMP	NOT NULL	
unblock_time	TIMESTAMP		
blocked_by	UNSIGNED INT	NOT NULL	

logging_session_id 唯一地标识记录会话（在此会话期间收集诊断信息）的编号。

lock_id 当行锁或表锁导致阻塞时导致产生阻塞的那个锁的 ID，否则为 NULL。

request_id 当阻塞不是因游标而引起时，为被阻塞请求的 ID，否则为 NULL。此值与 sa_diagnostic_request 中指定给请求的 ID 相对应。

cursor_id 当阻塞是因为游标而引起时，为游标的 ID，否则为 NULL。此值与 sa_diagnostic_cursor 中指定给游标的 ID 相对应。

original_table_object_id 当阻塞是因为表锁而引起时，为发生阻塞的表的 ID，否则为 NULL。

rowid 当阻塞是因为行锁而引起时，为发生阻塞的行的 ID，否则为 NULL。

block_time 阻塞的发生时间。

unblock_time 阻塞的结束时间。

blocked_by 连接的 ID，该连接包含引发阻塞的锁。

另请参见

- “事务阻塞和死锁”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “锁定的工作方式”一节 《SQL Anywhere 服务器 - SQL 的用法》

sa_diagnostic_cachecontents 表

sa_diagnostic_cachecontents 表属于 dbo 用户。启用诊断跟踪后，便会定期拍下高速缓存内容的快照。拍快照时，sa_diagnostic_cachecontents 表会记录高速缓存中每个表的表页数和行数。优化程序可利用这些信息重新创建最初优化查询时的条件，然后做出优化决定。

只要存在查询活动，便会每 20 秒对 sa_diagnostic_cachecontents 表中的数据更新一次。

此表有两个版本：sa_diagnostic_cachecontents 和 sa_tmp_diagnostic_cachecontents。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	
"time"	TIMESTAMP	NOT NULL	主键。
original_table_object_id	UNSIGNED BIGINT	NOT NULL	主键。
pages_in_cache	UNSIGNED INT	NOT NULL	
num_table_pages	UNSIGNED INT	NOT NULL	
num_table_rows	UNSIGNED BIGINT	NOT NULL	

logging_session_id 唯一地标识记录会话（在此会话期间收集诊断信息）的编号。

"time" 拍高速缓存快照时的时间。

original_table_object_id 在快照中出现的每个表的对象 ID。

pages_in_cache 对于快照中的指定表，拍快照时高速缓存中的总页数。

num_table_pages 对于快照中的指定表，该表的总页数。

num_table_rows 对于快照中的指定表，该表的总行数。

sa_diagnostic_connection 表

sa_diagnostic_connection 表属于 dbo 用户，对于在记录会话期间处于活动状态的每个数据库连接，该表中均只有一行。对于在记录会话中发生的连接和断开连接，其次数可从 sa_diagnostic_request 表中得出。

此表中的大多数值都反映了连接属性的值。

此表有两个版本：sa_diagnostic_connection 和 sa_tmp_diagnostic_connection。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	主键。
connection_number	UNSIGNED INT		主键。
connection_name	LONG VARCHAR		
user_name	LONG VARCHAR		
comm_link	CHAR(40)		
node_address	LONG VARCHAR		
appinfo	LONG VARCHAR		

logging_session_id 唯一地标识记录会话（在此会话期间收集诊断信息）的编号。

connection_number 由数据库服务器指定的编号，用于标识用户与数据库之间的连接。此值反映了 Number 连接属性的值。

connection_name 连接的可选名称属性。此值反映了 Name 连接属性的值。

user_name 连接到数据库的用户的名称。

comm_link 指定客户端网络协议选项。此值反映了 CommLinks 连接属性的值。

node_address 客户端/服务器连接中客户端的节点。此值反映了 NodeAddress 连接属性的值。

appinfo 有关客户端进程的信息，如客户端计算机的 IP 地址、运行该客户端进程的操作系统，等等。此值反映了 AppInfo 连接属性的值。

另请参见

- “连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》

sa_diagnostic_cursor 表

sa_diagnostic_cursor 表属于 dbo 用户。每一行都描述了在记录会话期间所打开的一个内部或外部游标。

此表有两个版本：sa_diagnostic_cursor 和 sa_tmp_diagnostic_cursor。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	主键。 外键引用 sa_diagnostic_query。

列名	列类型	列约束	表约束
cursor_id	UNSIGNED BIGINT	NOT NULL	主键。
query_id	UNSIGNED BIGINT	NOT NULL	外键引用 sa_diagnostic_query。
isolation_level	TINYINT		
flags	UNSIGNED INT		
forward_fetches	UNSIGNED INT		
reverse_fetches	UNSIGNED INT		
absolute_fetches	UNSIGNED INT		
first_fetch_time_ms	UNSIGNED INT		
total_fetch_time_ms	UNSIGNED INT		
plan_xml	LONG VARCHAR		

logging_session_id 唯一地标识记录会话（在此会话期间收集诊断信息）的编号。

cursor_id 标识游标的唯一编号。

query_id 标识此游标所涉及的查询。

isolation_level 打开此游标的隔离级别。

flags 内部使用。

forward_fetches 在游标上完成的向前读取数（包括预读）。

reverse_fetches 在游标上完成的反向读取数（包括预读）。

absolute_fetches 在游标上完成的绝对读取数。

first_fetch_time_ms 读取第一行所花费的持续时间。

total_fetch_time_ms 读取所花费的持续时间。此值不包括实际读取之间的应用程序处理时间（思考时间）。

plan_xml 在该游标关闭时转储的那些游标的详细计划。这些计划包含相应的详细统计信息。

另请参见

- “游标简介”一节 《SQL Anywhere 服务器 - 编程》

sa_diagnostic_deadlock 表

sa_diagnostic_deadlock 表属于 dbo 用户。在启用诊断跟踪并将其设置为包括跟踪死锁事件后，每当发生死锁时就会在此表中插入一组行（对于组成死锁的每一个连接都会插入一行）。构成单个死锁事件的所有行的集合由 snapshot_id 唯一地标识。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	
snapshot_id	UNSIGNED BIGINT	NOT NULL	
snapshot_at	TIMESTAMP	NOT NULL	
waiter	UNSIGNED INT	NOT NULL	
request_id	UNSIGNED BIGINT		
original_table_object_id	UNSIGNED BIGINT		
rowid	UNSIGNED BIGINT		
owner	UNSIGNED INT	NOT NULL	
rollback_operation_count	UNSIGNED INT	NOT NULL	

logging_session_id 唯一地标识记录会话（在此会话期间收集诊断信息）的编号。

snapshot_id 用于标识其中包含此行的那个死锁事件的编号。注意，此列与快照隔离无关。

snapshot_at 死锁的发生时间。

waiter 此行所表示的连接的连接编号。

request_id 死锁发生时此连接所正在处理的请求的 ID。

original_table_object_id 阻塞此连接的那个表的对象 ID。

rowid 阻塞此连接的那个行的记录 ID。

owner 锁定所需行的连接的连接编号。

rollback_operation_count 未提交操作的数量。

另请参见

- “事务阻塞和死锁”一节 《SQL Anywhere 服务器 - SQL 的用法》

sa_diagnostic_hostvariable 表

sa_diagnostic_hostvariable 表属于 dbo 用户，其中包含指定游标所使用的主机变量的值。

此表有两个版本：sa_diagnostic_hostvariable 和 sa_tmp_diagnostic_hostvariable。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	主键。 外键引用 sa_diagnostic_request。
request_id	UNSIGNED BIGINT	NOT NULL	主键。 外键引用 sa_diagnostic_request。
cursor_id	UNSIGNED BIGINT		
hostvar_num	UNSIGNED SMALLINT	NOT NULL	主键。
hostvar_type	UNSIGNED TINYINT	NOT NULL	
hostvar_value	LONG VARCHAR		

logging_session_id 唯一地标识记录会话（在此会话期间收集诊断信息）的编号。

request_id 主机变量所属请求的 ID。

cursor_id 主机变量所属游标的 ID。

hostvar_num 主机变量在 SQL 语句中的顺序位置。

hostvar_type 主机变量的域编号，通常为字符串、整数或浮点数。

hostvar_value 用于表示主机变量的值的字符串。即使主机变量为整数或浮点数，此时仍将该值表示为字符串。

另请参见

- “使用主机变量”一节 《SQL Anywhere 服务器 - 编程》

sa_diagnostic_internalvariable 表

sa_diagnostic_internalvariable 表属于 dbo 用户，其中包含给定语句所使用的内部（本地）变量的值。此表主要由索引顾问和 traced_plan 函数使用。

此表有两个版本：sa_diagnostic_internalvariable 和 sa_tmp_diagnostic_internalvariable。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	
request_id	UNSIGNED BIGINT		
rowvariable_id	UNSIGNED INT		
variable_domain	UNSIGNED SMALLINT		
variable_name	CHAR(128)		
variable_value	LONG VARCHAR		

logging_session_id 唯一地标识记录会话（在此会话期间收集诊断信息）的编号。

request_id 包含内部变量的请求的 ID。

rowvariable_id 此值的行变量中的列编号。

variable_domain 内部变量的数据类型。

variable_name 内部变量的名称。

variable_value 用于表示内部变量的值的字符串。

另请参见

- “局部变量”一节第 64 页

sa_diagnostic_query 表

sa_diagnostic_query 表属于 dbo 用户，其中存储查询的优化信息，尤其是优化查询时的上下文。此表中的一行表示针对查询调用的一次优化程序。优化时所捕获的计划被存储在这里。

此表中的某些值反映了数据库选项值。

此表有两个版本：sa_diagnostic_query 和 sa_tmp_diagnostic_query。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	主键。外键引用 sa_diagnostic_statement。
query_id	UNSIGNED BIGINT	NOT NULL	主键。外键引用 sa_diagnostic_statement。

列名	列类型	列约束	表约束
statement_id	UNSIGNED BIGINT	NOT NULL	
user_object_id	UNSIGNED BIGINT	NOT NULL	
start_time	TIMESTAMP	NOT NULL	
cache_size_bytes	UNSIGNED BIGINT		
optimization_goal	TINYINT		
optimization_level	TINYINT		
user_estimates	TINYINT		
optimization_workload	TINYINT		
available_requests	TINYINT		
active_requests	TINYINT		
max_tasks	TINYINT		
used_bypass	TINYINT		
estimated_cost_ms	TINYINT		
plan_explain	LONG VARCHAR		
plan_xml	LONG VARCHAR		
sql_rewritten	LONG VARCHAR		

logging_session_id 在期间发生查询或请求的那个记录会话的 ID。

query_id 唯一地标识查询的编号。

statement_id 唯一地标识查询中的语句的编号。

user_object_id 在其下执行此查询的用户的对象 ID。如果查询是从过程运行的，则这将是过程所有者的用户 ID。

start_time 优化此查询时的时间。

cache_size_bytes 优化此查询时高速缓存的大小（以字节为单位）。

optimization_goal 确定优化查询处理的意图：是迅速返回第一行，还是为最大程度地降低返回整个结果集的开销。此值反映了 optimization_goal 数据库选项的值。

要查看此列的可能值，请参见“optimization_goal 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

optimization_level 控制 SQL Anywhere 查询优化程序为查找 SQL 语句的访问计划而消耗的资源量。此值反映了 optimization_level 数据库选项的值。

要查看此列的可能值，请参见“optimization_level 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

user_estimates 控制查询优化程序是考虑还是忽略查询谓语中的用户选择性估计。此值反映了 user_estimates 数据库选项的值。

要查看此列的可能值，请参见“user_estimates 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

optimization_workload 确定优化查询处理的目标是针对更新和读取混合进行的负载还是针对主要基于读取的负载。此值反映了 optimization_workload 数据库选项的值。

要查看此列的可能值，请参见“optimization_workload 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

available_requests 内部使用，以计算查询内并行机制的级别。

active_requests 内部使用，以计算查询内并行机制的级别。

max_tasks 内部使用，以计算查询内并行机制的级别。

used_bypass 是否使用了简单查询忽略。值 1 表示使用了忽略；值 0 表示查询已完全优化。

estimated_cost_ms 预计开销（毫秒）。

plan_explain 此查询的文本计划表示。

plan_xml 查询的图形式计划表示（如果记录了一个查询）。

sql_rewritten 应用优化后查询的文本。仅当启用优化记录后，此列中才会显示有值。

另请参见

- “数据库选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “优化程序的工作原理”一节 《SQL Anywhere 服务器 - SQL 的用法》

sa_diagnostic_request 表

sa_diagnostic_request 表属于 dbo 用户，它是所有请求的主表。一个请求就是一个与查询处理相关的事件，通常包括：

- 连接或断开连接事件
- 语句执行
- 语句准备
- 打开游标或删除游标事件

此表有两个版本：sa_diagnostic_request 和 sa_tmp_diagnostic_request。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	主键。 外键引用 sa_diagnostic_connection。 外键引用 sa_diagnostic_cursor。 外键引用 sa_diagnostic_query。 外键引用 sa_diagnostic_statement。
request_id	UNSIGNED BIGINT	NOT NULL	主键。
start_time	TIMESTAMP	NOT NULL	
finish_time	TIMESTAMP	NOT NULL	
duration_ms	UNSIGNED INT	NOT NULL	
connection_number	UNSIGNED INT		外键引用 sa_diagnostic_connection。
request_type	UNSIGNED SMALLINT		
statement_id	UNSIGNED BIGINT		外键引用 sa_diagnostic_statement。
query_id	UNSIGNED BIGINT		外键引用 sa_diagnostic_query。
cursor_id	UNSIGNED BIGINT		外键引用 sa_diagnostic_cursor。
sql_code	SMALLINT		

logging_session_id　在期间发生请求的那个记录会话。

request_id　唯一地标识请求的编号。

start_time　事件的开始时间。

finish_time　对于语句执行而言，指的是语句的完成时间；否则为 NULL。

duration_ms　事件的持续时间（毫秒）。

connection_number　导致事件发生的那个连接的 ID。

request_type　请求的类型。值包括：

值	说明
1	启动新的跟踪会话

值	说明
2	语句执行
3	打开游标
4	关闭游标
5	连接
6	断开连接

statement_id 如果事件与语句相关，指的是进行跟踪的语句的 ID。

query_id 如果事件与查询相关，指的是指定给查询以进行跟踪的 ID。

cursor_id 如果事件与游标相关，指的是指定给游标以进行跟踪的 ID。

sql_code 由于此表中的各行分别表示对语句、游标或查询执行的操作，因此大多都返回一个 SQL 代码。此列包含返回的 SQL 代码。如果返回的 SQL 代码为 0，则该列包含 NULL。

sa_diagnostic_statement 表

sa_diagnostic_statement 表属于 dbo 用户，它用于存储语句的文本。此表中的一行表示一个已由服务器执行的 SQL 语句。此类语句可能已由外部源（如客户端请求）或内部源（如过程、触发器或用户定义的函数）发出。其中，内部语句在每个会话中仅出现一次。

此表有两个版本：sa_diagnostic_statement 和 sa_tmp_diagnostic_statement。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	主键。
statement_id	UNSIGNED BIGINT	NOT NULL	主键。
database_object	UNSIGNED BIGINT		
line_number	UNSIGNED SMALLINT		
signature	UNSIGNED INT		
statement_text	LONG VARCHAR	NOT NULL	

logging_session_id 在期间提交语句的那个记录会话。

statement_id 对于进行跟踪的语句的唯一编号。

database_object 如果语句来自过程、触发器或函数，则为在 ISYSOBJECT 系统表中指定的 ID。

line_number 如果语句是某个复合语句的组成部分，则反映了该语句在复合语句中的顺序位置。

signature 在内部使用，以将相似的查询分组。

statement_text 语句文本。

sa_diagnostic_statistics 表

sa_diagnostic_statistics 表属于 dbo 用户，其中包含服务器中所拥有的性能计数器的历史记录。每一行都表示给定性能计数器在某一给定时间的瞬时值。

此表有两个版本：sa_diagnostic_statistics 和 sa_tmp_diagnostic_statistics。

列

列名	列类型	列约束	表约束
logging_session_id	UNSIGNED INT	NOT NULL	
"time"	TIMESTAMP	NOT NULL	
counter_id	UNSIGNED SMALLINT	NOT NULL	
type	TINYINT	NOT NULL	
connection_number	UNSIGNED INT	NOT NULL	
counter_value	UNSIGNED INT	NOT NULL	

logging_session_id 唯一地标识记录会话（在此会话期间收集诊断信息）的编号。

"time" 捕获性能计数器值时的时间。

counter_id 唯一地标识性能计数器的编号。使用 PROPERTY_NAME 函数，您可以获得此 counter_id 所表示的属性的名称。

type 指明它是数据库、服务器还是连接统计信息。可能的值有：0 表示服务器、1 表示数据库、2 表示连接、4 表示外部数据库。

connection_number 对于连接统计信息，它表示从中捕获此属性的连接编号。对于扩展数据库统计信息，它表示从中捕获此属性的文件的文件编号。否则，该值为 0。

counter_value 性能计数器的值。

另请参见

- “PROPERTY_NAME 函数 [System]”一节第 257 页

sa_diagnostic_tracing_level 表

sa_diagnostic_tracing_level 表属于 dbo 用户，此表中的每一行都是一个用于决定将何种类型的诊断信息发送到跟踪数据库的条件。如果某段记录数据符合此表中一行或多行条件，则会记录相应的数据。

此表中的数据使用 CONNECT TRACING 或 REFRESH TRACING LEVELS 语句进行填充。

列

列名	列类型	列约束	表约束
id	UNSIGNED INT	NOT NULL	主键。
scope	CHAR(32)	NOT NULL	
identifier	CHAR(128)		
trace_type	CHAR(32)	NOT NULL	
trace_condition	CHAR(32)		
value	UNSIGNED INT		
enabled	BIT	NOT NULL	

scope 诊断跟踪的范围，如下面列出的内容所示。要查看每个范围的说明，请参见“诊断跟踪范围”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- DATABASE
- ORIGIN
- USER
- CONNECTION_NAME
- CONNECTION_NUMBER
- FUNCTION
- PROCEDURE
- EVENT
- TRIGGER
- TABLE

id 仅供内部使用。

identifier 范围的标识符。此值会随着指定范围的改变而发生改变。例如：

- 如果 *scope* 是 DATABASE，则 *identifier* 可能不存在。
- 如果 *scope* 是 ORIGIN，则 *identifier* 必须是 Internal 或 External。
- 如果 *scope* 是 USER，则 *identifier* 是用户的 ID。

- 如果 *scope* 是 CONNECTION_NAME 或 CONNECTION_NUMBER，则 *identifier* 分别是连接的名称或编号。
- 如果 *scope* 是 FUNCTION、PROCEDURE、EVENT、TRIGGER 或 TABLE，则 *identifier* 是对象的完全限定标识符。

trace_type 在指定范围中要跟踪数据的类型，如下面列出的内容所示。要查看每种跟踪类型的说明，请参见“诊断跟踪类型”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- VOLATILE_STATISTICS
- NONVOLATILE_STATISTICS
- CONNECTION_STATISTICS
- BLOCKING
- PLANS
- PLANS_WITH_STATISTICS
- STATEMENTS
- STATEMENTS_WITH_VARIABLES
- OPTIMIZATION_LOGGING
- OPTIMIZATION_LOGGING_WITH_PLANS

condition 仅适用于计划，用于控制着是跟踪大量的高开销查询还是跟踪优化程序未对其做出最佳选择的查询。下面列出了可能的值。要查看每个条件的说明，请参见“诊断跟踪条件”一节 《SQL Anywhere 服务器 - SQL 的用法》。

- NONE 或 NULL
- SAMPLE_EVERY
- ABSOLUTE_COST
- RELATIVE_COST_DIFFERENCE

condition_value 该值与 *condition* 相关联。例如，如果 *condition* 为 SAMPLE_EVERY，则 *condition_value* 将是一个反映时间的正整数（以毫秒为单位）。其它规则如下所示：

- 如果 *condition* 为 NULL 或 NONE，则不存在 *condition_value*。
- 如果 *condition* 为 ABSOLUTE_COST，则 *condition_value* 反映执行语句的总实际开销（以毫秒为单位）。
- 如果 *condition* 为 RELATIVE_COST_DIFFERENCE，则 *condition_value* 将以预计开销百分比的形式反映执行开销。

enabled 行是否已启用。即行中的跟踪设置是否处于活动状态。1 表示启用；0 表示禁用。

另请参见

- “ATTACH TRACING 语句”一节第 388 页
- “REFRESH TRACING LEVEL 语句”一节第 669 页

其它表

下面是有关其它表（如 Java 在数据库和 SQL Remote 中所使用的系统表）的信息。

RowGenerator 表 (dbo)

dbo.RowGenerator 表是以只读表形式提供的，其中包含 255 行。对于会生成较小结果集以及需要某一范围数字值的查询，此表可能会非常有用。

RowGenerator 表由系统过程和视图所使用，不能以任何方式对其进行修改。

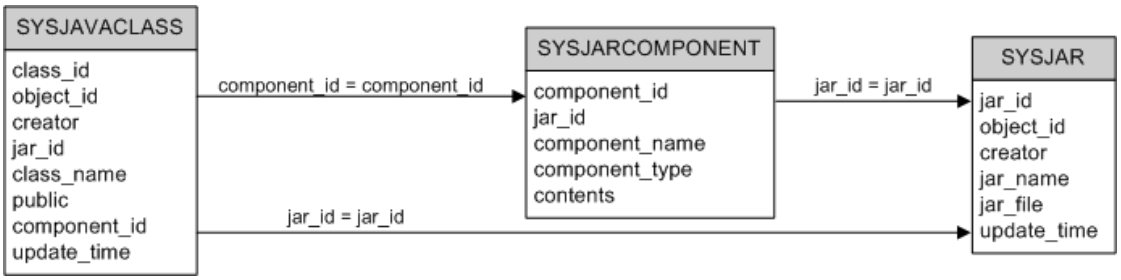
您还可以使用 sa_rowgenerator 过程生成一定范围内的数字值。有关使用 sa_rowgenerator 系统过程（包括示例）的详细信息，请参见“sa_rowgenerator 系统过程”一节第 881 页。

列名	列类型	列约束	基础表约束
row_num	SMALLINT	NOT NULL	

row_num 介于 1 和 255 之间的值。

Java 系统表

下面列出了 Java 使用的系统表。各表之间的外键关系由箭头加以指示：箭头从外表指向主表。



MobiLink 系统表

有关 MobiLink 系统表的信息，请参见“MobiLink 服务器系统表” 《MobiLink - 服务器管理》。

SQL Remote 系统表

有关 SQL Remote 系统表的信息，请参见“SQL Remote 系统表”一节 《SQL Remote》。

UltraLite 系统表

有关 UltraLite 系统表的信息，请参见“UltraLite 系统表” 《UltraLite - 数据库管理和参考》。

系统过程

目录

系统过程简介

以下各节记录了 SQL Anywhere 所附带的系统过程。某些系统过程（例如 sa_get_table_definition）是以函数形式实现的。但是，因为它们与系统过程一样以相同的方式用于相同的上下文中，所以它们随系统过程一同提供，并且命名方式与系统过程类似 (sa_xxx)。

SQL Anywhere 包含以下几种系统过程：

- 系统过程，用于以表格形式显示系统信息。
- SOAP 和 HTTP 服务系统过程，用于支持 Web 服务。
- MAPI 和 SMTP 系统过程，用于发送电子邮件。
- Transact-SQL 系统和类过程。请参见“Adaptive Server Enterprise 系统和分类过程”一节第 789 页。

♦ 查看有关系统过程和函数的详细信息

1. 使用 DBA 权限连接到数据库。
2. 右击数据库，然后选择 [**配置所有者过滤**]。
3. 单击 [**DBO**]，然后单击 [**确定**]。
4. 在左窗格中，双击 [**过程和函数**]。
5. 在左窗格中选择过程，然后在右窗格中单击 [**SQL**] 选项卡。

Web 服务系统过程

以下系统过程用于 Web 服务：

- “sa_http_header_info 系统过程”一节第 837 页
- “sa_http_php_page 系统过程”一节第 838 页
- “sa_http_php_page_interpreted 系统过程”一节第 838 页
- “sa_http_variable_info 系统过程”一节第 840 页
- “sa_set_http_header 系统过程”一节第 894 页
- “sa_set_http_option 系统过程”一节第 895 页
- “sa_set_soap_header 系统过程”一节第 897 页

还有许多函数可用于 Web 服务。请参见“Web 服务函数”一节第 124 页。

MAPI 和 SMTP 过程

SQL Anywhere 中包含使用 Microsoft 的消息传递 API（Messaging API，简称 MAPI）标准或 Internet 标准简单邮件传输协议（Simple Mail Transfer Protocol，简称 SMTP）发送电子邮件的系统过程。这些系统过程是作为扩展系统过程来实现的：每个过程都调用外部 DLL 中的一个函数。

这些过程的所有者用户 ID 为 dbo。用户在使用这些过程之前必须被授予 EXECUTE 权限，除非已具有 DBA 权限。

要使用 MAPI 或 SMTP 系统过程，必须可以从数据库服务器计算机上访问 MAPI 或 SMTP 电子邮件系统。

MAPI 和 SMTP 系统过程是：

- **xp_startmail** 用指定的邮件帐户登录到 MAPI 消息系统，以启动邮件会话。请参见“xp_startmail 系统过程”一节第 931 页。
- **xp_startsmtp** 用指定的邮件帐户登录到 SMTP 消息系统，以启动邮件会话。请参见“xp_startsmtp 系统过程”一节第 932 页。
- **xp_sendmail** 将邮件消息发送给指定的用户。请参见“xp_sendmail 系统过程”一节第 926 页。
- **xp_stopmail** 关闭 MAPI 邮件会话。请参见“xp_stopmail 系统过程”一节第 933 页。
- **xp_stopsmtp** 关闭 SMTP 邮件会话。请参见“xp_stopsmtp 系统过程”一节第 934 页。

示例

以下过程通知一组用户：备份已经完成。

```
CREATE PROCEDURE notify_backup( )
BEGIN
   CALL xp_startmail( mail_user='ServerAccount',
                  mail_password='ServerPassword'
                     );
   CALL xp_sendmail( recipient='IS Group',
                     subject='Backup',
                     "message"='Backup completed'
                     );
   CALL xp_stopmail( )
END;
```

MAPI 和 SMTP 系统过程的返回代码

以下代码可能由 MAPI 或 SMTP 系统过程返回：

返回代码	含义
0	成功
2	xp_startmail 或 xp_startsmtp 失败
3	xp_stopmail 或 xp_stopsmtp 失败
5	xp_sendmail 失败
12	没有找到附件

返回代码	含义
15	内存不足
20	未知的接收者
25	邮件会话启动失败

此外，MAPI 系统过程还可能返回以下代码：

返回代码	含义
11	接收者不明确
12	没有找到附件
13	磁盘已满
14	失败
15	内存不足
16	会话无效
17	文本太大
18	文件太多
19	接收者太多
20	未知的接收者
21	登录失败
22	会话太多
23	用户中止
24	找不到 MAPI

此外，SMTP 系统过程还可能返回以下代码：

返回代码	含义
100	套接字错误。
101	套接字超时。
102	无法解析 SMTP 服务器主机名。

返回代码	含义
103	无法连接到 SMTP 服务器。
104	服务器错误，不理解响应。例如，消息的格式不正确，或者服务器不是 SMTP。
421	*domain* 服务不可用，正在关闭传输通道。
450	请求的邮件操作未执行：邮箱不可用。
451	请求的操作未执行：处理时发生本地错误。
452	请求的操作未执行：系统存储不足。
500	语法错误，命令无法识别。（这可能包括命令太长这样的错误）。
501	参数语法错误。
502	命令未实现。
503	命令的顺序错误。
504	命令参数未实现。
550	请求的操作未执行：邮箱不可用。例如，未找到邮箱，没有访问权或者不允许转发。
551	非本地用户；请尝试 *forward-path*
552	请求的邮件操作已中止：超过分配的存储。
553	请求的操作未执行：邮箱名被禁用。例如，邮箱语法不正确。
554	事务处理失败。

Adaptive Server Enterprise 系统和分类过程

Adaptive Server Enterprise 提供了执行许多管理功能和获取系统信息的系统和分类过程。系统过程是用于从系统中获取报告和更新系统表的内置存储过程；分类存储过程以表格形式从系统表中检索信息。

SQL Anywhere 已实现了对这些 Adaptive Server Enterprise 过程中的某些过程的支持。然而，有关使用这些过程的信息，请参见 Adaptive Server Enterprise 文档。

Adaptive Server Enterprise 系统过程

以下列表介绍了 SQL Anywhere 中提供的 Adaptive Server Enterprise 系统过程。

虽然这些过程执行的功能与它们在 Adaptive Server Enterprise 和 Adaptive Server IQ 12 版之前的版本中执行的功能相同，但是它们并不完全相同。如果您的先前存在的脚本使用这些过程，您可能需要检查一下这些过程。要查看存储过程的文本，可以在 Sybase Central 中打开它，或者在 Interactive SQL 中运行以下命令。

```
sp_helptext 'dbo.procedure_name'
```

为了查看完整的文本，可能需要重置 Interactive SQL 输出的宽度，方法是选择 [**工具**] » [**选项**] » [**SQL Anywhere**] » [**截断长度**]，然后输入一个新值。

系统过程名	说明
sp_addgroup	将组添加到数据库中
sp_addlogin	将新登录 ID 添加到数据库中
sp_addmessage	将用户定义的消息添加到 ISYSUSERMESSAG 中，供存储过程 PRINT 和 RAISERROR 调用使用
sp_addtype	创建用户定义的数据类型
sp_adduser	将新用户 ID 添加到数据库中
sp_changegroup	更改用户所在组或将用户添加到组中
sp_dropgroup	从数据库中删除组
sp_droplogin	从数据库中删除一个登录 ID
sp_dropmessage	删除用户定义的消息
sp_droptype	删除用户定义的数据类型
sp_dropuser	从数据库中删除一个用户 ID
sp_getmessage	从 ISYSUSERMESSAGE 中检索存储消息字符串，供 PRINT 和 RAISERROR 语句使用。
sp_helptext	显示系统过程、触发器或视图的文本
sp_password	添加或更改用户 ID 的口令

Adaptive Server Enterprise 分类过程

SQL Anywhere 实现了 Adaptive Server Enterprise 分类过程的子集。下表描述了已实现的分类过程。

分类过程名	说明
sp_column_privileges	不支持
sp_columns	返回指定列的数据类型
sp_fkeys	返回指定表的外键信息
sp_pkeys	返回指定表的主键信息
sp_special_columns	返回唯一标识指定表中的行的优化列集
sp_sproc_columns	返回存储过程的输入和返回参数信息
sp_statistics	返回有关表及其索引的信息
sp_stored_procedures	返回有关一个或多个存储过程的信息
sp_tables	返回可以显示在指定表的 FROM 子句中的对象列表

按字母顺排序的系统过程列表

系统过程由用户 ID dbo 所拥有。其中一些过程供系统内部使用。本节只介绍不仅仅供系统内部使用的过程。不能在 Windows Mobile 上调用外部函数。

openxml 系统过程

从 XML 文档生成一个结果集。

语法

openxml(*xml-data*,
xpath [**,** *flags* [**,** *namespaces*]] **)**
WITH (*column-name column-type* [*xpath*],... **)**

参数

- **xml_data** 生成结果集所基于的 XML。可以是任意的字符串表达式，如常量、变量或列。
- **xpath** 包含 XPath 查询的字符串。XPath 允许您指定描述要查询的 XML 文档结构的模式。此参数中包含的 XPath 模式将从 XML 文档中选择节点。每个与出现在第二个 *xpath* 参数中的 XPath 查询相匹配的节点都将在表中生成一行记录。

  元属性只能在 WITH 子句的 *xpath* 参数中指定。一个元属性可在 XPath 查询中作为一个属性来访问。如果不指定 *namespaces*，则缺省情况下将前缀 mp 绑定到统一资源定位符（Uniform Resource Identifier，简称 URI）urn:ianywhere-com:sa-xpath-metaprop。如果指定了一个 *namespaces*，则此 URI 必须绑定到 mp 或某个其它前缀以访问查询中的元属性。元属性名称区分大小写。openxml 语句支持以下元属性：

  - **@mp:id** 返回一个在 XML 文档中唯一的节点的 ID。当数据库服务器重新启动后，给定文档中给定节点的 ID 可能会更改。此元属性的值按文档的顺序递增。
  - **@mp:localname** 返回节点名称中的本地部分，如果该节点没有名称则返回 NULL。
  - **@mp:prefix** 返回节点名称中的前缀部分，如果该节点没有名称或名称没有前缀则返回 NULL。
  - **@mp:namespaceuri** 返回该节点所属的命名空间的 URI，如果该节点不在命名空间中则返回 NULL。
  - **@mp:xmltext** 以 XML 形式返回 XML 文档的子树。例如，当匹配了一个内部节点时，可使用此元属性来返回一个 XML 字符串，而不再是下级文本节点的连接值。
- **flags** 当 WITH 子句中没有指定 XPath 查询时，指明在 XML 数据和结果集之间应使用的映射。如果没有指定 *flags* 参数，则缺省的行为是将属性映射到结果集中的列。*flags* 参数可为以下值之一：

Value	说明
1	将 XML 属性映射到结果集中的列（缺省）。

Value	说明
2	将 XML 元素映射到结果集中的列。

- **namespace-declaration** 一个 XML 文档。查询的作用域命名空间取自该文档的根元素。如果指定命名空间，则必须给出一个 *flags* 参数，即使已指定了所有的 *xpath* 参数。
- **WITH 子句** 指定结果集的模式及如何在结果集中查找每一列的值。WITH 子句的 *xpath* 参数根据第二个参数中的 *xpath* 进行匹配。如果 WITH 子句表达式匹配多个节点，则只有文档顺序中的第一个节点被使用。如果该节点不是文本节点，则通过附加所有下级文本节点来获得结果。如果一个 WITH 子句表达式不与任何节点匹配，则该行的列为 NULL。

  openxml WITH 子句的语法与从存储过程中进行选择的语法类似。

  有关从存储过程中选择的信息，请参见“FROM 子句”一节第 582 页。
- **column-name** 结果集中列的名称。
- **column-type** 结果集中列的数据类型。该数据类型必须与从 XML 文档中选择的值相兼容。请参见“SQL 数据类型”第 75 页。

用法

openxml 系统过程分析 *xml_data* 并将结果采用树的形式建模。每个元素、属性、文本节点或其它 XML 构造在树中都有单独的节点。提供给 openxml 系统过程的 XPath 查询用来从树中选择节点，然后所选择的节点被映射到结果集中。

openxml 系统过程所使用的 XML 分析器是非校验的，该分析器不读取外部 DTD 子集或外部参数实体。

当出现列表达式的多个匹配时，使用按照文档中的顺序（被分析之前的原始 XML 文档的顺序）的第一个匹配。如果没有匹配的节点，则返回 NULL。当选择了一个内部节点时，返回结果为内部节点的所有下级文本节点的连接值。

类型为 BINARY、LONG BINARY、IMAGE 和 VARBINARY 的列被假定采用 base64 编码格式并自动进行解码。如果使用 FOR XML 子句生成 XML，则上述类型的数据为 base64 编码并且可以使用 openxml 系统过程将其解码。请参见“FOR XML 和二进制数据”一节 《SQL Anywhere 服务器 - SQL 的用法》。

openxml 系统过程支持下列的 XPath 语法的子集：

- 完全支持 child、self、attribute、descendant、descendant-or-self 和 parent axes 语法。
- 在所有支持的功能中均可使用缩写的和未缩写形式的语法。例如，`'a'` 等效于 `'child::a'`，而 `'..'` 等效于 `'parent::node()'`。
- 在名称测试中可使用通配符。例如，`'a/*/b'`。
- 支持以下类型测试：node()、text()、processing-instruction() 和 comment()。
- 可使用 *expr1*[*expr2*] 和 *expr1*[*expr2*="*string*"] 形式的限定符，其中 *expr2* 为所支持的任意 XPath 表达式。如果 *expr2* 匹配一个或多个节点，则限定符的值为 TRUE。例如，`'a[b]'` 查找至少具有一个 `a` 子项的 `b` 节点，而 `a[b="I"]` 查找至少具有一个文本值为 `I` 的 `b` 子项的 `a` 节点。

另请参见

- “使用 XPath 表达式”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “使用 openxml 导入 XML”一节 《SQL Anywhere 服务器 - SQL 的用法》
- XPath 查询语言：http://www.w3.org/TR/xpath。

示例

以下查询从作为 openxml 系统过程的第一个参数提供的 XML 文档中生成结果集：

```
SELECT * FROM openxml( '<products>
                  <ProductType ID="301">Tee Shirt</ProductType>
                  <ProductType ID="401">Baseball Cap</ProductType>
                  </products>',
                  '/products/ProductType' )
WITH ( ProductName LONG VARCHAR 'text()', ProductID CHAR(3) '@ID');
```

此查询会生成以下结果：

ProductName	ProductID
Tee Shirt	301
Baseball Cap	401

在以下示例中，第一个 <ProductType> 元素包含一个实体。当执行查询时，将此节点分析为具有四个子项的元素，其四个子项为：Tee、&、Sweater 和 Set。可使用 . 在结果集中将子项连接起来。

```
SELECT * FROM openxml( '<products>
                  <ProductType ID="301">Tee &amp; Sweater Set</ProductType>
                  <ProductType ID="401">Baseball Cap</ProductType>
                  </products>',
                  '/products/ProductType' )
WITH ( ProductName LONG VARCHAR '.', ProductID CHAR(3) '@ID');
```

此查询会生成以下结果：

ProductName	ProductID
Tee Shirt & Sweater Set	301
Baseball Cap	401

以下查询使用一个相等谓语来从所提供的 XML 文档中生成结果集。

```
SELECT * FROM openxml('<EmployeeDirectory>
   <Employee>
      <column name="EmployeeID">105</column>
      <column name="GivenName">Matthew</column>
      <column name="Surname">Cobb</column>
      <column name="Street">7 Pleasant Street</column>
      <column name="City">Grimsby</column>
      <column name="State">UT</column>
      <column name="PostalCode">02154</column>
      <column name="Phone">6175553840</column>
   </Employee>
```

```
  <Employee>
      <column name="EmployeeID">148</column>
      <column name="GivenName">Julie</column>
      <column name="Surname">Jordan</column>
      <column name="Street">1244 Great Plain Avenue</column>
      <column name="City">Woodbridge</column>
      <column name="State">AZ</column>
      <column name="PostalCode">01890</column>
      <column name="Phone">6175557835</column>
   </Employee>
  <Employee>
      <column name="EmployeeID">160</column>
      <column name="GivenName">Robert</column>
      <column name="Surname">Breault</column>
      <column name="Street">358 Cherry Street</column>
      <column name="City">Milton</column>
      <column name="State">PA</column>
      <column name="PostalCode">02186</column>
      <column name="Phone">6175553099</column>
   </Employee>
  <Employee>
      <column name="EmployeeID">243</column>
      <column name="GivenName">Natasha</column>
      <column name="Surname">Shishov</column>
      <column name="Street">151 Milk Street</column>
      <column name="City">Grimsby</column>
      <column name="State">UT</column>
      <column name="PostalCode">02154</column>
      <column name="Phone">6175552755</column>
   </Employee>
</EmployeeDirectory>', '/EmployeeDirectory/Employee')
WITH ( EmployeeID INT 'column[@name="EmployeeID"]',
       GivenName    CHAR(20) 'column[@name="GivenName"]',
       Surname      CHAR(20) 'column[@name="Surname"]',
       PhoneNumber  CHAR(10) 'column[@name="Phone"]');
```

此查询会生成以下结果集：

EmployeeID	GivenName	Surname	PhoneNumber
105	Matthew	Cobb	6175553840
148	Julie	Jordan	6175557835
160	Robert	Breault	6175553099
243	Natasha	Shishov	6175552755

以下查询使用 XPath @attribute 表达式来生成结果集：

```
SELECT * FROM openxml( '<Employee
      EmployeeID="105"
      GivenName="Matthew"
      Surname="Cobb"
      Street="7 Pleasant Street"
      City="Grimsby"
      State="UT"
      PostalCode="02154"
      Phone="6175553840"
/>', '/Employee' )
```

```
WITH ( EmployeeID INT '@EmployeeID',
       GivenName    CHAR(20) '@GivenName',
       Surname      CHAR(20) '@Surname',
       PhoneNumber  CHAR(10) '@Phone');
```

以下查询操作的 XML 文档与上面查询中使用的 XML 文档类似，只是引入了 XML 命名空间。它演示了如何在 XPath 查询的名称测试中使用通配符，并生成与上面查询相同的结果集。

```
SELECT * FROM openxml( '<Employee  xmlns="http://www.iAnywhere.com/
EmployeeDemo"
      EmployeeID="105"
      GivenName="Matthew"
      Surname="Cobb"
      Street="7 Pleasant Street"
      City="Grimsby"
      State="UT"
      PostalCode="02154"
      Phone="6175553840"
/>', '/*:Employee' )

WITH ( EmployeeID INT '@EmployeeID',
       GivenName    CHAR(20) '@GivenName',
       Surname      CHAR(20) '@Surname',
       PhoneNumber  CHAR(10) '@Phone');
```

或者，也可以指定命名空间声明：

```
SELECT * FROM openxml( '<Employee  xmlns="http://www.iAnywhere.com/
EmployeeDemo"
      EmployeeID="105"
      GivenName="Matthew"
      Surname="Cobb"
      Street="7 Pleasant Street"
      City="Grimsby"
      State="UT"
      PostalCode="02154"
      Phone="6175553840"
/>', '/prefix:Employee', 1, '<r xmlns:prefix="http://www.iAnywhere.com/
EmployeeDemo"/>' )
WITH ( EmployeeID INT '@EmployeeID',
       GivenName    CHAR(20) '@GivenName',
       Surname      CHAR(20) '@Surname',
       PhoneNumber  CHAR(10) '@Phone');
```

有关使用 openxml 系统过程的更多示例，请参见 “使用 openxml 导入 XML”一节 《SQL Anywhere 服务器 - SQL 的用法》。

sa_ansi_standard_packages 系统过程

返回有关 SQL 语句中使用的非核心 SQL 扩充的信息？？？。

语法

sa_ansi_standard_packages(*sql-standard-string*, *sql-statement-string* **)**

参数

- **sql-standard-string**　该标准用于核心扩展。SQL:1999 或 SQL:2003 其中之一。

- **sql-statement-string** 要计算的 SQL 语句。

注释

如果存在用于语句的非核心扩展，则结果集为空。

权限

无

副作用

无

另请参见

- “SQL 预处理器”一节 《SQL Anywhere 服务器 - 编程》
- “SQLFLAGGER 函数 [Miscellaneous]”一节第 298 页
- “sql_flagger_error_level 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sql_flagger_warning_level 选项 [兼容性]”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下是调用 sa_ansi_standard_packages 系统过程的示例：

```
CALL sa_ansi_standard_packages( 'SQL:2003',
'SELECT *
   FROM ( SELECT o.SalesRepresentative,
                 o.Region,
                 SUM( s.Quantity * p.UnitPrice ) AS total_sales,
                 DENSE_RANK() OVER ( PARTITION BY o.Region,
                                     GROUPING( o.SalesRepresentative )
                                     ORDER BY total_sales DESC ) AS
sales_rank
            FROM Product p, SalesOrderItems s, SalesOrders o
            WHERE p.ID = s.ProductID AND s.ID = o.ID
            GROUP BY GROUPING SETS( ( o.SalesRepresentative, o.Region ),
o.Region ) ) AS DT
   WHERE sales_rank <= 3
   ORDER BY Region, sales_rank');
```

此查询会生成以下结果集：

package_id	package_name
T612	高级 OLAP 操作
T611	基础 OLAP 操作
F591	派生表
T431	扩展的分组功能

sa_audit_string 系统过程

将字符串添加到事务日志中。

语法

sa_audit_string(*string* **)**

参数

- **string** 要添加到事务日志中的字符串。

注释

如果审记功能是打开的，此系统过程会向存储在事务日志中的审计信息添加一条注释。字符串的最大长度可以是 200 个字节。

权限

需要 DBA 权限

副作用

无

另请参见

- “auditing 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “审计数据库活动”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下示例使用 sa_audit_string 向审计日志添加一条注释：

```
CALL sa_audit_string( 'Auditing test' );
```

sa_char_terms 系统过程

将 CHAR 字符串分解为多个语词，然后返回每个语词及其位置。

语法

sa_char_terms('*char-string*' [, '*text-config-name*' [, '*owner*']]]
)

参数

- **char-string** 所分析的 CHAR 字符串。
- **text-config-name** 处理字符串时应用的文本配置对象。缺省值为 'default_char'。
- **owner** 所指定的文本配置对象的所有者。缺省值为 DBA。

注释

可以使用此系统过程了解应用文本配置对象的设置时，会如何对字符串进行解释。如果想知道索引过程中哪些术语会被删除或者会从查询字符串中删除哪些术语，这一点很有用。

权限

无

副作用

None

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本配置对象”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_nchar_terms 系统过程”一节第 866 页

示例

以下语句返回 CHAR 字符串 'the quick brown fox jumped over the fence' 中的语词：

```
CALL sa_char_terms ( 'the quick brown fox jumped over the fence' );
```

term	position
the	1
quick	2
brown	3
fox	4
jumped	5
over	6
the	7
fence	8

sa_check_commit 系统过程

在提交前检查是否存在未完成的参照完整性违规。

语法

sa_check_commit(
tname,

keyname
)

参数

- **tname** VARCHAR(128) 参数，其中包含有一个行当前违反了参照完整性的表的名称。
- **keyname** VARCHAR(128) 参数，其中包含相应外键索引的名称。

注释

如果数据库选项 wait_for_commit 为 On，或者在 CREATE TABLE 语句中使用 CHECK ON COMMIT 定义了外键，则可以更新数据库，而如果在提交更改前解决了违规问题，还会引起参照完整性违规。

您可以使用 sa_check_commit 系统过程，在试图提交更改前检查是否存在未完成的参照完整性违规情况。

返回的参数表示某行当前违反参照完整性的表的名称和对应的外键索引名。

权限

无

副作用

无

另请参见

- “wait_for_commit 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “CREATE TABLE 语句”一节第 497 页

示例

以下命令集可以通过 Interactive SQL 执行。示例数据库的 Departments 表中删除了若干行，致使出现参照完整性违规。sa_check_commit 系统过程调用将检查哪些表和键有未解决的违规，然后回退以取消此更改：

```
SET TEMPORARY OPTION wait_for_commit='On'
go
DELETE FROM Departments
go
CREATE VARIABLE tname VARCHAR( 128 );
CREATE VARIABLE keyname VARCHAR( 128 )
go
CALL sa_check_commit( tname, keyname )
go
SELECT tname, keyname
go
ROLLBACK
go
```

sa_clean_database 系统过程

启动数据库清理程序，并设置可运行的最长时间。

语法

sa_clean_database([*duration*] **)**

参数

- **duration** 清理操作允许运行的秒数。如果未指定参数或指定的参数为 0，则数据库清理程序将一直运行，直至所有 dbspace 中的所有页均已清理。

注释

数据库清理程序是在缺省调度上运行的内部任务。可使用此系统过程强制数据库清理程序立即运行，以及指定每次调用清理程序时其可运行的时间。

如果将请求的某些部分推迟到以后执行，则某些数据库任务（例如处理快照隔离事务、索引维护和删除行）的执行会更为有效。这些可推迟活动通常涉及通过从数据库页中删除已删除条目、历史条目和其它不必要的条目，或重组数据库页以便更有效访问而进行的清理。

推迟一些这样的活动不仅可使当前请求更快地完成，还可能允许在数据库服务器不太忙时执行清理操作。这些不必要的条目被标识，以便其它事务看不到它们；然而，它们确实占用页上的空间，在某些时候必须删除。

数据库清理程序执行任何推迟的清理活动。安排每 20 秒钟运行一次清理程序。调用清理程序时，数据库按顺序循环通过数据库的 dbspace，检查和清理每个可清理页，然后进入下 一个可清理页。当数据库清理程序由数据库服务器自动调用时，其为自我调节过程。数据库清理程序执行的工作量和执行的持续时间取决于几个因素，包括 dbspace 中未完成的可清理页的百分率、数据库服务器中当前的活动数量以及数据库清理程序已花费的清理时间量。如果在运行 0.5 秒后，清理程序检测到服务器中有活动请求，它将停止运行并重新调度其自己以定期执行。当服务器中没有其它请求执行时，数据库清理程序将尝试处理页，因此充分利用了服务器的非活动期间。

数据库清理程序统计信息可通过以下四个数据库属性来获取：

- **CleanablePagesAdded** 返回需要进行清理的页数
- **CleanablePagesCleaned** 返回已进行清理的页数
- **CleanableRowsAdded** 返回需要进行清理的行数
- **CleanabledRowsCleaned** 返回已进行清理的行数

值 CleanablePagesAdded 与 CleanablePagesCleaned 间的差值指明仍需进行清理的数据库页数。

可使用 sa_clean_database 系统过程来配置数据库清理程序，以运行直至数据库中的所有页均已进行清理，或指定数据库清理程序运行的最大持续时间。

要进一步自定义数据库清理程序的行为，可设置一个事件，当需要进行清理的页或行数超过指定阈值时，通过该事件启动数据库清理程序。请参见“CREATE EVENT 语句”一节第 429 页。

权限

需要 DBA 权限

副作用

无

另请参见

- “CREATE EVENT 语句”一节第 429 页
- CleanablePagesAdded、CleanablePagesCleaned、CleanableRowsAdded 和 CleanableRowsCleaned 属性： “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下示例将数据库清理程序的持续时间设置为 10 秒：

```
CALL sa_clean_database( 10 );
```

以下示例创建了一个调度事件，该事件每天运行以允许数据库清理程序运行，直至数据库中的所有页均已进行清理：

```
CREATE EVENT DailyDatabaseCleanup
SCHEDULE
   START TIME '6:00 pm'
   ON ( 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday' )
   HANDLER
      BEGIN
         CALL sa_clean_database( );
      END;
```

以下示例在数据库中有不少于 20% 的页需要进行清理时强制数据库清理程序运行：

```
CREATE EVENT PERIODIC_CLEANER
SCHEDULE
BETWEEN '9:00 am' and '5:00 pm'
EVERY 1 HOURS
HANDLER
BEGIN
     DECLARE @num_db_pages INTEGER;
     DECLARE @num_dirty_pages INTEGER;

     -- Get the number of database pages
     SELECT (SUM( DB_EXTENDED_PROPERTY( 'FileSize', t.dbspace_id ) -
                    DB_EXTENDED_PROPERTY( 'FreePages', t.dbspace_id ) ))
     INTO @num_db_pages
     FROM (SELECT dbspace_id FROM SYSDBSPACE) AS t;

     -- Get the number of dirty pages to be cleaned
     SELECT (DB_PROPERTY( 'CleanablePagesAdded' ) -
                    DB_PROPERTY( 'CleanablePagesCleaned' ))
     INTO @num_dirty_pages;

     -- Check whether the number of dirty pages exceeds 20% of
     -- the size of the database
     IF @num_dirty_pages > @num_db_pages * 0.20 THEN
       -- Start cleaning the database for a maximum of 60 seconds
       CALL sa_clean_database( 60 );
     END IF;
END;
```

sa_column_stats 系统过程

返回有关指定列的各种统计信息。这些统计信息与维护的供优化程序使用的列统计信息无关。

语法

sa_column_stats (
[*tab_name*]
[, *col_name*]
[, *tab_owner*]
[, *max_rows*]
)

参数

- **tab_name** 此可选 CHAR(128) 参数指定表的名称。如果未指定此参数，则计算所有表中所有列的统计信息。
- **col_name** 此可选 CHAR(128) 参数指定要计算统计信息的列。如果未指定此参数，则计算指定表中所有列的统计信息。
- **tab_owner** 此可选 CHAR(128) 参数指定表的所有者。如果未指定此参数，则数据库服务器使用与指定的 *tab_name* 相匹配的第一个表的所有者。
- **max_rows** 此可选 INTEGER 参数指定要用于计算的行数。如果未指定此参数，则缺省情况下使用 1000 行。指定为 0 则表示数据库服务器根据表中的所有行计算比率。

结果集

除 table_owner、table_name 和 column_name 外，对于非字符串列，结果集中的所有值均为 NULL。此外对于空表，num_rows_processed 和 num_values_compressed 为 0，而所有其它值为 NULL。

列名	数据类型	说明
table_owner	CHAR(128)	表的所有者。
table_name	CHAR(128)	表名。
column_name	CHAR(128)	列名称。
num_rows_processed	INTEGER	读取的用来计算统计信息的总行数。
num_values_compressed	INTEGER	在列中被压缩的值的数量。如果列未压缩，则该值为 0。
avg_compression_ratio	DOUBLE	平均压缩率，表示为减少大小的百分比，针对列中的压缩值。如果列未压缩，则该值为 NULL。
avg_length	DOUBLE	列中所有非 NULL 字符串的平均长度。
stddev_length	DOUBLE	列中所有非 NULL 字符串长度的标准差。
min_length	INTEGER	列中非 NULL 字符串的最小长度。
max_length	INTEGER	列中字符串的最大长度。
avg_uncompressed_length	DOUBLE	列中所有未压缩的非 NULL 字符串的平均长度。

列名	数据类型	说明
stddev_uncompressed_length	DOUBLE	列中所有未压缩的非 NULL 字符串长度的标准差。
min_uncompressed_length	INTEGER	列中所有未压缩的非 NULL 字符串的最小长度。
max_uncompressed_length	INTEGER	列中所有未压缩的非 NULL 字符串的最大长度。

注释

数据库服务器确定与指定的所有者、表和列名相匹配的列，然后对于每一列，计算每个指定列中的数据的统计信息。缺省情况下，数据库服务器仅使用前 1000 行的数据。

对于 avg_compression_ratio，值不能大于或等于 100，然而如果将高度不可压缩的数据（例如已压缩的数据）插入到压缩列中，该值可能小于 0。值越高表示压缩越好。例如，如果返回值为 80，则压缩后数据的大小比未压缩数据的大小小 80%。

权限

需要 DBA 权限

副作用

无

另请参见

- “选择是否压缩列”一节 《SQL Anywhere 服务器 - SQL 的用法》

示例

在此示例中，在 SELECT 语句中使用 sa_column_stats 系统过程来确定数据库中压缩率最高的那些列：

```
SELECT * FROM sa_column_stats()
  WHERE num_values_compressed > 0
  ORDER BY avg_compression_ratio desc;
```

在此示例中，将选择范围从上一个示例缩小到由 bsmith 拥有的表：

```
SELECT * FROM sa_column_stats( tab_owner='bsmith' )
  WHERE num_values_compressed > 0
  ORDER BY avg_compression_ratio desc;
```

sa_conn_activity 系统过程

为服务器上的每个指定数据库连接返回一个最近预准备 SQL 语句。

语法

sa_conn_activity([*connidparm*] **)**

参数

- **connidparm** 使用此可选的 INTEGER 参数指定连接的 ID 号。

结果集

列名	数据类型	说明
Number	INT	连接的 ID 号。
Name	VARCHAR(255)	连接的名称。
Userid	VARCHAR(255)	连接的用户 ID。
DBNumber	INT	数据库的 ID 编号。
LastReqTime	VARCHAR(255)	指定连接的最后请求开始的时间。
LastStatement	LONG VARCHAR	连接的最近预准备 SQL 语句。

注释

sa_conn_activity 系统过程为每个连接都返回一个包含最近预准备 SQL 语句的结果集（前提是已通知服务器搜集该信息）。在调用 sa_conn_activity 之前，必须为数据库服务器启用语句记录。要达到此目的，可在启动数据库服务器时指定 -zl 选项，也可执行以下内容：

```
CALL sa_server_option('RememberLastStatement','ON');
```

当数据库服务器忙而您想获取为每个连接预准备的最近 SQL 语句的信息时，此过程很有用。此功能可以替代请求记录。

有关派生出这些值的 LastStatement 属性的信息，请参见“连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》。

如果未指定 *connidparm*，则为数据库服务器上运行的所有数据库的所有连接返回信息。如果 *connidparm* 小于零，则返回当前连接的选项值。

权限

无

副作用

无

另请参见

- “-zl 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_server_option 系统过程”一节第 886 页

sa_conn_compression_info 系统过程

提供通信压缩比摘要信息。

语法

sa_conn_compression_info([*connidparm*] **)**

参数

- **connidparm** 使用此可选的 INTEGER 参数指定连接的 ID 号。

结果集

列名	数据类型	说明
Type	VARCHAR(20)	一个字符串，用于确定其后面的压缩统计信息是表示一个 Connection 还是到 Server 的所有连接。
ConnNumber	INTEGER	一个表示连接 ID 的 INTEGER 类型值。如果 Type 是 Server，则返回 NULL。
Compression	VARCHAR(10)	一个字符串，表示连接是否启用了压缩。如果 Type 为 Server，则返回 NULL，如果 Type 为 Connection，则返回 ON/OFF。
TotalBytes	INTEGER	一个 INTEGER 类型值，表示发送和收到的实际字节总数。
TotalBytesUnComp	INTEGER	一个 INTEGER 类型值，表示当禁用压缩时本来会发送和收到的字节数。
CompRate	NUMERIC(5,2)	一个 NUMERIC (5,2) 类型值，表示总压缩比。例如，值 0 表示没有发生压缩。值 75 表示数据压缩比是 75%，即一直压缩到原大小的四分之一。
CompRateSent	NUMERIC(5,2)	一个 NUMERIC (5,2) 类型值，表示发送到客户端的数据的压缩比。
CompRateReceived	NUMERIC(5,2)	一个 NUMERIC (5,2) 类型值，表示从客户端收到的数据的压缩比。
TotalPackets	INTEGER	一个 INTEGER 类型值，表示发送和收到的实际数据包总数。
TotalPacketsUnComp	INTEGER	一个 INTEGER 类型值，表示当禁用压缩时本来会发送和收到的数据包总数。
CompPktRate	NUMERIC(5,2)	一个 NUMERIC (5,2) 类型值，表示数据包的总压缩比。

列名	数据类型	说明
CompPktRateSent	NUMERIC(5,2)	一个 NUMERIC (5,2) 类型值，表示发送到客户端的数据包的压缩比。
CompPktRateReceived	NUMERIC(5,2)	一个 NUMERIC (5,2) 类型值，表示从客户端收到的数据包的压缩比。

注释

如果指定连接 ID 号，则 sa_conn_compression_info 系统过程为提供的连接返回一个包含 compression 属性的结果集。如果没有提供 *connection-id*，则此系统过程返回到服务器上数据库的所有当前连接的信息。

有关派生出这些值的属性的信息，请参见“连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

无

副作用

无

示例

以下示例使用 sa_conn_compression_info 系统过程为连接到服务器的所有连接返回一个汇总其 compression 属性的结果集。

```
CALL sa_conn_compression_info( );
```

Type	ConnNumber	Compression	TotalBytes	...
Connection	79	Off	7841	...
Server	(NULL)	(NULL)	2737761	...
...	...	...	...	...

sa_conn_info 系统过程

报告连接属性信息。

语法

sa_conn_info([*connidparm*] **)**

参数

- **connidparm** 此可选 INTEGER 参数指定连接的 ID 号。

结果集

列名	数据类型	说明
Number	INTEGER	连接的 ID 号。
Name	VARCHAR(255)	连接的名称。
Userid	VARCHAR(255)	连接的用户 ID。
DBNumber	INTEGER	数据库的 ID 号。
LastReqTime	VARCHAR(255)	指定连接的最后请求开始的时间。
ReqType	VARCHAR(255)	上次请求类型的字符串。
CommLink	VARCHAR(255)	连接的通信链接。这是 SQL Anywhere 所支持的网络协议之一，或者对于相同计算机连接为 local。
NodeAddr	VARCHAR(255)	客户端/服务器连接中客户端的地址。
ClientPort	INTEGER	客户端应用程序进行 TCP/IP 通信的端口号。
ServerPort	INTEGER	服务器进行 TCP/IP 通信的端口号。
BlockedOn	INTEGER	如果没有阻塞当前连接，则该值为 0。如果阻塞了当前连接，则为由于锁定冲突而阻塞连接的连接数。
LockTable	VARCHAR(255)	如果此连接当前正等待一个锁，LockTable 将成为与该锁相关联的表的名称。否则，LockTable 将是空字符串。
UncommitOps	INTEGER	未提交操作的数量。
LockRowID	UNSIGNED BIGINT	如果连接正在等待与特定行标识符相关联的锁，LockRowID 会包含此行标识符。如果连接未等待与行相关联的锁（即连接未等待锁或者它所等待的锁没有相关联的行），则 LockRowID 为 NULL。
LockIndexID	INTEGER	如果连接正在等待与特定索引相关联的锁，则 LockIndexID 会包含该索引的标识符（如果该锁与 LockTable 中表上的所有索引相关联，则会包含值 -1）。如果连接未等待与索引相关联的锁（即连接未等待锁或者它所等待的锁没有相关联的索引），则 LockIndexID 为 NULL。

注释

如果指定连接 ID 号，则 sa_conn_info 系统过程为提供的连接返回一个包含连接属性的结果集。如果没有提供 *connidparm*，则此系统过程返回服务器上数据库的所有当前连接的信息。如果 *connidparm* 小于零，则返回当前连接的选项值。

当发生阻塞时，使用此过程所返回的 BlockedOn 值可以检查哪些用户被阻塞以及是谁阻塞的。sa_locks 系统过程可用于显示阻塞的连接所持有的锁。

有关基于任意这些属性的详细信息，可执行与以下类似的内容：

```
SELECT *, DB_NAME( DBNumber ),
   CONNECTION_PROPERTY( 'LastStatement', Number )
   FROM sa_conn_info( );
```

LockRowID 的值可用于在 sa_locks 过程的输出中查找锁。

LockIndexID 的值可用于在 sa_locks 过程的输出中查找锁。此外，LockIndexID 的值对应于 ISYSIDX 系统表的主键，可使用 SYSIDX 系统视图查看此系统表。

每个锁都具有关联的表，因此，LockTable 的值可用于明确地确定某连接是否正在等待锁。

权限

无

副作用

None

另请参见

- “连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_locks 系统过程”一节第 846 页
- “SYSIDX 系统视图”一节第 952 页

示例

以下示例使用 sa_conn_info 系统过程为连接到服务器的所有连接返回一个汇总连接属性的结果集。

```
CALL sa_conn_info( );
```

Number	Name	Userid	DBNumber	...
79		DBA	0	...
46	Sybase Central 1	DBA	0	...
...	...	...	...	...

sa_conn_list 系统过程

返回包含连接 ID 的结果集。

语法

sa_conn_list(
[*connidparm*]
[, *dbidparm*]
)

参数

- **connidparm** 使用此可选的 INTEGER 参数指定连接的 ID 号。
- **dbidparm** 使用此可选的 INTEGER 参数指定数据库的 ID 号。

结果集

列名	数据类型	说明
Number	INTEGER	连接的 ID 号。

注释

如果未指定任何参数，或两个参数均为 NULL，则为在数据库服务器上运行的所有数据库的所有连接返回连接 ID。如果 *connidparm* 小于 0，则只返回当前连接的连接 ID。如果 *connidparm* 为 NULL 且 *dbidparm* 小于 0，则只返回当前数据库的连接 ID。如果 *connidparm* 为 NULL，同时 *dbidparm* 不为 NULL 且其值大于或等于 0，则只返回该数据库的连接 ID。

权限

无

副作用

无

另请参见

- “sa_db_list 系统过程”一节第 815 页
- “sa_conn_options 系统过程”一节第 810 页

sa_conn_options 系统过程

返回对应于数据库选项的连接属性的属性信息。

语法

sa_conn_options([*connidparm*] **)**

参数

- **connidparm** 使用此可选的 INTEGER 参数指定连接的 ID 号。

结果集

列名	数据类型	说明
Number	INTEGER	连接的 ID 号。
PropNum	INTEGER	连接属性编号。

列名	数据类型	说明
OptionName	VARCHAR(255)	选项名称。
OptionDescription	VARCHAR(255)	选项说明。
Value	LONG VARCHAR	选项值。

注释

以 Number 的形式返回连接 ID，并返回对应于数据库选项的每个可用连接属性的 PropNum、OptionName、OptionDescription 和 Value。

如果未指定 *connidparm*，则返回当前数据库的所有连接的选项值。如果 *connidparm* 小于零，则返回当前连接的选项值。

权限

无

副作用

无

另请参见

- “sa_db_list 系统过程”一节第 815 页
- “sa_conn_list 系统过程”一节第 809 页

sa_conn_properties 系统过程

报告连接属性信息。

语法

sa_conn_properties([*connidparm*] **)**

参数

- **connidparm** 使用此可选的 INTEGER 参数指定连接的 ID 号。

结果集

列名	数据类型	说明
Number	INTEGER	连接的 ID 号。
PropNum	INTEGER	连接属性编号。
PropName	VARCHAR(255)	连接属性名称。

列名	数据类型	说明
PropDescription	VARCHAR(255)	连接属性说明。
Value	LONG VARCHAR	连接属性值。

注释

以 Number 的形式返回连接 ID，并返回每个可用连接属性的 PropNum、PropName、PropDescription 和 Value。返回所有连接属性、与连接相关的数据库选项设置以及与连接相关的统计信息的值。同时返回具有 NULL 值的有效属性。

如果没有提供 *connidparm*，则返回当前数据库的所有连接的属性。如果 *connidparm* 小于零，则返回当前连接的选项值。

权限

无

副作用

无

另请参见

- “sa_conn_list 系统过程”一节第 809 页
- “sa_conn_options 系统过程”一节第 810 页
- “系统函数”一节第 127 页
- “连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下示例使用 sa_conn_properties 系统过程为所有连接返回一个汇总连接属性信息的结果集。

```
CALL sa_conn_properties( );
```

Number	PropNum	PropName	...
79	37	CacheHits	...
79	38	CacheRead	...
...	...	...	...

以下示例使用 sa_conn_properties 系统过程返回所有连接的列表，并按 CPU 时间降序排序*：

```
SELECT Number AS connection_number,
    CONNECTION_PROPERTY ( 'Name', Number ) AS connection_name,
    CONNECTION_PROPERTY ( 'Userid', Number ) AS user_id,
  CAST ( Value AS NUMERIC ( 30, 2 ) ) AS approx_cpu_time
  FROM sa_conn_properties()
  WHERE PropName = 'ApproximateCPUTime'
  ORDER BY approx_cpu_time DESC;
```


*经 Breck Carter 允许使用的示例，引自 RisingRoad Professional Services，网址为 http://www.risingroad.com。

sa_convert_ml_progress_to_timestamp 系统过程

仅用于 MobiLink 脚本式上载。它将脚本式上载的进度值从 64 位的 INTEGER 类型值转换为 TIMESTAMP 类型值。

语法

sa_convert_ml_progress_to_timestamp(*progress* **)**

参数

- **progress** 该函数采用一个 UNSIGNED BIGINT 参数。

注释

该函数返回由传入值表示的 TIMESTAMP 值。此过程为 sa_convert_timestamp_to_ml_progress 的逆过程。

权限

无

副作用

无

另请参见

- “sa_convert_timestamp_to_ml_progress 系统过程”一节第 813 页
- “脚本式上载” 《MobiLink - 客户端管理》

示例

```
SELECT sa_convert_timestamp_to_ml_progress( 3600000 );
```

sa_convert_timestamp_to_ml_progress 系统过程

仅用于 MobiLink 脚本式上载。它将脚本式上载的进度值从 TIMESTAMP 类型转换为 64 位 UNISIGNED BIGINT 类型。

语法

sa_convert_timestamp_to_ml_progress([*t1*] **)**

参数

- **t1** 使用此可选的 TIMESTAMP 参数指定要转换为 64 位 UNISIGNED BIGINT 类型的进度值。

注释

该函数返回由作为参数传入的时间戳表示的 UNISIGNED BIGINT 值。此过程为 sa_convert_ml_progress_to_timestamp 的逆过程。

权限

无

副作用

无

另请参见

- “sa_convert_ml_progress_to_timestamp 系统过程”一节第 813 页
- “脚本式上载” 《MobiLink - 客户端管理》

示例

```
SELECT sa_convert_timestamp_to_ml_progress( CURRENT TIMESTAMP );

SELECT sa_convert_timestamp_to_ml_progress( '1900/01/01 1:00' );
```

sa_db_info 系统过程

报告数据库属性信息。

语法

sa_db_info([*dbidparm*] **)**

参数

- **dbidparm** 使用此可选的 INTEGER 参数指定数据库的 ID 号。

结果集

列名	数据类型	说明
Number	INTEGER	连接的 ID 号。
Alias	VARCHAR(255)	数据库名称。
File	VARCHAR(255)	数据库根文件的文件名，包括路径。
ConnCount	INTEGER	数据库的连接数量。
PageSize	INTEGER	数据库的页大小，以字节表示。
LogName	VARCHAR(255)	事务日志的文件名，包括路径。

注释

如果指定数据库 ID，则 sa_db_info 返回单个行，其中包含指定数据库的 Number、Alias、File、ConnCount、PageSize 和 LogName。

如果没有提供 *dbidparm*，则返回所有数据库的属性。

权限

无

副作用

无

另请参见

- “sa_db_properties 系统过程”一节第 816 页
- “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下语句为服务器上运行的每个数据库返回一行：

```
CALL sa_db_info( );
```

属性	值
Number	0
Alias	demo
File	C:\Documents and Settings\All Users\Documents\SQL Anywhere 11\Samples\demo.db
ConnCount	1
PageSize	4096
LogName	C:\Documents and Settings\All Users\Documents\SQL Anywhere 11\Samples\demo.log

sa_db_list 系统过程

返回数据库 ID。

语法

sa_db_list([*dbidparm*] **)**

参数

- **dbidparm** 使用此可选的 INTEGER 参数指定数据库的 ID 号。

结果集

列名	数据类型	说明
Number	INTEGER	数据库的 ID 号。

注释

如果未指定 *dbidparm*，或者 *dbidparm* 为 NULL，则返回在数据库服务器上运行的所有数据库的 ID。如果 *dbidparm* 小于 0，则只返回当前数据库的 ID。

权限

无

副作用

无

另请参见

- “sa_conn_list 系统过程”一节第 809 页
- “sa_conn_options 系统过程”一节第 810 页

sa_db_properties 系统过程

报告数据库属性信息。

语法

sa_db_properties([*dbidparm*] **)**

参数

- **dbidparm** 使用此可选的 INTEGER 参数指定数据库的 ID 号。

结果集

列名	数据类型	说明
Number	INTEGER	数据库的 ID 号。
PropNum	INTEGER	数据库属性编号。
PropName	VARCHAR(255)	数据库属性名称。
PropDescription	VARCHAR(255)	数据库属性说明。
Value	LONG VARCHAR	数据库属性值。

注释

如果指定数据库 ID，则 sa_db_properties 系统过程返回数据库 ID 号和每个可用数据库属性的 PropNum、PropName、PropDescription 和 Value。返回所有数据库属性和与数据库相关的统计信息的值。同时返回具有 NULL 值的有效属性。

如果未指定 *dbidparm*，则返回所有数据库的属性。

权限

无

副作用

无

另请参见

- “sa_db_info 系统过程”一节第 814 页
- “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下示例使用 sa_db_properties 系统过程为所有数据库返回一个汇总数据库属性信息的结果集。

```
CALL sa_db_properties( );
```

Number	PropNum	PropName	...
0	0	ConnCount	...
0	1	IdleCheck	...
0	2	IdleWrite	...
...	...	...	...

sa_dependent_views 系统过程

返回给定表或视图的所有相关视图的列表。

语法

sa_dependent_views('*tbl_name* **' [, '** *owner_name* **')**

参数

- **tbl_name**　使用此 CHARACTER 参数指定表或视图的名称。
- **owner_name**　使用此可选的 CHARACTER 参数指定 *tbl_name* 的所有者。

结果集

列名	数据类型	说明
table_id	UNSIGNED INTEGER	表或视图的对象 ID。
dep_view_id	UNSIGNED INTEGER	相关视图的对象 ID。

注释

使用此过程获取相关视图的 ID 的列表。或者，也可以在语句中使用该过程返回有关视图的详细信息，如视图名称。

如果现有的表均不满足为表和所有者名称指定的条件，则不会生成错误。另外：

- *table_name* 为可选项，其缺省值为空。
- 如果 *owner* 和 *table_name* 均为空，则将返回具有相关视图的所有表的相关信息。
- 如果 *table_name* 为空，但指定了 *owner*，则将返回指定所有者拥有的所有表的相关信息。
- 如果指定了 *table_name*，但 *owner* 为空，则将返回具有指定名称的所有表的相关信息。

缺省情况下，执行该过程不需要任何权限，而且假定 PUBLIC 组可以访问该目录。DBA 可以根据需要控制对视图和/或目录的访问。

权限

无

副作用

无

另请参见

- “SYSDEPENDENCY 系统视图”一节第 945 页
- “视图依赖性”一节 《SQL Anywhere 服务器 - SQL 的用法》

示例

在此示例中，sa_dependent_views 系统过程用来获取 SalesOrders 表的相关视图的 ID 列表。该过程为 SalesOrders 返回 table_id，为相关视图 (ViewSalesOrders) 返回 dep_view_id。

```
sa_dependent_views( 'SalesOrders' );
```

在以下示例中，在 SELECT 语句中使用 sa_dependent_views 系统过程以获取 SalesOrders 表的相关视图的名称列表。该过程返回 ViewSalesOrders 视图。

```
SELECT t.table_name FROM SYSTAB t,
sa_dependent_views( 'SalesOrders' ) v
WHERE t.table_id = v.dep_view_id;
```


sa_describe_query 系统过程

描述查询的结果集，且有一行描述查询的每个输出列。

语法

sa_describe_query(
query
[, *add_keys*]
)

参数

- **query** 使用此 LONG VARCHAR 参数指定所描述的 SQL 语句的文本。
- **add_keys** 使用此可选的 BIT 参数指定是否确定一组列，用来唯一标识所描述查询的结果集中的行。缺省值为 0；数据库服务器不尝试确定列。有关此参数的完整说明，请参见下面的注释部分。

结果集

列名	数据类型	说明
column_number	INTEGER	此行描述的列的顺序位置，从 1 开始。
name	VARCHAR(128)	列的名称。
domain_id	SMALLINT	列的数据类型。请参见“SYSDOMAIN 系统视图”一节第 945 页。
domain_name	VARCHAR(128)	数据类型名称。请参见“SYSDOMAIN 系统视图”一节第 945 页。
domain_name_with_size	VARCHAR(160)	数据类型名称，包括大小和精度（如 CREATE TABLE 或 CAST 函数中所用）。
width	INTEGER	字符串参数的长度、数字参数的精度或任何其它数据类型的存储字节数。
scale	INTEGER	数字数据类型列的小数点后的位数，对于所有其它数据类型该值为零。
declared_width	INTEGER	字符串参数的长度、数字参数的精度或任何其它数据类型的存储字节数。
user_type_id	SMALLINT	如果存在用户定义数据类型，则为其 type_id，否则为 NULL。请参见“SYSUSERTYPE 系统视图”一节第 985 页。

列名	数据类型	说明
user_type_name	VARCHAR(128)	如果存在用户定义数据类型,则为其名称，否则为 NULL。请参见“SYSUSERTYPE 系统视图”一节 第 985 页。
correlation_name	VARCHAR(128)	如果提供了与表达式关联的相关名，则为该相关名，否则为 NULL。
base_table_id	UNSIGNED INTEGER	如果表达式是字段，则为 table_id，否则为 NULL。请参见“SYSTAB 系统视图”一节第 973 页。
base_column_id	UNSIGNED INTEGER	如果表达式是字段，则为 column_id，否则为 NULL。请参见“SYSTABCOL 系统视图”一节 第 976 页。
base_owner_name	VARCHAR(128)	如果表达式是字段，则为所有者名称，否则为 NULL。请参见“SYSUSER 系统视图”一节 第 983 页。
base_table_name	VARCHAR(128)	如果表达式是字段，则为表名，否则为 NULL。
base_column_name	VARCHAR(128)	如果表达式是字段，则为列名，否则为 NULL。
nulls_allowed	BIT	一个指示符，如果表达式可以是 NULL，则为 1，否则为 0。
is_autoincrement	BIT	一个指示符，如果表达式是声明为自动增量的列，则为 1，否则为 0。
is_key_column	BIT	一个指示符，如果表达式是结果集的键的一部分，则为 1，否则为 0。有关详细信息，请参见下面的注释部分。
is_added_key_column	BIT	一个指示符，如果表达式是添加的键列，则为 1，否则为 0。有关详细信息，请参见下面的注释部分。

注释

sa_describe_query 过程提供独立于 API 的机制，以描述查询结果集中表达式的名称和类型信息。

当为 *add_keys* 指定 1 时，sa_describe_query 过程尝试从被查询的对象中查找一组列，组合这组列后，可将其用作键以唯一标识所描述查询的结果集中的行。该键采用所查询对象中一个或多个列的形式，且可能包括未在查询中显式引用的列。如果优化程序找到一个键，则将通过使 is_key_column 值为 1 的方式在结果中标识键所使用的一个或多个列。如果未找到键，则返回错误。

对于包括在键中但未在查询中显式引用的任意列，将 is_added_key_column 值设置为 1 表示该列已添加到过程的结果中；否则，is_added_key_column 的值为 0。

如果未指定 *add_keys*，或者指定其值为 0，则优化程序不会尝试查找结果集的键，且 is_key_column 和 is_added_key_column 列均包含 NULL。

declared_width 和 width 值均描述列的大小。declared_width 描述当由 CREATE TABLE 语句或由查询定义列时的列大小，而 width 值给出读取到客户端时字段的大小。客户端的类型表示形式可能与数据库服务器不同。例如，如果将 return_date_time_as_string 选项打开，日期和时间类型则会转换为字符串。对于字符串，声明具有字符长度语义的字段具有与 CREATE TABLE 大小相匹配的 declared_width 值，而 width 值给出存储返回字符串所需的最大字节数。例如：

声明	width	declared_width
CHAR(10)	10	10
CHAR(10 CHAR)	40	10
TIMESTAMP	取决于时间戳格式字符串的长度	8
NUMERIC(10, 3)	10（精度）	10（精度）

权限

无

副作用

无

另请参见

- “EXPRTYPE 函数 [Miscellaneous]”一节第 196 页
- “字符数据类型”一节第 76 页
- “return_date_time_as_string 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下示例介绍查询 Departments 表中的所有列时所返回的信息：

```
SELECT *
FROM sa_describe_query( 'SELECT * FROM Departments DEPT' );
```

结果显示 is_key_column 和 is_added_key_column 的值为 NULL，因为未指定 *add_keys* 参数。

以下示例介绍查询 Employees 表（与 Departments 表连接在一起）的 DepartmentName 和 Surname 列时所返回的信息：

```
SELECT  *
FROM  sa_describe_query( 'SELECT DepartmentName, Surname
 FROM Employees E JOIN Departments D ON E.EmployeeID = D.DepartmentHeadId',
 add_keys = 1 );
```

在结果集的第 3 行和第 4 行所显示的结果为 1，表明需要用来唯一标识查询结果集中行的列是 Employees.EmployeeID 和 Departments.DepartmentID。此外，因为 Employees.EmployeeID 和 Departments.DepartmentID 未在所描述的查询中显式引用，所以在第 3 行和第 4 行的 is_added_key_column 中显示为 1。

sa_disable_auditing_type 系统过程

禁用指定事件的审计。

语法

sa_disable_auditing_type(' *types* ')

参数

- **types** 使用此 VARCHAR(128) 参数指定包含以下一个或多个值的以逗号分隔的字符串：
  - **all** 禁用所有类型的审计。
  - **connect** 禁用成功和失败的连接尝试的审计。
  - **connectFailed** 禁用失败的连接尝试的审计。
  - **DDL** 禁用 DDL 语句的审计。
  - **options** 禁用公共选项的审计。
  - **permission** 禁用权限检查、用户检查和 SETUSER 语句的审计。
  - **permissionDenied** 禁用失败的权限和用户检查审计。
  - **triggers** 禁用响应触发事件的审计。

注释

可使用 sa_disable_auditing_type 系统过程来禁用一个或多个类别信息的审计。

将此选项设置为 all 则禁用所有审计。也可以通过将 PUBLIC.auditing 选项设置为 Off 来禁用审计。

权限

需要 DBA 权限

副作用

无

另请参见

- “sa_enable_auditing_type 系统过程”一节第 824 页
- “审计数据库活动”一节 《SQL Anywhere 服务器 - 数据库管理》
- “auditing 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

禁用所有审计：

```
CALL sa_disable_auditing_type( 'all' );
```


sa_disk_free_space 系统过程

报告 dbspace、事务日志、事务日志镜像和/或临时文件的可用空间信息。

语法

sa_disk_free_space([*p_dbspace_name*] **)**

参数

- **p_dbspace_name** 使用此 VARCHAR(128) 参数指定 dbspace 的名称、事务日志文件、事务日志镜像文件或临时文件。

  如果有一个名为 log、mirror 或 temp 的 dbspace，则可以在关键字前面加上一个下划线。例如，若存在一个名为 log 的 dbspace，则使用 _log 可以获得有关日志文件的信息。

  指定 SYSTEM 以获取有关主数据库文件的信息，指定 TEMPORARY 或 TEMP 以获取有关临时文件的信息，指定 TRANSLOG 以获取有关事务日志的信息，或指定 TRANSLOGMIRROR 以获取有关事务日志镜像的信息。请参见“预定义 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》。

结果集

列名	数据类型	说明
dbspace_name	VARCHAR(128)	这可以是 dbspace 名称、事务日志文件、事务日志镜像文件或临时文件。
free_space	UNSIGNED BIGINT	卷上的空闲字节数。
total_space	UNSIGNED BIGINT	dbspace 所在驱动器的可用磁盘空间的总量。

注释

如果未指定 *p_dbspace_name* 参数或该参数为 NULL，则每个 dbspace 在结果集中都占一行，另外每个事务日志、事务日志镜像和临时文件（如果存在）也都占一行。如果指定了 *p_dbspace_name*，则正好返回一行或零行（如果不存在这样的 dbspace，或者指定了 log 或 mirror 但是没有日志文件或镜像文件，则返回零行）。

有关 SQL Anywhere 数据库的预定义 dbspace 的名称列表，请参见“预定义 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

需要 DBA 权限

副作用

无

示例

以下示例使用 sa_disk_free_space 系统过程返回一个包含可用空间信息的结果集。

```
CALL sa_disk_free_space( );
```

dbspace_name	free_space	total_space
system	10952101888	21410402304
translog	10952101888	21410402304
temporary	10952101888	21410402304

sa_enable_auditing_type 系统过程

启用审计并指定需要审计的事件。

语法

sa_enable_auditing_type('*types* ')

参数

- **types** 使用此 VARCHAR(128) 参数指定包含以下一个或多个值的以逗号分隔的字符串：
  - **all** 启用所有类型的审计。
  - **connect** 启用成功和失败的连接尝试的审计。
  - **connectFailed** 启用失败的连接尝试的审计。
  - **DDL** 启用 DDL 语句审计。
  - **options** 启用公共选项的审计。
  - **permission** 启用权限检查、用户检查和 SETUSER 语句的审计。
  - **permissionDenied** 启用失败的权限和用户检查审计。
  - **触发器** 启用触发事件之后的审计。

注释

sa_enable_auditing_type 与 PUBLIC.auditing 选项配合使用可启用指定类型信息的审计。

如果将 PUBLIC.auditing 选项设置为 On，并且不指定要审计的信息的类型，则会采用缺省设置 (all)。在这种情况下，将记录所有类型的审计信息。

如果将 PUBLIC.auditing 选项设置为 On，并使用 sa_disable_auditing_type 禁用所有类型的审计，则不记录任何审计信息。要想重新建立审计，必须使用 sa_enable_auditing_type 指定想要审计的信息的类型。

如果将 PUBLIC.auditing 选项设置为 Off，则无论 sa_enable_auditing_type 的设置如何，都不记录任何审计信息。

权限

需要 DBA 权限

副作用

无

另请参见

- “sa_disable_auditing_type 系统过程”一节第 822 页
- “审计数据库活动”一节 《SQL Anywhere 服务器 - 数据库管理》
- “auditing 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

若仅启用选项审计：

```
CALL sa_enable_auditing_type( 'options' );
```

sa_eng_properties 系统过程

报告数据库服务器属性信息。

语法

sa_eng_properties( )

结果集

列名	数据类型	说明
PropNum	INTEGER	数据库服务器属性编号。
PropName	VARCHAR(255)	数据库服务器属性名称。
PropDescription	VARCHAR(255)	数据库服务器属性说明。
Value	LONG VARCHAR	数据库服务器属性值。

注释

返回每个可用服务器属性的 PropNum、PropName、PropDescription 和 Value。返回所有数据库服务器属性以及与数据库服务器相关的统计信息的值。有关可用数据库服务器属性的列表，请参见“系统函数”一节第 127 页。

权限

无

副作用

无

另请参见

- “数据库服务器属性”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下语句返回可用服务器属性的集合

```
CALL sa_eng_properties( );
```

PropNum	PropName	...
1	IdleWrite	...
2	IdleChkPt	...
...	...	...

sa_flush_cache 系统过程

清空数据库服务器高速缓存中的当前数据库的所有页。

语法

sa_flush_cache( )

注释

数据库管理员可以使用此过程来清空数据库服务器高速缓存中的当前数据库内容。这在测量性能以确保得到可重复的结果时很有用。

权限

需要 DBA 权限

副作用

无

sa_flush_statistics 系统过程

保存在数据库服务器高速缓存中的所有开销模型统计。

语法

sa_flush_statistics( )

注释

使用此过程将当前高速缓存在数据库中的当前开销模型统计刷新到磁盘。然后，可使用 sa_get_histogram 系统过程或直方图实用程序 (dbhist) 来检索统计。当此系统过程运行时，ISYSCOLSTAT 系统表即会更新。通常操作情况下不需要执行此过程，因为服务器自动定期将统计信息写入磁盘。

权限

需要 DBA 权限

副作用

无

另请参见

- “sa_get_histogram 系统过程”一节第 830 页
- “SYSCOLSTAT 系统视图”一节第 941 页
- “直方图实用程序 (dbhist)”一节 《SQL Anywhere 服务器 - 数据库管理》

sa_get_bits 系统过程

获取一个位字符串，并为该字符串中的每个位返回一行。缺省情况下，仅返回位值为 1 的行。

语法

sa_get_bits(*bit_string* [, *only_on_bits*])

参数

- **bit_string** 使用此 LONG VARBIT 参数指定从中获取位的位字符串。如果该 *bit_string* 参数为 NULL，则不返回任何行。
- **only_on_bits** 使用此可选的 BIT 参数指定是否只返回 1 值位（值为 1 的位）的行。指定 1（缺省值）仅返回一值位的行；指定 0 则为位字符串中的所有位返回行。

结果集

列	数据类型	说明
bitnum	UNSIGNED INT	此行所描述的位的位置。例如，位字符串中第一个位的 bitnum 为 1。
bit_val	BIT	在位置 bitnum 处的位的值。如果 *only_on_bits* 设置为 1，则此值始终为 1。

注释

sa_get_bits 系统过程解码一个位字符串，为该位字符串中的每个位返回一行，表示该位的值。如果 *only_on_bits* 设置为 1（缺省值）或 NULL，则仅返回对应于一值位的行。对于一值位很少的长位字符串，通过优化可有效地处理此情况。如果 *only_on_bits* 设置为 0，则为位字符串中的每个位返回一行。

例如，语句 `CALL sa_get_bits( '1010' )` 返回以下结果集，指明位字符串的位置 1 和位置 3 中的一值位。

bitnum	bit_val
1	1
3	1

sa_get_bits 系统过程可用来将位字符串转换到关系中。这可用来将位字符串与表连接起来，或将位字符串检索为结果集，而不是单个二进制值。如果位字符串中有大量 0 位，则将该字符串检索为结果集会更有效，因为这些 0 位不需要进行检索。

权限

None

副作用

无

另请参见

- “sa_split_list 系统过程”一节第 900 页
- “SET_BIT 函数 [Bit array]”一节第 288 页
- “SET_BITS 函数 [Aggregate]”一节第 289 页
- “GET_BIT 函数 [Bit array]”一节第 200 页

示例

以下示例显示如何使用 sa_get_bits 系统过程将一组整数编码为位字符串，然后对其进行解码以供在连接中使用：

```
CREATE VARIABLE @s_depts LONG VARBIT;

SELECT  SET_BITS( DepartmentID )
 INTO @s_depts
 FROM Departments
 WHERE DepartmentName like 'S%';

SELECT *
 FROM sa_get_bits( @s_depts ) B
  JOIN Departments D ON B.bitnum = D.DepartmentID;
```

sa_get_dtt 系统过程

报告磁盘传送时间 (DTT) 模型（开销模型的一部分）的当前值。

语法

sa_get_dtt(*dbspace_id* **)**

参数

- **dbspace_id**　使用此 UNSIGNED SMALLINT 参数指定数据库文件 ID。

注释

从 SYSDBSPACE 系统视图中可以获得 *dbspace_id*。

此用于内部诊断的过程检索来自 ISYSOPTSTAT 系统表的数据。

结果集

列名	数据类型	说明
BandSize	UNSIGNED INTEGER	进行随机访问的磁盘的大小（以页为单位）。
ReadTime	UNSIGNED INTEGER	读取一页的分摊开销（以毫秒为单位）。
WriteTime	UNSIGNED INTEGER	编写一页的分摊开销（以毫秒为单位）。

权限

无

副作用

无

另请参见

- “SYSDBSPACE 系统视图”一节第 943 页
- “SYSOPTSTAT 系统视图”一节第 960 页
- “sa_get_dtt_groupreads 系统过程”一节第 829 页

sa_get_dtt_groupreads 系统过程

估计并报告在数据库服务器上发出组读取的开销。

语法

sa_get_dtt_groupreads(*dbspace_id* **)**

参数

- **dbspace_id** 使用此 UNSIGNED SMALLINT 参数指定数据库文件 ID。

注释

从 SYSDBSPACE 系统视图中可以获得 *dbspace_id*。sa_get_dtt_groupreads 系统过程返回的估计是开销模型的一部分，用于在排序等操作期间选择适当大小的组读取。

此用于内部诊断的过程检索来自 ISYSOPTSTAT 系统表的数据。如果指定的 dbspace 不具有 SYSOPTSTAT 中记录的任何估计，则不会返回行。要为特定硬件设备定制估计，请执行以下语句：

```
ALTER DATABASE CALIBRATE GROUP READ;
```

结果集

列名	数据类型	说明
GroupSize	UNSIGNED INTEGER	进行随机访问的磁盘的大小（以页为单位）。
ReadTime	FLOAT	读取一页的分摊开销（以毫秒为单位）。

权限

None

副作用

None

另请参见

- “SYSDBSPACE 系统视图”一节第 943 页
- “SYSOPTSTAT 系统视图”一节第 960 页
- “ALTER DATABASE 语句”一节第 344 页
- “sa_get_dtt 系统过程”一节第 828 页

sa_get_histogram 系统过程

检索列的直方图。

语法

sa_get_histogram(
col_name,
tbl_name
[, *owner_name*]
)

参数

- **col_name** 使用此 CHAR(128) 参数指定从中检索直方图的列。
- **tbl_name** 使用此 CHAR(128) 参数指定从中找到 *col_name* 的表。
- **owner_name** 使用此可选的 CHAR(128) 参数指定 *tbl_name* 的所有者。

结果集

列名	数据类型	说明
StepNumber	SMALLINT	直方图域桶数。第一个域桶的频率 (StepNumber = 0) 指示 NULL 的选择性。
Low	CHAR(128)	域桶中的最低列值（包括最低值）。

列名	数据类型	说明
High	CHAR(128)	域桶中的最高列值（不包括最高值）。
Frequency	DOUBLE	域桶中各值的选择性。

注释

此用于内部诊断的过程检索来自数据库服务器指定列的列统计信息。请注意，当这些统计信息永久存储在系统表 ISYSCOLSTAT 中时，它们在服务器运行时保持在内存中，并被定期写入到 ISYSCOLSTAT。因此，由 sa_get_histogram 系统过程返回的统计信息可能与通过在任意给定时间点从 ISYSCOLSTAT 中选择而获取的统计信息不同。

通过使用 sa_flush_statistics 系统过程，可使用内存中所保存的最新统计信息来手工更新 ISYSCOLSTAT，然而，建议在生产环境中不要这样做，此做法应保留以用于诊断目的。请参见“sa_flush_statistics 系统过程”一节第 826 页。

单一域桶是通过结果集中的 Low 值等于相应的 High 值进行指示的。

建议您使用直方图实用程序查看直方图。请参见“直方图实用程序 (dbhist)”一节 《SQL Anywhere 服务器 - 数据库管理》。

要确定关于字符串列的谓词选择性，使用 ESTIMATE 或 ESTIMATE_SOURCE 函数。对于字符串列，sa_get_histogram 和直方图实用程序都从 ISYSCOLSTAT 系统表中检索不到任何内容。尝试检索字符串数据将会生成错误。请参见“ESTIMATE 函数 [Miscellaneous]”一节第 187 页和“ESTIMATE_SOURCE 函数 [杂类]”一节第 188 页。

表或实例化视图可能不会显示统计信息（包括直方图），例如，对于最近已删除统计信息的情况。在这种情况下，sa_get_histogram 系统过程的结果集为空。要为表或实例化视图创建统计信息，需执行 CREATE STATISTICS 语句。请参见“CREATE STATISTICS 语句”一节第 491 页。

权限

需要 DBA 权限

副作用

无

另请参见

- “优化程序估计值和列统计信息”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “直方图实用程序 (dbhist)”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SYSCOLSTAT 系统视图”一节第 941 页

示例

例如，以下语句检索 SalesOrderItems 表中 ProductID 列的直方图：

```
CALL sa_get_histogram( 'ProductID', 'SalesOrderItems' );
```

sa_get_request_profile 系统过程

分析请求日志以确定类似语句的执行时间。

语法

sa_get_request_profile(
[*filename*
[, *conn_id*
[, *first_file*
[, *num_files*]]]]
)

参数

- **filename** 使用此可选的 LONG VARCHAR 参数指定请求记录文件名。
- **conn_id** 使用此可选的 UNSIGNED INTEGER 参数指定连接的 ID 号。
- **first_file** 使用此可选的 INTEGER 参数指定要分析的第一个请求日志文件。
- **num_files** 使用此可选的 INTEGER 参数指定要分析的请求日志文件的数目。

注释

此过程调用 sa_get_request_times 以处理请求日志文件，然后将结果汇总到全局临时表 satmp_request_profile 中。该表包含日志中的语句，以及每个语句的执行次数和它们的总执行时间、平均和最大执行时间。该表可以按多种方式排序以确定性能优化目标。

如果不指定日志文件 (*filename*)，则缺省使用当前日志文件，即在带有 -zo 选项的命令中指定的日志文件，或是通过以下语句指定的日志文件

```
sa_server_option( 'RequestLogFile', filename )
```

如果指定了连接 ID，则会使用它从日志中过滤信息，以便仅检索该连接的请求。

权限

需要 DBA 权限

副作用

自动提交

示例

以下命令获得文件 *req.out.3*、*req.out.4* 和 *req.out.5* 中请求的请求次数。

```
CALL sa_get_request_profile('req.out',0,3,3);
```

另请参见

- “sa_get_request_times 系统过程”一节第 833 页
- “sa_statement_text 系统过程”一节第 902 页
- “sa_server_option 系统过程”一节第 886 页

sa_get_request_times 系统过程

分析请求日志以确定语句的执行时间。

语法

sa_get_request_times(*filename*
[, *conn_id*
[, *first_file*
[, *num_files*]]]
)

参数

- **filename** 使用此可选的 LONG VARCHAR 参数指定请求记录文件名。
- **conn_id** 使用此可选的 UNSIGNED INTEGER 参数指定连接的 ID 号。
- **first_file** 使用此可选的 INTEGER 参数指定要分析的第一个文件。
- **num_files** 使用此可选的 INTEGER 参数指定要分析的请求日志文件的数目。

注释

此过程读取指定的请求日志，并用日志中的语句和它们的执行时间填充全局临时表 satmp_request_time。

对于像插入和更新这样的语句，执行时间简单明了。而对于查询，时间是从准备语句开始一直计算到将其删除，包括描述语句、打开游标、读取行和关闭游标等一系列操作。对于大多数查询，这个值精确反映了查询所用的时间。而在有些情况中，当其它事件正在进行时游标仍然是打开的，例如操作员交互或客户端处理，这时时间显示为一个很大的值，但这并不是查询实际所花费的时间。

此过程识别请求日志中的主机变量，并使用其值填充全局临时表 satmp_request_hostvar。对于此临时表不存在的旧数据库，主机变量值将被忽略。

如果不指定日志文件，则缺省使用当前日志文件，即在带有 -zo 选项的命令中指定的日志文件，或是通过以下语句指定的日志文件

```
sa_server_option( 'RequestLogFile', filename )
```

如果指定了连接 ID，则会使用它从日志中过滤信息，以便仅检索该连接的请求。

权限

需要 DBA 权限

副作用

自动提交

示例

以下命令获得文件 *req.out.3*、*req.out.4* 和 *req.out.5* 中请求的执行次数。

```
CALL sa_get_request_times('req.out',0,3,3);
```

另请参见

- “sa_get_request_profile 系统过程”一节第 832 页
- “sa_statement_text 系统过程”一节第 902 页
- “sa_server_option 系统过程”一节第 886 页

sa_get_server_messages 系统过程 [不建议使用]

用于以结果集的形式从数据库服务器消息窗口返回常量。

不建议使用此系统过程。可改用 sa_server_messages。请参见“sa_server_messages 系统过程”一节第 884 页。

语法

sa_get_server_messages(*first_line* **)**

参数

- **first_line** 使用此 INTEGER 参数指定从其开始显示服务器消息的行号。

结果集

列名	数据类型	说明
line_num	INTEGER	服务器消息的行号。
message_text	VARCHAR(255)	服务器消息文本。
message_time	TIMESTAMP	消息的时间。

注释

此过程采用一个指定要显示的行的起始行号的 INTEGER 参数，并为该行及其后面的所有行返回一行。如果起始行是负数，则结果集从第一个可用行开始。结果集中包含行号、消息文本和消息时间。

权限

None

副作用

None

示例

以下示例使用 sa_get_server_messages 系统过程返回一个结果集，其中包含数据库服务器消息窗口中从第 16 行开始的内容。

```
CALL sa_get_server_messages( 16 );
```


line_num	message_text	...
16	正在 Windows 2000 Build 2195 上运行...	...
17	2132K 内存已用于高速缓存	...
...	...	...

sa_get_table_definition 系统过程

返回一个 LONG VARCHAR 字符串，其中包含创建指定表及其索引、外键、触发器和授予权限所必需的 SQL 语句。

语法

sa_get_table_definition(*@table_owner*, *@table_name* **)**

参数

- **@table_owner** 使用此 CHAR(128) 参数指定 *@table_name* 的所有者。
- **@table_name** 使用此 CHAR(128) 参数指定表的名称。

注释

要使用相同的定义创建新表，请使用由 sa_get_table_definition 系统过程返回的字符串以及 EXECUTE IMMEDIATE 语句和 LOCATE、SUBSTRING 及 REPLACE 函数。

权限

需要 DBA 权限

副作用

None

另请参见

- “sa_split_list 系统过程”一节第 900 页
- “EXECUTE IMMEDIATE 语句 [SP]”一节第 570 页
- “LOCATE 函数 [String]”一节第 230 页
- “SUBSTRING 函数 [String]”一节第 305 页
- “REPLACE 函数 [String]”一节第 278 页

示例

以下语句使用 sa_get_table_definition 系统过程显示包含 SQL 语句的字符串，创建 Departments 表时将用到这些 SQL 语句。

```
SELECT row_value
FROM sa_split_list( sa_get_table_definition( 'GROUPO', 'Departments'),
CHAR(10)) ;
```

sa_get_user_status 系统过程

用于确定用户的当前状态。

语法

sa_get_user_status

参数

None

结果集

列名	数据类型	说明
user_id	UNSIGNED INT	标识用户的唯一编号。
user_name	CHAR(128)	用户的名称。
connections	INT	该用户当前建立的连接数。
failed_logins	UNSIGNED INT	用户进行的登录失败重试次数。
last_login_time	TIMESTAMP	用户上次登录的时间。
locked	TINYINT	指示用户帐户是否锁定。
reason_locked	LONG VARCHAR(255)	帐户被锁定的原因。

注释

此过程返回一个显示用户当前状态的结果集。除了基本用户信息外，该过程还包括两列，分别指示用户是否被锁定以及锁定原因。用户可能由于以下几种原因而被锁定：由于策略、口令到期或失败重试次数过多而被锁定。

权限

需要具有 DBA 权限才能查看所有用户的信息。没有 DBA 权限的用户只能查看其自身的信息。

副作用

None

另请参见

- “管理登录策略概述”一节 《SQL Anywhere 服务器 - 数据库管理》
- “创建新登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》
- “创建用户并为其分配登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》
- “为现有用户分配登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》
- “修改登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》
- “删除登录策略”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下示例使用 sa_get_user_status 系统过程返回数据库用户的状态。

```
CALL sa_get_user_status;
```

sa_http_header_info 系统过程

返回 HTTP 标头的名称和值。

语法

sa_http_header_info([*header_parm*] **)**

参数

- **header_parm** 使用此可选的 VARCHAR(255) 参数指定 HTTP 标头名。

结果集

列名	数据类型	说明
Name	VARCHAR(255)	HTTP 标头的名称。
Value	LONG VARCHAR	HTTP 标头的值。

注释

sa_http_header_info 系统过程返回标头名和值。如果未使用可选参数指定标头名，则结果集包含所有标头的值。

如果在 Web 服务中处理 HTTP 请求时调用此过程，则此过程返回一个非空结果集。

权限

无

副作用

无

另请参见

- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_http_variable_info 系统过程”一节第 840 页
- “sa_set_http_header 系统过程”一节第 894 页
- “sa_set_http_option 系统过程”一节第 895 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

sa_http_php_page 系统过程

返回传递 PHP 代码的结果，PHP 解释器使用上下文信息（如标头、GET/POST 数据、协议版本、请求 URL、方法等）的当前 HTTP 请求来解释该代码。

语法

sa_http_php_page(*php_page* **)**

参数

- **php_page** 此 LONG VARCHAR 参数包含要解释的全部 PHP 代码，包括起始标记和结束标记（`<?php` 和 `?>`）。

注释

只有安装了 PHP 外部环境，才能使用此系统过程。请参见“PHP 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

此系统过程的所有者是 DBO。但为了提高安全性，需要将 sa_http_php_page 系统过程作为调用者执行。

权限

None

副作用

None

另请参见

- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “HTTP_BODY 函数 [HTTP]”一节第 214 页
- “sa_http_php_page_interpreted 系统过程”一节第 838 页
- “sa_http_header_info 系统过程”一节第 837 页
- “sa_set_http_header 系统过程”一节第 894 页
- “sa_set_http_option 系统过程”一节第 895 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

sa_http_php_page_interpreted 系统过程

返回传递 PHP 代码的结果，PHP 解释器使用上下文信息（如标头、GET/POST 数据、协议版本、请求 URL、方法等）的指定参数来解释该代码。

语法

sa_http_php_page_interpreted(
php_page,
method,

url,
version,
headers,
request_body
)

参数

- **php_page** 此 LONG VARCHAR 参数包含要解释的全部 PHP 代码，包括起始标记和结束标记（<?php 和 ?>）。
- **方法** 此 LONG VARCHAR 参数包含 HTTP 请求方法（例如 GET、POST、PUT 或其它标准请求方法之一）。可使用当前 HTTP 请求中的 @HttpMethod 值来确定 *method* 值。
- **url** 此 LONG VARCHAR 参数包含完整的 HTTP 请求 URL，如果存在查询字符串，还将包含查询字符串。可使用当前 HTTP 请求中的 @HttpURI 值来确定 *url* 值。
- **version** 此 LONG VARCHAR 参数包含 HTTP 请求协议版本（例如，HTTP/1.1）。可使用当前 HTTP 请求中的 @HttpVersion 值来确定 *version* 值。
- **标头** 此 LONG BINARY 参数包含 HTTP 请求标头，采用标准 HTTP 标头格式：Field-Name: Value\r\n。可以使用以下 SELECT 语句从当前 HTTP 请求中检索标头值：

```
SELECT LIST( name || ': ' || value, CHAR(13) || CHAR(10) )
   FROM sa_http_header_info();
```

- **request_body** 此 LONG BINARY 参数包含二进制形式的 HTTP 请求主体。可以使用 HTTP_BODY 函数从当前 HTTP 请求中检索 *request_body* 的值。请参见“HTTP_BODY 函数 [HTTP]”一节第 214 页。

注释

只有安装了 PHP 外部环境，才能使用此系统过程。请参见“PHP 外部环境”一节 《SQL Anywhere 服务器 - 编程》。

要在 Web 服务请求以外使用此系统过程，必须提供请求信息。在 PHP 代码中设置的任何标头都将丢失。

此系统过程的所有者是 DBO。但为了提高安全性，需要将 sa_http_php_page_interpreted 系统过程作为调用者执行。

权限

None

副作用

None

另请参见

- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “HTTP_BODY 函数 [HTTP]”一节第 214 页
- “sa_http_php_page 系统过程”一节第 838 页
- “sa_http_header_info 系统过程”一节第 837 页
- “sa_set_http_header 系统过程”一节第 894 页
- “sa_set_http_option 系统过程”一节第 895 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

sa_http_variable_info 系统过程

返回 HTTP 变量的名称和值。

语法

sa_http_variable_info([*variable_parm*] **)**

参数

- **variable_parm** 使用此可选的 VARCHAR(255) 参数指定 HTTP 变量名。

结果集

列名	数据类型	说明
Name	VARCHAR(255)	HTTP 变量的名称。
Value	LONG VARCHAR	HTTP 变量的值。

注释

sa_http_variable_info 系统过程返回变量名和值。如果未使用可选参数指定变量名，则结果集包含所有变量的值。

如果在 Web 服务中处理 HTTP 请求时调用此过程，则此过程返回一个非空结果集。

权限

无

副作用

无

另请参见

- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_http_header_info 系统过程”一节第 837 页
- “sa_set_http_header 系统过程”一节第 894 页
- “sa_set_http_option 系统过程”一节第 895 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

sa_index_density 系统过程

报告索引中碎片和分布偏差的数量信息。

语法

sa_index_density(
 [*tbl_name*
 [**,** *owner_name*]]
)

参数

- **tbl_name** 使用此可选的 CHAR(128) 参数指定表名。
- **owner_name** 使用此可选的 CHAR(128) 参数指定所有者名。

结果集

列名	数据类型	说明
TableName	CHAR(128)	表的名称。
TableId	UNSIGNED INTEGER	表 ID。
IndexName	CHAR(128)	索引的名称。
IndexId	UNSIGNED INTEGER	索引 ID。此列包含以下值之一： ● **0** 对于主键 ● **SYSFKEY.*foreign_key_id*** 对于外键 ● **SYSIDX.*index_id*** 对于所有其它索引

列名	数据类型	说明
IndexType	CHAR(4)	索引类型。此列包含以下值之一： ● **PKEY** 对于主键 ● **FKEY** 对于外键 ● **UI** 对于唯一索引 ● **UC** 对于唯一约束 ● **NUI** 对于非唯一索引
LeafPages	UNSIGNED INTEGER	叶页的数目。
Density	NUMERIC(8,6)	介于 0 和 1 之间的分数，提供每个索引页的平均占用程度的指示。
Skew	NUMERIC(8.6)	一个数字，用于指示索引中的不均衡级别。值 1 表示完全均衡的索引。数值越大，分布偏差的程度越大。

注释

sa_index_density 系统过程用于获取索引中碎片和分布偏差程度的信息。对于有较多叶页的索引，理想的情况是密度值较高、分布偏差值较低。

索引密度反映索引页的平均丰满度，以百分比表示。密度为 0.7 表示平均有 70% 的索引页已填满索引数据。索引分布偏差反映了与平均密度的典型差值。进行选择性估计时，分布偏差量对于优化程序十分重要。

当叶页数较低时，无需考虑密度和分布偏差值。只有叶页数较高时，密度和分布偏差值才变得重要。当叶页数较高时，较低的密度值表示存在碎片，而较高的分布偏差值则表示索引分布不均衡。二者都有可能是导致性能低下的因素。通过执行 REORGANIZE TABLE 语句即可解决这两个问题。请参见“REORGANIZE TABLE 语句”一节第 673 页。

如果调用此过程时未指定表，则返回数据库中所有表上的所有索引信息。

还可以使用 [**应用程序分析向导**] 确定索引密度和分布偏差是否处于可以接受的级别。请参见“应用程序分析向导”一节 《SQL Anywhere 服务器 - SQL 的用法》。

权限

需要 DBA 权限

副作用

无

另请参见

● “减少索引碎片和分布偏差”一节 《SQL Anywhere 服务器 - SQL 的用法》

示例

以下示例使用 sa_index_density 系统过程返回一个结果集，汇总了数据库中所有索引的碎片和分布偏差的数量。

```
CALL sa_index_density( );
```

sa_index_levels 系统过程

通过报告索引的级数帮助优化性能。

语法

sa_index_levels(
[*tbl_name*
[, *owner_name*]]
)

参数

- **tbl_name** 使用此可选的 CHAR(128) 参数指定表名。
- **owner_name** 使用此可选的 CHAR(128) 参数指定所有者名。

结果集

列名	数据类型	说明
TableName	CHAR(128)	表的名称。
TableId	UNSIGNED INTEGER	表 ID。
IndexName	CHAR(128)	索引的名称。
IndexId	UNSIGNED INTEGER	索引 ID。此列包含以下值之一： ● **0** 对于主键 ● **SYSFKEY.*foreign_key_id*** 对于外键 ● **SYSIDX.*index_id*** 对于所有其它索引
IndexType	CHAR(4)	索引类型。此列包含以下值之一： ● **PKEY** 对于主键 ● **FKEY** 对于外键 ● **UI** 对于唯一索引 ● **UC** 对于唯一约束 ● **NUI** 对于非唯一索引

列名	数据类型	说明
Levels	INTEGER	索引中的级数。

注释

索引树的级数决定了使用索引访问一行所需的 I/O 操作数。级别少的索引比级别多的索引效率更高。

此过程返回一个包含表名、表 ID、索引名、索引 ID、索引类型和索引级数的结果集。

如果不提供参数，则返回数据库中的所有索引的级数。如果只提供了 *tbl_name*，则提供那个表的所有索引的级数。如果 *tbl_name* 为 NULL 并且提供了 *owner_name*，则只返回该用户所拥有的表的索引的级数。

权限

需要 DBA 权限

副作用

无

另请参见

- “CREATE INDEX 语句”一节第 449 页
- “使用索引”一节 《SQL Anywhere 服务器 - SQL 的用法》

示例

以下示例使用 sa_index_levels 系统过程返回 Products 索引的级数。

```
CALL sa_index_levels( );
```

TableName	TableId	IndexName	...	Levels
Products	436	Products	...	1
...	...	...	...	...

sa_java_loaded_classes 系统过程

列出当前由数据库服务器装载到 Java 虚拟机中的类。

语法

sa_java_loaded_classes( )

结果集

列名	数据类型	说明
class_name	VARCHAR(512)	当前由数据库服务器装载到 Java 虚拟机中的类的名称。

注释

返回的结果集包含当前由数据库服务器装载到 Java 虚拟机中的所有 Java 类的名称。

此过程在诊断遗失的类时很有用。它还可以用来确定具体某个 jar 中的哪些类被给定的应用程序使用。

权限

需要 DBA 权限

副作用

无

另请参见

- “将 Java 类安装到数据库中”一节 《SQL Anywhere 服务器 - 编程》

sa_load_cost_model 系统过程

使用指定文件中存储的开销模型替换当前开销模型。

语法

sa_load_cost_model (*file_name* **)**

参数

- ***file_name*** 使用此 CHAR(1024) 参数指定要装载的开销模型文件的名称。

注释

优化程序使用开销模型来确定查询的最佳访问计划。数据库服务器维护每个数据库的开销模型。可随时使用 ALTER DATABASE 语句中的 CALIBRATE SERVER 子句来重新校准数据库的开销模型。例如，如果将数据库移动到非标准硬件上，您可能决定要重新校准开销模型。

sa_load_cost_model 系统过程允许您装载已保存到文件 (*file_name*) 的开销模型。装载开销模型会替换数据库的当前开销模型。

注意
sa_unload_cost_model 系统过程不会在 sa_load_cost_model 装载的文件中包含 CALIBRATE PARALLEL READ 信息。

在存在大量相同的硬件安装的情况下，使用 sa_load_cost_model 系统过程可以避免重复且耗时的重新校准活动。

装载新开销模型时，要求独占使用数据库。

当装载开销模型时，考虑是否为位于相似硬件上的数据库生成开销模型。如果装载来自存储在明显不同硬件上的数据库的开销模型，可能会由于访问计划效率不高而导致性能很差。

使用 sa_unload_cost_model 系统过程将开销模型保存到文件。请参见“sa_unload_cost_model 系统过程”一节第 910 页。

权限

必须具有 DBA 权限。

副作用

装载新开销模型后，数据库服务器会执行 COMMIT。

另请参见

- “ALTER DATABASE 语句”一节第 344 页
- “sa_unload_cost_model 系统过程”一节第 910 页
- “查询优化与执行” 《SQL Anywhere 服务器 - SQL 的用法》

示例

以下示例装载来自名为 costmodel8 的文件中的开销模型：

```
CALL sa_load_cost_model( 'costmodel8' );
```

sa_locks 系统过程

显示数据库中的所有锁。

语法

sa_locks(
 [*connection*
 [, *creator*
 [, *table_name*
 [, *max_locks*]]]]
)

参数

- **connection** 使用此 INTEGER 参数指定连接 ID。该过程只返回有关指定连接的锁信息。缺省值为 0（或 NULL），此情况下返回有关所有连接的信息。
- **creator** 使用此 CHAR(128) 参数指定用户 ID。该过程只返回有关指定用户所拥有的表的信息。creator 参数的缺省值为 NULL。当此参数设置为 NULL 时，sa_locks 返回以下信息：
  - 如果未指定 table_name 参数，则返回数据库中所有表的锁定信息
  - 如果指定了 table_name 参数，则返回当前用户创建的具有指定名称的表的锁定信息
- **table_name** 使用此 CHAR(128) 参数指定表名。此过程仅返回指定表的信息。缺省值为 NULL，此情况下返回所有表的信息。

- **max_locks** 使用此 INTEGER 参数指定为其返回信息的锁的最大数目。缺省值为 1000。值 -1 表示返回所有锁信息。

结果集

列名	数据类型	说明
conn_name	VARCHAR(128)	当前连接的名称。
conn_id	INTEGER	连接的 ID 号
user_id	CHAR(128)	通过连接 ID 连接的用户。
table_type	CHAR(6)	表的类型。该类型为 BASE（对于表）、GLBTMP（对于全局临时表）或 MVIEW（对于实例化视图）。
creator	VARCHAR(128)	表的所有者。
table_name	VARCHAR(128)	控制锁的表。
index_id	INTEGER	索引 ID 或 NULL。
lock_class	CHAR(8)	锁类。Schema、Row、Table 或 Position 之一。请参见“可以锁定的对象”一节 《SQL Anywhere 服务器 - SQL 的用法》。
lock_duration	CHAR(11)	锁的持续时间。Transaction、Position 或 Connection 之一。
lock_type	CHAR(9)	锁类型（这取决于锁类）。
row_identifier	UNSIGNED BIGINT	行的标识符。此为 8 字节行标识符或为 NULL。

注释

sa_locks 过程返回的结果集包含数据库中所有锁的信息。

lock_type 列中的值取决于 lock_class 列中的锁分类。可以返回以下值：

锁类	锁类型	注释
Schema	Shared（共享模式锁） Exclusive（独占模式锁）	对于模式锁，row_identifier 和索引 ID 值均为 NULL。请参见“模式锁”一节 《SQL Anywhere 服务器 - SQL 的用法》。

锁类	锁类型	注释
Row	Read（读锁定） Intent（意图锁） Write（写锁定） Surrogate（代理锁定）	行的读锁定可以是短期锁定（在隔离级别 1 的扫描），也可以是在更高隔离级别的长期锁定。lock_duration 列指明读锁定是由于游标稳定性而短期锁定 (Position)，还是长期锁定一直保持到 COMMIT/ROLLBACK (Transaction)。行锁定始终保持在其 8 字节行标识符作为 row_identifier 列中的 64 位整数值而被报告的特定行上。代理锁定是行锁定的一种特殊情况。代理锁定被保持在代理条目上，这些代理条目是在参照完整性检查延迟时创建的。请参见“插入过程中的锁定”一节 《SQL Anywhere 服务器 - SQL 的用法》。对于表中所创建的每个代理条目，没有唯一的代理锁定。而是一个代理锁定对应于由给定连接为给定表所创建的一组代理条目。对于与代理锁定关联的表和连接，row_identifier 值是唯一的。请参见“行锁”一节 《SQL Anywhere 服务器 - SQL 的用法》。
Table	Shared（共享表锁） Intent（意图更新表锁） Exclusive（独占表锁）	请参见“表锁”一节 《SQL Anywhere 服务器 - SQL 的用法》。
Position	Phantom（幻像锁） Insert（插入锁）	多数情况下，位置锁也被保持在特定行上，且此行的 64 位行标识符出现在结果集的 row_identifier 列中。然而，位置锁还可以被保持在整个扫描上（索引扫描或顺序扫描），在此情况下，row_identifier 列为 NULL。请参见“位置锁”一节 《SQL Anywhere 服务器 - SQL 的用法》。

位置锁可以与顺序表扫描或索引扫描相关联。index_id 列指明位置锁是否与顺序扫描相关联。如果由于顺序扫描而保持位置锁，则 index_id 列为 NULL。如果由于特定的索引扫描而保持位置锁，则此索引的索引标识符在 index_id 列中列出。该索引标识符对应于 ISYSIDX 系统表的主键，可使用 SYSIDX 视图查看此系统表。如果由于全部索引的扫描而保持位置锁，则索引 ID 值为 -1。

权限

需要 DBA 权限

副作用

无

另请参见

- “锁定的工作方式”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “SYSIDX 系统视图”一节第 952 页

示例

有关此系统过程的示例以及扩展可返回的信息量的提示，请参见“获取有关锁的信息”一节《SQL Anywhere 服务器 - SQL 的用法》。

sa_make_object 系统过程

请确保在执行 ALTER 语句前存在对象的框架实例。

语法

```
sa_make_object(
  objtype,
  objname
  [, owner
  [, tabname ] ]
)

objtype:
'procedure'
| 'function'
| 'view'
| 'trigger'
| 'service'
| 'event'
```

参数

- **objtype** 使用此 CHAR(30) 参数指定所创建对象的类型。如果 objtype 为 **'trigger'**，此参数指定要在其上创建触发器的表的所有者。
- **objname** 使用此 CHAR(128) 参数指定要创建的对象的名称。
- **owner** 使用此可选的 CHAR(128) 参数指定要创建的对象的所有者。缺省值为 CURRENT USER。
- **tabname** 仅在 objtype 为 'trigger' 时才需要此 CHAR(128) 参数，在此情况下，该参数用于指定要在其上创建触发器的表的名称。

注释

在那些需要重复运行以创建或修改数据库模式的脚本或命令文件中，此过程很有用。这类脚本中的一个共同问题是：当这些脚本在第一次运行时，必须执行一个 CREATE 语句，以后运行时必须执行 ALTER 语句。使用此过程，就不必通过查询系统视图来查看对象是否存在。

若要使用此过程，请在此过程后使用一个包含整个对象定义的 ALTER 语句。

权限

创建或修改数据库对象需要有资源权限

副作用

自动提交

另请参见

- “ALTER EVENT 语句”一节第 351 页
- “ALTER FUNCTION 语句”一节第 354 页
- “ALTER PROCEDURE 语句”一节第 361 页
- “ALTER TRIGGER 语句”一节第 384 页
- “ALTER VIEW 语句”一节第 387 页
- “ALTER SERVICE 语句”一节第 366 页

示例

以下语句可确保创建框架过程定义，可以定义过程，并可授予对其的权限。可以对数据库重复运行包含这些指令的命令文件而不会出现任何错误。

```
CALL sa_make_object( 'procedure','myproc' );
ALTER PROCEDURE myproc( in p1 INT, in p2 CHAR(30) )
BEGIN
    // ...
END;
GRANT EXECUTE ON myproc TO public;
```

以下示例使用 sa_make_object 系统过程添加一个框架 Web 服务。

```
CALL sa_make_object( 'service','my_web_service' );
```

sa_materialized_view_can_be_immediate 系统过程

返回是否可将指定的实例化视图定义为快速视图。

语法

sa_materialized_view_can_be_immediate(
view_name
, *owner_name*
)

参数

- **view_name** 此 CHAR(128) 参数用于指定实例化视图的名称。如果 *view_name* 为 NULL，则返回 [未找到表...] 错误。
- **owner_name** 此 CHAR(128) 参数用于指定实例化视图的所有者。如果 *owner_name* 为 NULL，则返回 [未找到表...] 错误。

注释

指定的手动视图是否可以变为快速视图有某些限制。此系统过程用于确定是否允许此更改。有关创建快速视图的其它限制的列表，请参见“快速视图的附加限制”一节 《SQL Anywhere 服务器 - SQL 的用法》。

sa_materialized_view_can_be_immediate 系统过程返回指定实例化视图的以下信息。

列名	数据类型	说明
SQLStateVal	CHAR(6)	返回的 SQLSTATE。
ErrorMessage	LONG VARCHAR	与 SQLSTATE 对应的错误消息。

结果集中的每一行与返回的视图的一个 SQLSTATE 相对应。因此，如果实例化视图定义违反多条限制，结果会包含视图的多个行。

可将此系统过程的输出与 sa_materialized_view_info 系统过程的输出相结合，以获取有关视图状态和能否将其转换为快速视图的信息。请参见“sa_materialized_view_info 系统过程”一节第 852 页中的 "示例" 部分。

权限

DBA 权限，或对 DBO 拥有的过程的执行权限。

另请参见

- “将手动视图更改为快速视图”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “快速视图的附加限制”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_materialized_view_info 系统过程”一节第 852 页

副作用

指定实例化视图的所有元数据和所有依赖性都将装载到服务器高速缓存中。

示例

执行以下语句，创建手动视图 view10，然后刷新该视图。

```
CREATE MATERIALIZED VIEW view10
   AS (SELECT C.ID, C.Surname, sum(P.UnitPrice) as revenue, C.CompanyName,
SO.OrderDate
         FROM Customers C, SalesOrders SO, SalesOrderItems SOI, Products P
         WHERE C.ID = SO.CustomerID
         AND SO.ID = SOI.ID
         AND P.ID = SOI.ProductID
         GROUP BY C.ID, C.Surname, C.CompanyName, SO.OrderDate);
REFRESH MATERIALIZED VIEW view10;
```

可使用以下查询找出无法将 view10 转变为快速视图的原因：

```
SELECT SQLStateVal AS "SQLstate", ErrorMessage AS Description
   FROM sa_materialized_view_can_be_immediate( 'view10', 'DBA' )
   ORDER BY SQLSTATE;
```

SQLstate	说明
42WC3	实例化视图 view10 已经初始化，所以不能转变为快速视图。
42WCA	实例化视图 view10 在不可为空的列上没有唯一索引，所以不能转变为快速视图。
42WC6	COUNT(*) 需要作为 SELECT 列表的一部分，所以实例化视图不能转变为快速视图。

SQLstate	说明
42WC7	实例化视图在非集合不可为空的列上没有唯一索引，所以不能转变为快速视图。

sa_materialized_view_info 系统过程

返回指定的实例化视图的信息。

语法

sa_materialized_view_info(
[*view_name*
[, *owner_name*]]
)

参数

- **view_name** 使用此可选的 CHAR(128) 参数指定要为其返回信息的实现化视图 (Materialized View) 的名称。
- **owner_name** 使用此可选的 CHAR(128) 参数指定实现化视图 (Materialized View) 的所有者。

注释

如果 *view_name* 和 *owner_name* 都未指定，则返回有关数据库中所有实例化视图的信息。

如果 *owner_name* 未指定，则返回所有名为 *view_name* 的实例化视图信息。

sa_materialized_view_info 系统过程返回实例化视图的以下信息：

列名	数据类型	说明
OwnerName	CHAR(128)	视图的所有者。
viewname	CHAR(128)	视图的名称。
Status	CHAR(1)	视图的状态信息。可能的值为： ● **D** 禁用 ● **E** enabled

列名	数据类型	说明
DataStatus	CHAR(1)	视图中数据的状态信息。可能的值为： ● **E** 上次尝试刷新过程中出现错误。视图已启用，但未初始化。 ● **F** 自上次刷新后，基础表未发生更改，因此认为此视图为最新视图。视图已启用且已初始化。 ● **N** 视图尚未初始化。下列其中一项为真时会出现这种情况： ○ 视图自创建以来从未刷新 ○ 数据已从视图中截断 ○ 视图被禁用 ● **S** 自上次刷新后基础表已更改，因此认为此视图失效。视图已启用且已初始化。
ViewLastRefreshed	TIMESTAMP	上次刷新视图时的时间。如果 ViewLastRefreshed 值为 NULL，则视图尚未初始化。
DataLastModified	TIMESTAMP	针对失效视图，上次修改基础数据的时间。 对于未初始化的视图，或认为未失效的视图，其值为 NULL。

列名	数据类型	说明
AvailForOptimization	CHAR(1)	有关视图供优化程序使用的可用性信息。可能的值为： ● **D** 禁止优化程序使用视图。视图所有者不允许优化程序使用该视图。 ● **I** 由于某些内部原因（如视图定义不符合所需的条件），优化程序可能无法使用视图。但所有者并未显式禁止优化程序使用该视图。 ● **N** 因为尚未刷新或刷新失败，所以视图不包含任何数据。视图所有者允许优化程序使用该视图，但视图未初始化。 ● **O** 当前连接中存在不兼容的选项值。该视图可供优化程序使用，并且其定义满足所有要求的条件，但当前选项设置与创建视图所使用的选项设置不兼容。 ● **Y** 优化程序可以使用视图。视图所有者允许优化程序使用该视图，并且视图定义满足优化程序使用时需要的所有条件。 有关优化程序是否选择实例化视图以及如何选择的详细信息，请参见“使用实例化视图提高性能”一节 《SQL Anywhere 服务器 - SQL 的用法》。
RefreshType	CHAR(1)	视图的刷新类型。可能的值为： ● **I** 视图为快速视图。基础表中的数据更改影响到实例化视图中的数据时，快速视图立即刷新。 ● **M** 视图为手动视图。手动视图需要手动刷新，例如可以使用 REFRESH MATERIALIZED VIEW 语句或 sa_refresh_materialized_views 系统过程刷新视图。 有关手工视图和快速视图的详细信息，请参见“手动和快速实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。

对于确定由于视图定义问题而优化程序永不考虑使用的实例化视图列表，此过程非常有用。对于这些实现化视图 (Materialized View)，AvailForOptimization 值为 I。要进一步了解有关实例化视图定义的限制，请参见“实例化视图的限制”一节 《SQL Anywhere 服务器 - SQL 的用法》。

下表显示如何确定 AvailForOptimization 属性。从左列开始逐行读取，查看为了能够产生 AvailForOptimization 列中找到的值而必须满足的条件。

用户是否允许视图在优化中使用？	视图定义是否满足使用所需的所有条件？	连接选项是否与使用视图所需的选项相匹配？	视图是否初始化？	AvailForOptimization 值
是	是	是	是	Y
否	N/A	N/A	N/A	D
是	否	N/A	是	I
是	N/A	N/A	否	N
是	是	否	是	O

经过初始化的实例化视图可以为空。满足实例化视图定义的基础表中没有数据时会出现这种情况。空视图不视为与未初始化的实例化视图相同，虽然该视图中也没有数据。可以使用 ViewLastRefreshed 属性的值区分该视图是尚未初始化 (NULL)，还是由于基础表中没有数据而为空（非 NULL）。

权限

DBA 权限，或对 DBO 拥有的过程的执行权限。

副作用

指定实例化视图的所有元数据和所有依赖性都将装载到数据库服务器高速缓存中。

另请参见

- “使用实例化视图提高性能”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “将手动视图更改为快速视图”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “快速视图的附加限制”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_materialized_view_can_be_immediate 系统过程”一节第 850 页

示例

下列语句返回有关数据库中所有实例化视图的信息：

```
SELECT *
   FROM sa_materialized_view_info();
```

可将 sa_materialized_view_info 系统过程的结果与 sa_materialized_view_can_be_immediate 系统过程的结果相结合，以返回状态信息以及视图是否满足成为快速视图的条件。执行以下语句可创建为此示例检查的实例化视图：

```
CREATE MATERIALIZED VIEW view0 AS (
   SELECT ID, Name, Description, Size
   FROM Products
   WHERE Quantity > 0 );
CREATE UNIQUE INDEX u_view0
   ON view0( ID );
ALTER MATERIALIZED VIEW view0
   IMMEDIATE REFRESH;
CREATE MATERIALIZED VIEW view00 AS (
```

```
   SELECT ID, Name, Description, Size
   FROM Products
   WHERE Quantity <= 0 );
CREATE UNIQUE INDEX u_view00
   ON view00( ID );
CREATE MATERIALIZED VIEW view1 AS (
   SELECT ID, Name, Description, Size
   FROM Products
   WHERE Quantity = 0 );
ALTER MATERIALIZED VIEW view1
   DISABLE;
CREATE MATERIALIZED VIEW view100
   AS (SELECT C.ID, C.Surname, sum(P.UnitPrice) as revenue, C.CompanyName,
SO.OrderDate
         FROM Customers C, SalesOrders SO, SalesOrderItems SOI, Products P
         WHERE C.ID = SO.CustomerID
         AND SO.ID = SOI.ID
         AND P.ID = SOI.ProductID
         GROUP BY C.ID, C.Surname, C.CompanyName, SO.OrderDate);
REFRESH MATERIALIZED VIEW view100;
```

执行以下语句可返回刚才创建的视图的状态信息以及是否符合条件的信息：

```
SELECT ViewName, Status, ViewLastRefreshed, AvailForOptimization,
RefreshType, CanBeImmediate
FROM sa_materialized_view_info() AS V,
   LATERAL( SELECT LIST( ErrorMessage )
            FROM sa_materialized_view_can_be_immediate( V.ViewName,
V.OwnerName ) ) AS I( CanBeImmediate );
```

View-Name	Status	ViewLast-Refreshed	AvailFor-Optimization	RefreshType	CanBeImmediate
view0	E	(NULL)	N	I	
view00	E	(NULL)	N	M	
view1	D	(NULL)	N	M	不能使用视图 'view1'，因为它已经被禁用
view100	E	2008-02-12 16:47:00.000	Y	M	实例化视图 view10 已经初始化，所以不能转变为快速视图。实例化视图 view10 在不可为空的列上没有唯一索引，所以不能转变为快速视图。COUNT(*) 需要作为选择列表的一部分，所以实例化视图不能转变为快速视图。实例化视图在非集合不可为空的列上没有唯一索引，所以不能转变为快速视图。

从结果中可以看出：

- view0 从未经过刷新且为快速视图。

- view00 从未经过刷新且为手动视图。
- view1 被禁用
- view100 为手动视图，上次刷新时间为 2008-02-12 16:47:00.000。
- 因为 CanBeImmediate 列中没有错误消息，所以 view00 可以转变为快速视图。
- 由于 CanBeImmediate 列中列出的原因，view1 和 view100 不能变为快速视图。

sa_migrate 系统过程

将一组远程表迁移到 SQL Anywhere 数据库中。

语法

sa_migrate(
base_table_owner,
server_name
[, *table_name*]
[, *owner_name*]
[, *database_name*]
[, *migrate_data*]
[, *drop_proxy_tables*]
[, *migrate_fkeys*]
)

参数

- **base_table_owner** 使用此 VARCHAR(128) 参数指定在目标 SQL Anywhere 数据库中拥有迁移表的用户。使用 GRANT CONNECT 语句创建此用户。此参数需要值。请参见“GRANT 语句”一节第 595 页。
- **server_name** 使用此 VARCHAR(128) 参数指定用于连接到远程数据库的远程服务器的名称。使用 CREATE SERVER 语句创建此服务器。此参数需要值。请参见“CREATE SERVER 语句”一节第 481 页。
- **table_name** 如果要迁移单个表，请使用此 VARCHAR(128) 参数指定表名。否则，应该为此参数指定 NULL（缺省值）。不要将 table_name 和 owner_name 参数都指定为 NULL。
- **owner_name** 如果只迁移属于一个所有者的表，请使用此 VARCHAR(128) 参数指定该所有者的名称。否则，应该为此参数指定 NULL（缺省值）。不要将 table_name 和 owner_name 参数都指定为 NULL。
- **database_name** 使用此 VARCHAR(128) 参数指定远程数据库的名称。如果想仅迁移远程服务器上的一个数据库中的表，必须指定数据库名。否则，为此参数输入 NULL（缺省值）。
- **migrate_data** 使用此可选的 BIT 参数指定是否迁移远程表中的数据。此参数可为 0（不迁移数据）或 1（迁移数据）。缺省情况下迁移数据。 (1)
- **drop_proxy_tables** 使用此可选的 BIT 参数指定在迁移完成后是否删除为迁移进程所创建的代理表。此参数可为 0（不删除代理表）或 1（删除代理表）。缺省情况下删除代理表 (1)。

- **migrate_fkeys** 使用此可选的 BIT 参数指定是否迁移外键映射。此参数可为 0（不迁移外键映射）或 1（迁移外键映射）。缺省情况下迁移外键映射 (1)。

注释

可以使用此过程将表从远程 Oracle、DB2、SQL Server、Adaptive Server Enterprise 或 SQL Anywhere 数据库迁移到 SQL Anywhere 中。使用此过程，仅需要一步，即可从指定的服务器迁移一组远程表，包括它们的外键映射。sa_migrate 系统过程调用以下系统过程：

- sa_migrate_create_remote_table_list
- sa_migrate_create_tables
- sa_migrate_data
- sa_migrate_create_remote_fks_list
- sa_migrate_create_fks
- sa_migrate_drop_proxy_tables

如果需要更大的灵活性，可能需要使用这些系统过程而不是 sa_migrate。例如，如果迁移外键关系由不同用户拥有的表，则使用 sa_migrate 时无法保持外键关系。

在可以迁移任何表之前，必须先使用 CREATE SERVER 语句创建远程服务器以连接到远程数据库。可能还需要使用 CREATE EXTERNLOGIN 语句创建到远程数据库的外部登录。请参见“CREATE SERVER 语句”一节第 481 页和“CREATE EXTERNLOGIN 语句”一节第 436 页。

可通过仅指定 *base_table_owner* 和 *server_name* 参数，将所有表从远程数据库迁移到 SQL Anywhere 数据库。然而，如果仅指定这两个参数，所有迁移的表都将属于目标 SQL Anywhere 数据库中的一个所有者。如果表在远程数据库上有不同的所有者，并且您希望它们在 SQL Anywhere 数据库上有不同的所有者，则必须分别为每个所有者迁移表，每次调用 sa_migrate 过程时都指定 *base_table_owner* 和 *owner_name* 参数。

小心
不要将 *table_name* 和 *owner_name* 参数都指定为 NULL。为 *table_name* 和 *owner_name* 参数提供 NULL 值将迁移数据库中的所有表，包括系统表。此外，远程数据库中那些名称相同但所有者不同的表在目标数据库中均属同一所有者。建议您一次只迁移与一个所有者关联的表。

权限

无

副作用

无

另请参见

- “将数据库迁移到 SQL Anywhere”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_migrate_create_remote_table_list 系统过程”一节第 861 页
- “sa_migrate_create_tables 系统过程”一节第 863 页
- “sa_migrate_data 系统过程”一节第 864 页
- “sa_migrate_create_remote_fks_list 系统过程”一节第 860 页
- “sa_migrate_create_fks 系统过程”一节第 859 页
- “sa_migrate_drop_proxy_tables 系统过程”一节第 865 页

示例

以下语句迁移远程数据库中所有属于用户 p_chin 的表（包括外键映射），迁移远程表中的数据，并且在迁移完成后删除代理表。此例中，所有迁移的表都属于目标 SQL Anywhere 数据库中的 local_user。

```
CALL sa_migrate( 'local_user', 'server_a', NULL, 'p_chin', NULL, 1, 1, 1 );
```

以下语句仅迁移远程数据库中属于用户 remote_a 的表。在目标 SQL Anywhere 数据库中，这些表属于用户 local_a。完成时不会删除在迁移阶段创建的代理表。

```
CALL sa_migrate( 'local_a', 'server_a', NULL, 'remote_a', NULL, 1, 0, 1 );
```

sa_migrate_create_fks 系统过程

为表 dbo.migrate_remote_fks_list 中列出的每个表创建外键。

语法

sa_migrate_create_fks(*i_table_owner* **)**

参数

- **i_table_owner** 使用此 VARCHAR(128) 参数指定在目标 SQL Anywhere 数据库中拥有所迁移外键的用户。如果想迁移属于不同用户的表，则必须为拥有您想迁移的表的每个用户执行此过程。*i_table_owner* 是使用 GRANT CONNECT 语句创建的。此参数需要值。请参见“GRANT 语句”一节第 595 页。

注释

此过程为 dbo.migrate_remote_fks_list 表中列出的每个表创建外键。*i_table_owner* 参数所指定的用户在目标数据库中拥有外键。

如果目标 SQL Anywhere 数据库中的表不是全都属于同一个所有者，则必须为拥有您想迁移其外键的表的每个用户执行此过程。

注意
此系统过程与其它几个迁移系统过程一起使用，它们必须按以下所列的顺序执行：

1. sa_migrate_create_remote_table_list
2. sa_migrate_create_tables
3. sa_migrate_data
4. sa_migrate_create_remote_fks_list
5. sa_migrate_create_fks
6. sa_migrate_drop_proxy_tables

另一种选择是，使用 sa_migrate 系统过程在一步中迁移所有的表。

权限

无

副作用

无

另请参见

- “将数据库迁移到 SQL Anywhere”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_migrate 系统过程”一节第 857 页
- “sa_migrate_create_remote_table_list 系统过程”一节第 861 页
- “sa_migrate_create_tables 系统过程”一节第 863 页
- “sa_migrate_data 系统过程”一节第 864 页
- “sa_migrate_create_remote_fks_list 系统过程”一节第 860 页
- “sa_migrate_drop_proxy_tables 系统过程”一节第 865 页

示例

以下语句基于 dbo.migrate_remote_fks_list 表创建外键。外键属于本地 SQL Anywhere 数据库中的用户 local_a。

```
CALL sa_migrate_create_fks( 'local_a' );
```

sa_migrate_create_remote_fks_list 系统过程

填充 dbo.migrate_remote_fks_list 表。

语法

sa_migrate_create_remote_fks_list(*server_name* **)**

参数

- **server_name** 使用此 VARCHAR(128) 参数指定用于连接到远程数据库的远程服务器的名称。远程服务器是用 CREATE SERVER 语句创建的。此参数需要值。请参见“CREATE SERVER 语句”一节第 481 页。

注释

此过程使用可从远程数据库中迁移的外键列表来填充 dbo.migrate_remote_fks_list 表。可以从此表中删除不想迁移的外键的行。

另一种选择是，使用 sa_migrate 系统过程在一步中迁移所有的表。

此系统过程与其它几个迁移系统过程一起使用。sa_migrate_create_fks 系统过程注释部分的注意中包含迁移过程的列表，以及执行它们必须遵循的顺序。请参见“sa_migrate_create_fks 系统过程”一节第 859 页。

权限

无

副作用

无

另请参见

- “将数据库迁移到 SQL Anywhere”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_migrate 系统过程”一节第 857 页
- “sa_migrate_create_remote_table_list 系统过程”一节第 861 页
- “sa_migrate_create_tables 系统过程”一节第 863 页
- “sa_migrate_data 系统过程”一节第 864 页
- “sa_migrate_create_fks 系统过程”一节第 859 页
- “sa_migrate_drop_proxy_tables 系统过程”一节第 865 页

示例

以下语句创建远程数据库中的外键的列表。

```
CALL sa_migrate_create_remote_fks_list( 'server_a' );
```

sa_migrate_create_remote_table_list 系统过程

填充 dbo.migrate_remote_table_list 表。

语法

sa_migrate_create_remote_table_list(
i_server_name
[, *i_table_name*
[, *i_owner_name*
[, *i_database_name* **]]]**
)

参数

- **i_server_name** 使用此 VARCHAR(128) 参数指定用于连接到远程数据库的远程服务器的名称。远程服务器是用 CREATE SERVER 语句创建的。此参数需要值。请参见“CREATE SERVER 语句”一节第 481 页。
- **i_table_name** 使用此可选的 VARCHAR(128) 参数指定要迁移的表的名称，如果要迁移所有表，则值为 NULL。缺省值为 NULL。不要将 *i_table_name* 和 *i_owner_name* 参数都指定为 NULL。
- **i_owner_name** 使用此可选的 VARCHAR(128) 参数指定远程数据库中您想迁移其所拥有表的用户，如果想迁移所有表，则为 NULL。缺省值为 NULL。不要将 *i_table_name* 和 *i_owner_name* 参数都指定为 NULL
- **i_database_name** 使用此可选的 VARCHAR(128) 参数指定要从中迁移表的远程数据库的名称。此参数缺省为 NULL。从 Adaptive Server Enterprise 和 Microsoft SQL Server 数据库中迁移表时，必须指定数据库名。

注释

此过程使用可从远程数据库中迁移的表的列表来填充 dbo.migrate_remote_table_list 表。可以从此表中删除不想迁移的远程表的行。

如果不希望所有的迁移表都属于目标 SQL Anywhere 数据库中的同一个所有者，则必须为每个拥有您想迁移的表的用户执行此过程。

另一种选择是，使用 sa_migrate 系统过程在一步中迁移所有的表。

小心
不要将 *i_table_name* 和 *i_owner_name* 参数都指定为 NULL。为 *i_table_name* 和 *i_owner_name* 参数同时提供 NULL 值将迁移数据库中的所有表，包括系统表。此外，远程数据库中那些名称相同但所有者不同的表在目标数据库中均属同一所有者。建议您一次迁移与一个所有者关联的表。

此系统过程与其它几个迁移系统过程一起使用。sa_migrate_create_fks 系统过程注释部分的注意项中包含迁移过程的列表，以及执行它们时必须遵循的顺序。请参见“sa_migrate_create_fks 系统过程”一节第 859 页。

权限

无

副作用

无

另请参见

- “将数据库迁移到 SQL Anywhere”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_migrate 系统过程”一节第 857 页
- “sa_migrate_create_tables 系统过程”一节第 863 页
- “sa_migrate_data 系统过程”一节第 864 页
- “sa_migrate_create_remote_fks_list 系统过程”一节第 860 页
- “sa_migrate_create_fks 系统过程”一节第 859 页
- “sa_migrate_drop_proxy_tables 系统过程”一节第 865 页

示例

以下语句创建属于远程数据库中的用户 remote_a 的表列表。

```
CALL sa_migrate_create_remote_table_list( 'server_a', NULL, 'remote_a',
NULL );
```

sa_migrate_create_tables 系统过程

为 dbo.migrate_remote_table_list 表中列出的每个表创建代理表和基表。

语法

sa_migrate_create_tables(*i_table_owner* **)**

参数

- **i_table_owner** 使用此 VARCHAR(128) 参数指定在目标 SQL Anywhere 数据库中拥有迁移表的用户。此用户是用 GRANT CONNECT 语句创建的。此参数需要值。请参见“GRANT 语句”一节第 595 页。

注释

此过程为表 dbo.migrate_remote_table_list（使用 sa_migrate_create_remote_table_list 过程创建）中列出的每个表创建基表和代理表。这些代理表和基表由 *i_table_owner* 参数指定的用户所拥有。此过程还为远程表在远程数据库中的新表创建同样的主键索引和其它索引。

如果不希望目标 SQL Anywhere 数据库中的所有迁移表均属于同一个所有者，则必须对每个将拥有迁移表的用户执行 sa_migrate_create_remote_table_list 过程和 sa_migrate_create_tables 过程。

另一种选择是，使用 sa_migrate 系统过程在一步中迁移所有表。

此系统过程与其它几个迁移系统过程一起使用。sa_migrate_create_fks 系统过程注释部分的注意项中包含迁移过程的列表，以及执行它们时必须遵循的顺序。请参见“sa_migrate_create_fks 系统过程”一节第 859 页。

权限

无

副作用

无

另请参见

- “将数据库迁移到 SQL Anywhere”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_migrate 系统过程”一节第 857 页
- “sa_migrate_create_remote_table_list 系统过程”一节第 861 页
- “sa_migrate_data 系统过程”一节第 864 页
- “sa_migrate_create_remote_fks_list 系统过程”一节第 860 页
- “sa_migrate_create_fks 系统过程”一节第 859 页
- “sa_migrate_drop_proxy_tables 系统过程”一节第 865 页

示例

以下语句在目标 SQL Anywhere 数据库中创建基表和代理表。这些表属于用户 local_a。

```
CALL sa_migrate_create_tables( 'local_a' );
```

sa_migrate_data 系统过程

将远程数据库表中的数据迁移到目标 SQL Anywhere 数据库中。

语法

sa_migrate_data(*i_table_owner* **)**

参数

- **i_table_owner** 使用此 VARCHAR(128) 参数指定在目标 SQL Anywhere 数据库中拥有迁移表的用户。此用户是用 GRANT CONNECT 语句创建的。此参数需要值。请参见“GRANT 语句”一节第 595 页。

注释

此过程为属于 *i_table_owner* 参数所指定用户的表将数据从远程数据库迁移到 SQL Anywhere 数据库中。

当目标 SQL Anywhere 数据库中的表不是全都属于同一个所有者时，则必须为拥有您想迁移的数据的表的每个用户执行此过程。

另一种选择是，使用 sa_migrate 系统过程在一步中迁移所有表。

此系统过程与其它几个迁移系统过程一起使用。sa_migrate_create_fks 系统过程注释部分的注意项中包含迁移过程的列表，以及执行它们时必须遵循的顺序。请参见“sa_migrate_create_fks 系统过程”一节第 859 页。

权限

无

副作用

无

另请参见

- “将数据库迁移到 SQL Anywhere”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_migrate 系统过程”一节第 857 页
- “sa_migrate_create_remote_table_list 系统过程”一节第 861 页
- “sa_migrate_create_tables 系统过程”一节第 863 页
- “sa_migrate_create_remote_fks_list 系统过程”一节第 860 页
- “sa_migrate_create_fks 系统过程”一节第 859 页
- “sa_migrate_drop_proxy_tables 系统过程”一节第 865 页

示例

以下语句为属于用户 local_a 的表将数据迁移到目标 SQL Anywhere 数据库中。

```
CALL sa_migrate_data( 'local_a' );
```

sa_migrate_drop_proxy_tables 系统过程

删除为进行迁移而创建的代理表。

语法

sa_migrate_drop_proxy_tables(*i_table_owner* **)**

参数

- **i_table_owner** 使用此 VARCHAR(128) 参数指定在目标 SQL Anywhere 数据库中拥有代理表的用户。此用户是用 GRANT CONNECT 语句创建的。此参数需要值。请参见“GRANT 语句”一节第 595 页。

注释

此过程删除为进行迁移而创建的代理表。拥有这些代理表的用户是由 *i_table_owner* 参数指定的。

如果迁移的表不是全都属于目标 SQL Anywhere 数据库中的同一个用户，则必须为每个用户调用此过程以删除所有代理表。

另一种选择是，使用 sa_migrate 系统过程在一步中迁移所有表。

此系统过程与其它几个迁移系统过程一起使用。sa_migrate_create_fks 系统过程注释部分的注意项中包含迁移过程的列表，以及执行它们时必须遵循的顺序。请参见“sa_migrate_create_fks 系统过程”一节第 859 页。

权限

无

副作用

无

另请参见

- “将数据库迁移到 SQL Anywhere”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_migrate 系统过程”一节第 857 页
- “sa_migrate_create_remote_table_list 系统过程”一节第 861 页
- “sa_migrate_create_tables 系统过程”一节第 863 页
- “sa_migrate_data 系统过程”一节第 864 页
- “sa_migrate_create_remote_fks_list 系统过程”一节第 860 页
- “sa_migrate_create_fks 系统过程”一节第 859 页

示例

以下语句删除目标 SQL Anywhere 数据库中属于用户 local_a 的代理表。

```
CALL sa_migrate_drop_proxy_tables( 'local_a' );
```

sa_nchar_terms 系统过程

将 NCHAR 字符串分解为多个语词，然后返回每个语词及其位置。

语法

sa_nchar_terms('*char-string*' [, '*text-config-name*' [, '*owner*']]]
)

参数

- **char-string**　所分析的 NCHAR 字符串。
- **text-config-name**　处理字符串时应用的文本配置对象。缺省值为 'default_nchar'。
- **owner**　所指定的文本配置对象的所有者。缺省值为 DBA。

注释

可以使用此系统过程了解应用文本配置对象的设置时，会如何对字符串进行解释。如果想知道索引过程中哪些术语会被删除或者会从查询字符串中删除哪些术语，这一点很有用。

权限

None

副作用

None

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本配置对象”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_char_terms 系统过程”一节第 798 页

示例

此系统过程的语法与 sa_char_terms 系统过程类似。请参见“sa_char_terms 系统过程”一节第 798 页中的 "示例" 部分。

sa_performance_diagnostics 系统过程

返回所有连接的请求计时信息的汇总（前提是数据库服务器启用了请求计时记录）。

语法

sa_performance_diagnostics( )

结果集

列名	数据类型	说明
Number	INT	连接的 ID 号。
Name	VARCHAR(255)	连接的名称。
Userid	VARCHAR(255)	连接的用户 ID。
DBNumber	INT	数据库的 ID 号。
LoginTime	TIMESTAMP	建立连接的日期和时间。
TransactionStartTime	TIMESTAMP	在执行 COMMIT 或 ROLLBACK 后首次修改数据库的时间，或为空字符串（如果自上次执行 COMMIT 或 ROLLBACK 后未对数据库进行任何修改）。
LastReqTime	TIMESTAMP	指定连接的最后请求开始的时间。
ReqType	VARCHAR(255)	最后请求的类型。

列名	数据类型	说明
ReqStatus	VARCHAR(255)	请求的状态。它可以是以下各值之一： ● **Idle** 连接当前未处理请求。 ● **Unscheduled** 连接有工作要做并且正在等待工作线程。 ● **BlockedIO** 连接被阻塞，正在等待 I/O。 ● **BlockedContention** 连接被阻塞，正在等待访问共享数据库服务器数据结构。 ● **BlockedLock** 连接被阻塞，正在等待锁定的对象。 ● **Executing** 连接正在执行请求。
ReqTimeUnscheduled	DOUBLE	等待调度所用的时间。
ReqTimeActive	DOUBLE	等待处理请求所用的时间。
ReqTimeBlockIO	DOUBLE	等待 I/O 完成所用的时间。
ReqTimeBlockLock	DOUBLE	等待锁所用的时间。
ReqTimeBlockContention	DOUBLE	等待原子访问所用的时间。
ReqCountUnscheduled	INT	等待调度的次数。
ReqCountActive	INT	处理的请求数。
ReqCountBlockIO	INT	等待 I/O 完成的次数。
ReqCountBlockLock	INT	等待锁的次数。
ReqCountBlockContention	INT	等待原子访问的次数。
LastIdle	INT	请求间隔时间数。
BlockedOn	INT	如果没有阻塞当前连接，则该值为 0。如果阻塞了当前连接，则为由于锁定冲突而阻塞连接的连接号。
UncommitOp	INT	未提交操作的数量。
CurrentProcedure	VARCHAR(255)	连接当前正在执行的过程。如果该连接正在执行嵌套过程调用，则该名称为当前过程的名称。如果未在执行任何过程，则返回一个空字符串。

列名	数据类型	说明
EventName	VARCHAR(255)	在该连接运行事件处理程序情况下的相关事件的名称。否则，结果为 NULL。
CurrentLineNumber	INT	连接正在执行的过程或复合语句的当前行号。可使用 CurrentProcedure 属性标识该过程。如果该行是来自客户端的复合语句的一部分，则返回一个空字符串。
LastStatement	LONG VARCHAR	当前连接的最近预准备 SQL 语句。
LastPlanText	LONG VARCHAR	在连接上执行的最后一个查询的详细文本计划。
AppInfo	LONG VARCHAR	有关建立连接的客户端的信息。对于 HTTP 连接，这包括有关浏览器的信息。对于使用较旧版本 jConnect 或 Open Client 的连接，信息可能是不完整的。API 值可以是 DBLIB、ODBC、OLEDB 或 ADO.NET。
LockCount	INT	连接所持有的锁的个数。
SnapshotCount	INT	与连接关联的快照数。

注释

sa_performance_diagnostics 系统过程返回一个包含一组请求计时属性和统计信息的结果集（前提是已通知服务器搜集这些信息）。在调用 sa_performance_diagnostics 之前，必须在数据库服务器上打开请求计时信息的记录。要达到此目的，可在启动数据库服务器时指定 -zt 选项，也可执行以下内容：

```
CALL sa_server_option( 'RequestTiming','ON' );
```

权限

需要 DBA 权限

副作用

无

另请参见

- “-zt 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_performance_statistics 系统过程”一节第 870 页
- “sa_server_option 系统过程”一节第 886 页

示例

可以执行以下查询以标识花费长时间等待数据库服务器请求完成的连接。

```
SELECT  Number, Name,
      CAST( DATEDIFF( second, LoginTime, CURRENT TIMESTAMP ) AS DOUBLE ) AS
T,
      ReqTimeActive / T AS PercentActive
FROM  dbo.sa_performance_diagnostics()
WHERE PercentActive > 10.0
ORDER BY PercentActive DESC;
```

以下示例查找当前正在执行且执行时间已超过 60 秒的所有请求：

```
SELECT  Number, Name,
      CAST( DATEDIFF( second, LastReqTime, CURRENT TIMESTAMP ) AS DOUBLE ) AS
ReqTime
FROM  dbo.sa_performance_diagnostics()
WHERE ReqStatus <> 'IDLE' AND ReqTime > 60.0
ORDER BY ReqTime DESC;
```

sa_performance_statistics 系统过程

返回所有连接的内存诊断统计信息的汇总（前提是数据库服务器启用了请求计时记录）。

语法

sa_performance_statistics( )

结果集

列名	数据类型	说明
DBNumber	INT	数据库的 ID 号。
ConnNumber	INT	一个表示连接 ID 的 INTEGER 类型值。如果 Type 是 Server，则返回 NULL。
PropNum	INT	连接属性编号。
PropName	VARCHAR(255)	连接属性名称。
Value	INT	连接属性值。

注释

sa_performance_statistics 系统过程返回一个包含一组内存诊断统计信息的结果集（前提是已通知服务器搜集这些信息）。在调用 sa_performance_statistics 之前，必须在数据库服务器上打开内存诊断统计信息的记录。要达到此目的，可在启动数据库服务器时指定 -zt 选项，也可执行以下内容：

```
CALL sa_server_option( 'RequestTiming','ON' );
```

权限

需要 DBA 权限

副作用

无

另请参见

- “-zt 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_performance_diagnostics 系统过程”一节第 867 页
- “sa_server_option 系统过程”一节第 886 页

示例

以下示例将所有性能统计信息卸载到名为 *dump_stats.txt* 的文本文件：

```
UNLOAD
   SELECT CURRENT TIMESTAMP, *
   FROM sa_performance_statistics()
   TO 'dump_stats.txt'
   APPEND ON;
```

sa_post_login_procedure 系统过程

确定用户口令是否即将到期。

语法

sa_post_login_procedure( )

参数

None

结果集

sa_post_login_procedure 系统过程返回以下结果：

列名	数据类型	说明
message_text	VARCHAR(255)	如果 message_action 为 1，message_text 将返回并显示消息。 如果 message_action 为 0，则 message_text 为 NULL。
message_action	INTEGER	口令是否即将到期（1= 是，0= 否）。

注释

post_login_procedure 数据库选项的缺省设置为 sa_post_login_procedure 系统过程。请参见“post_login_procedure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

sa_post_login_procedure 通过用户的 password_life_time 和 password_grace_time 登录策略选项值以及当前的日期和时间，来确定用户口令是否即将到期。如果是，则在结果集中返回要显示给用户的消息。

权限

需要 DBA 权限

副作用

None

另请参见

- “管理登录策略概述”一节 《SQL Anywhere 服务器 - 数据库管理》

sa_procedure_profile 系统过程

报告已经在数据库中执行的过程、函数、事件或触发器中每一行的执行时间信息。此过程提供的信息与 Sybase Central 中 [分析] 选项卡上的信息相同。

语法

```
sa_procedure_profile(
  [ filename
  [, save_to_file ] ]
)
```

参数

- **filename** 使用此可选的 LONG VARCHAR(128) 参数指定应将分析信息保存到的文件，或应从其装载分析信息的文件。有关保存和装载分析信息的详细信息，请参见以下的注释部分。
- **save_to_file** 使用此可选的 INT(1) 参数指定是将分析信息保存到文件，还是从先前存储的文件装载分析信息。

结果集

列名	数据类型	说明
object_type	CHAR(1)	对象的类型。有关可能的对象类型的列表，请参见以下的注释部分。
object_name	CHAR(128)	存储过程、函数、事件或触发器的名称。如果 object_type 为 C 或 D，则这是为其定义了系统触发器的外键的名称。
owner_name	CHAR(128)	对象的所有者。
table_name	CHAR(128)	与触发器关联的表（对于其它对象类型，该值为 NULL）。
line_num	UNSIGNED INTEGER	过程中的行号。
executions	UNSIGNED INTEGER	该行的执行次数。

列名	数据类型	说明
millisecs	UNSIGNED INTEGER	该行的执行时间（以毫秒计）。
percentage	DOUBLE	特定行所需要的总执行时间的百分比。
foreign_owner	CHAR(128)	拥有系统触发器外表的数据库用户。
foreign_table	CHAR(128)	系统触发器外表的名称。

注释

此过程可用于：

- **返回详细的过程分析信息** 要实现此目的，可以只调用该过程，而不指定任何参数。
- **将详细的过程分析信息保存到文件** 要实现此目的，必须包括 *filename* 参数并将 *save_to_file* 参数指定为 1。
- **从先前保存的文件装载详细的过程分析信息** 要实现此目的，必须包括 *filename* 参数并将 *save_to_file* 参数指定为 0（或保留不变，因为缺省值为 0）。当以此方式使用该过程时，创建装载文件的数据库必须是正从其运行该过程的同一数据库；否则，结果可能不可用。

结果集中包含过程、触发器、函数和事件中每一行的执行时间信息，以及这些行使用的时间占总过程执行时间的百分比信息，因此可以使用此分析信息来调整那些可能会降低性能的较慢过程。

在可以分析数据库之前，必须启用分析。请参见“启用过程分析”一节 《SQL Anywhere 服务器 - SQL 的用法》。

结果集的 object_type 字段可以是：

- **P** 存储过程
- **F** 函数
- **E** 事件
- **T** 触发器
- **C** ON UPDATE 系统触发器
- **D** ON DELETE 系统触发器

如果想得到汇总信息（而不是每个执行的逐行详细信息），请改用 sa_procedure_profile_summary 过程。

权限

需要 DBA 权限

副作用

无

另请参见

- “sa_server_option 系统过程”一节第 886 页
- “sa_procedure_profile_summary 系统过程”一节第 874 页

示例

以下语句返回已经在数据库中执行的每个过程、函数、事件或触发器中每一行的执行时间：

```
CALL sa_procedure_profile( );
```

以下语句返回与以上示例相同的详细过程分析信息，并将其保存到名为 *detailedinfo.txt* 的文件：

```
CALL sa_procedure_profile( "detailedinfo.txt", 1 );
```

以下任一语句均可用来从名为 *detailedinfoOLD.txt* 的文件装载详细过程分析信息：

```
CALL sa_procedure_profile( "detailedinfoOLD.txt", 0 );

CALL sa_procedure_profile( "detailedinfoOLD.txt" );
```

sa_procedure_profile_summary 系统过程

报告已经在数据库中执行的所有过程、函数、事件或触发器的执行时间的汇总信息。此过程为这些对象提供的信息与 Sybase Central 中 [分析] 选项卡上的信息相同。

语法

sa_procedure_profile_summary(
 [*filename*
 [, *save_to_file*]]
)

参数

- **filename** 使用此可选的 LONG VARCHAR(128) 参数指定应将分析信息保存到的文件，或应从其装载分析信息的文件。有关保存和装载分析信息的详细信息，请参见以下的注释部分。
- **save_to_file** 使用此可选的 INT(1) 参数指定是将汇总信息保存到文件，还是从先前保存的文件装载汇总信息。

结果集

列名	数据类型	说明
object_type	CHAR(1)	对象的类型。有关可能的对象类型的列表，请参见以下的注释部分。
object_name	CHAR(128)	存储过程、函数、事件或触发器的名称。
owner_name	CHAR(128)	对象的所有者。
table_name	CHAR(128)	与触发器关联的表（对于其它对象类型，该值为 NULL）。

列名	数据类型	说明
executions	UNSIGNED INTEGER	每个过程的执行次数。
millisecs	UNSIGNED INTEGER	过程的执行时间，以毫秒计。
foreign_owner	CHAR(128)	拥有系统触发器外表的数据库用户。
foreign_table	CHAR(128)	系统触发器外表的名称。

注释

此过程可用于：

- **返回当前汇总信息** 要实现此目的，可以只调用该过程，而不指定任何参数。
- **将当前汇总信息保存到文件** 要实现此目的，必须包括 *filename* 参数并将 *save_to_file* 参数指定为 1。
- **从文件装载存储的汇总信息** 要实现此目的，必须包括 *filename* 参数并将 *save_to_file* 参数指定为 0（或保留不变，因为缺省值为 0）。当以此方式使用该过程时，创建装载文件的数据库必须是正从其运行该过程的同一数据库；否则，结果可能不可用。

由于此过程返回有关存储过程、函数、事件和触发器的使用频率和效率的信息，因此您可以使用此信息来调整较慢的过程以提高数据库性能。

必须先启用分析，之后才能分析数据库。请参见“启用过程分析”一节 《SQL Anywhere 服务器 - SQL 的用法》。

结果集的 object_type 字段可以是：

- **P** 存储过程
- **F** 函数
- **E** 事件
- **T** 触发器
- **S** 系统触发器
- **C** ON UPDATE 系统触发器
- **D** ON DELETE 系统触发器

如果想得到每个执行的逐行详细信息（而不是汇总信息），请改用 sa_procedure_profile 过程。

权限

需要 DBA 权限

副作用

无

另请参见

- “sa_server_option 系统过程”一节第 886 页
- “sa_procedure_profile 系统过程”一节第 872 页

示例

以下语句返回已经在数据库中执行的所有过程、函数、事件或触发器的执行时间：

```
CALL sa_procedure_profile_summary( );
```

以下语句返回与以上示例相同的汇总信息，并将其保存到名为 *summaryinfo.txt* 的文件：

```
CALL sa_procedure_profile_summary( "summaryinfo.txt", 1 );
```

以下任一语句均可用来从名为 *summaryinfoOLD.txt* 的文件装载存储的汇总信息：

```
CALL sa_procedure_profile_summary( "summaryinfoOLD".txt, 0 );
CALL sa_procedure_profile_summary( "summaryinfoOLD.txt" );
```

sa_recompile_views 系统过程

找到存储在目录中的没有列定义的视图定义并创建列定义。

语法

sa_recompile_views([*ignore_errors*] **)**

参数

- **ignore_errors** 使用此可选的 INTEGER 参数指定在重新编译期间是否返回错误。如果指定 0，则为列定义失败的每个视图返回错误。如果指定 1 或除 0 之外的任何值，则不返回错误。如果未指定值，则缺省情况下使用 0。

注释

此过程用于查找目录中没有列定义的视图，并执行带有 RECOMPILE 子句的 ALTER VIEW 语句以创建列定义。此过程对每个没有列定义的视图执行上述操作，直到没有视图需要编译或无法创建剩下的列定义为止。如果此过程无法编译任何视图，则会报告错误。通过将一个非零参数指定给此过程，可取消错误。

> **小心**
> sa_recompile_views 系统过程应该只在 *reload.sql* 脚本中调用。此过程由卸载实用程序 (dbunload) 使用，不应显式使用此过程。

sa_recompile_views 系统过程不会尝试重新编译实例化视图或任何标记为 [已禁用] 的视图。

权限

需要 DBA 权限

副作用

对于每个不具有 [有效] 状态的常规视图，则执行 `ALTER VIEW owner.viewname ENABLE` 语句并导致自动提交。

另请参见

- “常规视图状态”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “force_view_creation 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “ALTER VIEW 语句”一节第 387 页

sa_refresh_text_indexes 系统过程

刷新所有定义为 MANUAL REFRESH 或 AUTO REFRESH 的文本索引。

语法

sa_refresh_text_indexes( )

注释

sa_refresh_text_indexes 系统过程刷新所有定义为 MANUAL REFRESH 或 AUTO REFRESH 的文本索引。它不刷新定义为 IMMEDIATE REFRESH（缺省值）的文本索引，因为基础表中的数据发生变化时，这些索引便会随之更改。

权限

需要 DBA 权限

副作用

自动提交

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本配置对象”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “DROP TEXT INDEX 语句”一节第 560 页
- “REFRESH TEXT INDEX 语句”一节第 668 页
- “TRUNCATE 语句”一节第 727 页
- “SYSTEXTIDX 系统视图”一节第 980 页
- “sa_text_index_stats 系统过程”一节第 906 页
- “sa_text_index_vocab 系统过程”一节第 908 页

示例

以下语句刷新数据库中的所有的 MANUAL 和 AUTO REFRESH 文本索引：

```
CALL sa_refresh_text_indexes( );
```

sa_refresh_materialized_views 系统过程

初始化处于未初始化状态的所有实现化视图 (Materialized View)。

语法

sa_refresh_materialized_views([*ignore_errors*] **)**

参数

- **ignore_errors** 使用此可选的 INTEGER 参数指定在重新编译期间是否返回错误。如果指定 0，则为列定义失败的每个视图返回错误。如果指定 1 或除 0 之外的任何值，则不返回错误。如果未指定值，则缺省情况下使用 0。

注释

实现化视图 (Materialized View) 可能处于未初始化的状态，因为其刚被创建、刚被重新启用，或由于错误而导致上次尝试初始化或刷新失败。sa_refresh_materialized_views 系统过程扫描数据库以找到所有此类实现化视图 (Materialized View)，并尝试初始化它们。如果该过程在初始化实例化视图时遇到错误，它会继续尝试处理剩余的未初始化视图。

还可以使用 REFRESH MATERIALIZED VIEW 语句来初始化实现化视图 (Materialized View)。

权限

DBA 特权

副作用

无

另请参见

- “REFRESH MATERIALIZED VIEW 语句”一节第 665 页
- “刷新手动视图”一节 《SQL Anywhere 服务器 - SQL 的用法》

sa_remove_tracing_data 系统过程

永久删除诊断跟踪表中与指定的记录（跟踪）会话 ID 相关的所有记录。

语法

sa_remove_tracing_data(*log_session_id* **)**

参数

- **log_session_id** 使用此 INTEGER 参数指定要为其删除数据的记录会话的 ID。

注释

如果没有指定的 *log_session_id* 的记录，则该过程无效。该过程不返回值。

权限

需要 DBA 权限

副作用

导致在完成后就提交，即使未找到指定的 *log_session_id* 的任何记录。

另请参见

- “诊断跟踪表”一节第 767 页

sa_report_deadlocks 系统过程

从数据库服务器创建的内部缓冲区中检索有关死锁的信息。

语法

sa_report_deadlocks( )

结果集

列名	数据类型	说明
snapshotId	BIGINT	死锁实例（与特定死锁相关的所有行都具有相同的 ID）。
snapshotAt	TIMESTAMP	发生死锁的时间。
waiter	INT	正等待的连接的连接句柄。
who	VARCHAR(128)	与正等待的连接关联的用户 ID。
what	LONG VARCHAR	正等待的连接所执行的命令。 仅当通过在数据库服务器命令行中使用 -zl 选项或通过使用 sa_server_option 系统过程打开捕获最近预准备 SQL 语句的功能时，此信息才可用。
object_id	UNSIGNED BIGINT	包含该行的表的对象 ID。
record_id	BIGINT	关联行的行 ID。
owner	INT	拥有正等待的锁的连接的连接句柄。
is_victim	BIT	标识回退的事务。
rollback_operation_count	UNSIGNED INT	未提交操作的数量，如果事务回退，这些操作有可能丢失。

注释

当 log_deadlocks 选项设置为 On 时，数据库服务器在内部缓冲区中记录有关死锁的信息。可以使用 sa_report_deadlocks 系统过程查看日志中的信息。

权限

需要 DBA 权限

副作用

无

另请参见

- “了解系统事件”一节 《SQL Anywhere 服务器 - 数据库管理》
- “log_deadlocks 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》
- “确定被阻塞的连接”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “-zl 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_server_option 系统过程”一节第 886 页

sa_reset_identity 系统过程

允许为表设置下一个标识值。使用它可以改变将要插入的下一行的自动增量值。

语法

sa_reset_identity(
tbl_name,
owner_name,
new_identity
)

参数

- **tbl_name** 使用此 CHAR(128) 参数指定要为其重置标识值的表。如果未指定表的所有者，则 *tbl_name* 必须唯一标识数据库中的表。
- **owner_name** 使用此 CHAR(128) 参数指定要为其重置标识值的表的所有者。
- **new_identity** 使用此 BIGINT 参数指定一个要从其开始进行自动增量的值。

注释

为插入到表中的行生成的下一个标识值是 *new_identity* + 1。

不发生检查以查看 *new_identity* + 1 是否与表中的现有行冲突。例如，如果指定 *new_identity* 为 100，则插入的下一行的标识值将为 101。然而，如果表中已经存在 101，则行插入失败。

如果 *owner* 未指定或为 NULL，则 *tbl_name* 必须唯一标识数据库中的表。

权限

需要 DBA 权限

副作用

导致在值更新后执行检查点操作

示例

以下语句将下一个标识值重置为 101：

```
CALL sa_reset_identity( 'Employees', 'DBA', 100 );
```

sa_rowgenerator 系统过程

返回包含指定起始值和结束值之间的行的结果集。

语法

sa_rowgenerator(
[*rstart*
[, *rend*
[, *rstep*]]]
)

参数

- **rstart** 使用此可选的 INTEGER 参数指定起始值。缺省值是 0。
- **rend** 使用此可选的 INTEGER 参数指定结束值。缺省值是 100。
- **rstep** 使用此可选的 INTEGER 参数指定序列值的增量。缺省值是 1。

结果集

列名	数据类型	说明
row_num	INTEGER	序列号。

注释

此 sa_rowgenerator 过程可用在查询的 FROM 子句中，以生成一个数字序列。此过程可以作为使用 RowGenerator 系统表的替代方法。可以为类似如下的任务使用 sa_rowgenerator：

- 在结果集中为一定数目的行生成测试数据。
- 生成一个结果集，其中包含为每个范围中的值生成的行。例如，可以为一个月的每一天生成一行，也可以生成邮政编码范围。
- 生成一个查询，其结果集具有指定数目的行。这对于测试查询性能可能有用。

您可以使用以下语句模拟 RowGenerator 表的行为：

```
SELECT row_num FROM sa_rowgenerator( 1, 255 );
```

权限

无

副作用

无

另请参见

- “RowGenerator 表 (dbo)”一节第 782 页

示例

以下查询返回的结果集包含当前月的每一天的一行记录。

```
SELECT DATEADD( day, row_num-1,
         YMD( DATEPART( year, CURRENT DATE ),
             DATEPART( month, CURRENT DATE ), 1 ) )
    AS day_of_month
    FROM sa_rowgenerator( 1, 31, 1 )
    WHERE DATEPART( month, day_of_month ) =
       DATEPART( month, CURRENT DATE )
    ORDER BY row_num;
```

以下查询显示有多少雇员生活在邮政编码范围为 (0-9999)、(10000-19999)、...、及 (90000-99999) 的地方。其中一些范围内没有雇员，这时会出现警告：**集合函数中的空值已消除** (-109)。sa_rowgenerator 过程可以用来生成这些范围，即使该邮政编码范围没有任何雇员。

```
SELECT row_num AS r1, row_num+9999
 AS r2, COUNT( PostalCode ) AS zips_in_range
FROM sa_rowgenerator( 0, 99999, 10000 ) D LEFT JOIN Employees
   ON PostalCode BETWEEN r1 AND r2
GROUP BY r1, r2
ORDER BY 1;
```

以下示例生成 10 行数据并将其插入到 NewEmployees 表中：

```
INSERT INTO NewEmployees ( ID, Salary, Name )
SELECT row_num,
   CAST( RAND() * 1000 AS INTEGER ),
   'Mary'
FROM sa_rowgenerator( 1, 10 );
```

以下示例使用 sa_rowgenerator 系统过程来创建包含所有整数的视图。此示例中的值 2147483647 表示 SQL Anywhere 中所支持的最大有符号整数。

```
CREATE VIEW Integers AS
SELECT row_num AS n
FROM sa_rowgenerator( 0, 2147483647, 1 );
```

以下示例使用 sa_rowgenerator 系统过程来创建包含日期从 0001-01-01 到 9999-12-31 的视图。此示例中的值 3652058 表示 0001-01-01 和 9999-12-31 之间的天数，其中 0001-01-01 和 9999-12-31 分别为 SQL Anywhere 中所支持的最早和最晚的日期。

```
CREATE VIEW Dates AS
SELECT DATEADD( day, row_num, '0001-01-01' ) AS d
FROM sa_rowgenerator( 0, 3652058, 1 );
```


sa_save_trace_data 系统过程

将跟踪数据保存到基表中。

语法

sa_save_trace_data( )

注释

当跟踪会话正在运行时，诊断数据存储在诊断跟踪表的临时版本中。当停止跟踪会话时，您指定是否要将跟踪数据永久存储在诊断跟踪的基表中。如果您不选择保存数据，则通过使用 sa_save_trace_data 系统过程仍可在会话停止后保存数据。

如果跟踪仍在进行中，则 sa_save_trace_data 系统过程将返回错误；必须停止跟踪才能使用此系统过程。

即使用户在停止跟踪时指定 WITHOUT SAVING，也可使用 sa_save_trace_data 系统过程。此外，必须从跟踪数据库调用此过程。

权限

需要 DBA 权限

副作用

自动提交。

另请参见

- “创建诊断跟踪会话”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “诊断跟踪表”一节第 767 页

sa_send_udp 系统过程

向指定地址发送一个 UDP 包。

语法

sa_send_udp(
destAddress,
destPort,
msg
)

参数

- **destAddress** 使用此 CHAR(254) 指定主机名或 IP 号。
- **destPort** 使用此 UNSIGNED SMALLINT 参数指定要使用的端口号。
- **msg** 使用此 LONG BINARY 参数指定要发送到指定地址的消息。如果此值是字符串，则必须用单引号括起来。

注释

此过程将单个 UDP 包发送到指定地址。如果消息发送成功，则返回 0；如果出现错误，则返回一个错误代码。错误代码可为以下代码之一：

- -1，当消息太大而无法在 UDP 套接字上发送时（取决于操作系统）或目标地址有问题时
- 操作系统返回的 Winsock/Posix 错误代码

如果 *msg* 参数包含二进制数据或比字符串复杂得多，则您可能想要使用变量。例如，

```
CREATE VARIABLE v LONG BINARY;
SET v='This is a UDP message';
SELECT dbo.sa_send_udp( '10.25.99.124', 1234, v );
DROP VARIABLE v;
```

此过程可用于 MobiLink 服务器启动的同步，以唤醒监听器实用程序 (*dblsn.exe*)。如果将 sa_send_udp 系统过程用作通知监听器的一种方式，则应向 UDP 包添加一个 1。这是服务器启动的同步的协议号。在 MobiLink 的将来版本中，使用新的协议版本可能会使监听器以不同方式运行。

权限

需要 DBA 权限

副作用

无

示例

以下示例向 IP 地址 10.25.99.196 的 2345 端口发送消息 "This is a test"：

```
CALL sa_send_udp( '10.25.99.196', 2345', 'This is a test' );
```

sa_server_messages 系统过程

允许您以结果集的形式从数据库服务器消息窗口返回消息。

语法

sa_server_messages([*first_msg*] [, *num_msgs*] **)**

参数

- **first_msg** 此可选的 UNSIGNED BIGINT 参数用于指定要返回的第一个或最后一个消息的 ID，这取决于 *num_msgs* 参数的符号。缺省值为 NULL，即表示如果 *num_msgs* 为 NULL 或非负值，则从列表的起始处开始搜索；如果 *num_msgs* 为负值，将从列表的末端之后开始搜索。
- **num_msgs** 此可选的 BIGINT 参数用于指定要返回的消息的数量。符号表示该请求是针对开始于 *first_msg* 的消息还是针对结束于 *first_msg* 的消息。缺省值为 NULL，表示所有从 *first_msg* 到列表末端的消息都将被返回。

结果集

列名	数据类型	说明
msg_id	UNSIGNED BIGINT	唯一的消息 ID。消息 ID 开始于 0。
msg_text	LONG VARCHAR	消息文本。
msg_time	TIMESTAMP	发出消息的时间。
msg_severity	VARCHAR(255)	消息的严重级。此列包含以下值之一： ● **INFO** 信息性消息。 ● **WARN** 警告。 ● **ERR** 错误。
msg_category	VARCHAR(255)	消息类别。此列包含以下值之一： ● **STARTUP** 与数据库服务器、数据库启动或关闭有关的消息。 ● **CHKPT** 与检查点有关的消息。 ● **MSG** 使用 MESSAGE 或 PRINT 语句生成的消息。 ● **DBA_MSG** 使用 MESSAGE 语句生成且需要具有 DBA 权限的消息，例如发送到事件日志的消息。 ● **CONN** 有关数据库服务器连接的消息。 ● **OTHER** 所有其它类型的消息。
msg_database	VARCHAR(255)	如果已将消息应用到某一特定的数据库，则为与消息相关的数据库名称。否则，为 NULL。

注释

如果消息数量超过了 MessageCategoryLimit 属性的值，将新消息发送到控制台时，相同类别或严重级的旧消息将被删除。因此，在结果集中可能有间隔，而且两个连续的行可能没有连续的消息 ID。

权限

无

副作用

无

另请参见

- MessageCategoryLimit 属性： “数据库服务器属性”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下命令将请求 100 条消息（从消息 ID 为 3 的消息开始）：

```
CALL sa_server_messages( 3, 100 );
```

以下命令请求到消息 4032 为止的 500 条消息（包括消息 4032）：

```
CALL sa_server_messages( 4032, -500 );
```

以下命令请求从消息 3 开始的所有消息：

```
CALL sa_server_messages( 3, NULL );
CALL sa_server_messages( 3 );
```

以下命令请求表中的前 100 条消息：

```
CALL sa_server_messages( NULL, 100 );
```

以下命令请求列表中最后 100 条消息：

```
CALL sa_server_messages( NULL, -100 );
```

以下命令请求列表中的所有消息：

```
CALL sa_server_messages( NULL, NULL );
CALL sa_server_messages( );
```

sa_server_option 系统过程

当服务器正在运行时替换服务器选项。

语法

sa_server_option(
opt,
val
)

参数

- **opt** 使用此 CHAR(128) 参数指定服务器选项名称。
- **val** 使用此 CHAR(128) 参数为服务器选项指定新值。

注释

使用此过程，数据库管理员可以临时替换某些数据库服务器选项，而不需要重新启动数据库服务器。

当服务器关闭时，使用此存储过程更改的选项值将重置为其缺省值。如果想在服务器每次启动时都更改选项值，则可以在数据库服务器启动时指定相应的数据库服务器选项（如果存在），下表中最右侧的列给出了这些选项。

以下选项设置可以更改：

选项名称	值	缺省值	服务器选项
CacheSizingStatistics	YES、NO	NO	“-cs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
CollectStatistics	YES、NO	YES	“-k 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
ConnsDisabled	YES、NO	NO	
ConnsDisabledForDB	YES、NO	NO	
ConsoleLogFile	*filename*		“-o 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
ConsoleLogMaxSize	*file-size*（以字节为单位）		“-on 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
DatabaseCleaner	ON、OFF	ON	
DebuggingInformation	YES、NO	NO	“-z 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
IdleTimeout	INTEGER（以分钟为单位）	240	“-ti 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
LivenessTimeout	INTEGER（以秒为单位）	120	“-tl 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
MessageCategoryLimit	INTEGER	400	
OptionWatchAction	MESSAGE、ERROR	MESSAGE	“监控选项设置”一节 《SQL Anywhere 服务器 - 数据库管理》
OptionWatchList	以逗号分隔的数据库选项列表		“监控选项设置”一节 《SQL Anywhere 服务器 - 数据库管理》
ProcedureProfiling	YES、NO、RESET、CLEAR	NO	
ProfileFilterConn	*connection-id*		
ProfileFilterUser	*user-id*		

选项名称	值	缺省值	服务器选项
QuittingTime	有效的日期和时间		“-tq 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
RememberLastPlan	YES、NO	NO	“-zp 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
RememberLastStatement	YES、NO	NO	“-zl 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
RequestFilterConn	*connection-id*, -1		
RequestFilterDB	*database-id*, -1		
RequestLogFile	*filename*		“-zo 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
RequestLogging	SQL、HOSTVARS、PLAN、PROCEDURES、TRIGGERS、OTHER、BLOCKS、REPLACE、ALL、YES、NONE、NO	NONE	“-zr 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
RequestLogMaxSize	*file-size*（以字节为单位）		“-zs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
RequestLogNumFiles	INTEGER		“-zn 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
RequestTiming	YES、NO	NO	“-zt 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
SecureFeatures	*feature-list*		“-sf 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
WebClientLogFile	*filename*		“-zoc 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
WebClientLogging	ON、OFF	OFF	

- **CacheSizingStatistics** 当设置为 YES 时，每当高速缓存大小发生变化时，数据库服务器消息窗口中就会显示高速缓存信息。请参见“-cs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **CollectStatistics** 当设置为 YES 时，数据库服务器搜集性能监控器统计信息。请参见“-k 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **ConnsDisabled** 当设置为 YES 时，不允许与数据库服务器上的任何数据库建立任何其它连接。
- **ConnsDisabledForDB** 当设置为 YES 时，不允许与当前数据库建立任何其它连接。
- **ConsoleLogFile** 用来记录数据库服务器消息日志信息的文件的名称。指定一个空字符串会停止将信息记录到文件。路径中的任何反斜线字符必须写两遍，因为这是 SQL 字符串。请参见“-o 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **ConsoleLogMaxSize** 用来记录数据库服务器消息日志信息的文件的最大大小（以字节为单位）。当数据库服务器消息日志文件达到该属性或 -on 服务器选项指定的大小后，通过附加的扩展名 *.old*，该文件被重命名（如果已经存在具有相同名称的文件，则将其替换）。然后，该数据库服务器消息日志文件被重新启动。请参见“-on 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **DatabaseCleaner** 除非有 iAnywhere 技术支持的建议，否则不要更改此选项的设置。另请参见“sa_clean_database 系统过程”一节第 800 页。
- **DebuggingInformation** 显示诊断消息和其它消息以用于疑难解答。消息显示在数据库服务器消息窗口中。请参见“-z 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **IdleTimeout** 断开未在指定分钟内提交请求的 TCP/IP 连接。这可防止不活动的连接无限期地保持锁定。请参见“-ti 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **LivenessTimeout** 定期通过客户端/服务器 TCP/IP 网络发送活动包，用以确认连接的完整性。如果网络服务器运行了一个 LivenessTimeout 周期但没有检测到活动包，则切断通信。请参见“-tl 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **MessageCategoryLimit** 设置可使用 sa_server_messages 系统过程检索的各个严重级和类别的最小消息数。请参见“sa_server_messages 系统过程”一节第 884 页。
- **OptionWatchAction** 指定数据库服务器在尝试设置列表中的选项时应采取的操作。支持的值为 MESSAGE 和 ERROR。当 OptionWatchAction 设置为 MESSAGE，并且设置了一个由 OptionWatchList 指定的选项时，数据库服务器消息窗口中将出现一条消息，指示正在设置的选项位于选项监视项目列表中。

  当 OptionWatchAction 设置为 ERROR 时，将返回错误，指示无法设置选项，因为该选项在选项监视项目列表中。

  可以通过执行以下的查询来查看此属性的当前设置：

  ```
  SELECT DB_PROPERTY( 'OptionWatchAction' );
  ```

- **OptionWatchList** 指定一个以逗号分隔的数据库选项列表，在设置其中的选项时数据库服务器会向您发出通知，或返回错误。字符串的长度不应超过 128 个字节。缺省情况下它是一个空字符串。例如，以下命令将 automatic_timestamp、float_as_double 和 tsql_hex_constant 选项添加到监视选项列表中。

  ```
  CALL dbo.sa_server_option( 'OptionWatchList','automatic_timestamp,
                                        float_as_double,tsql_hex_constant' )
  ```

  可以通过执行以下的查询来查看此属性的当前设置：

```
SELECT DB_PROPERTY( 'OptionWatchList' );
```

- **ProcedureProfiling** 控制存储过程、函数、事件和触发器的过程分析。过程分析向您显示存储过程、函数、事件和触发器的执行时间。还可以在 Sybase Central 的 [**数据库属性**] 窗口中设置过程分析选项。
  - **YES** 对您当前连接到的数据库启用过程分析。
  - **NO** 禁用过程分析，但使分析数据可供查看。
  - **RESET** 将分析计数器恢复到零，并且不改变 YES 或 NO 设置。
  - **CLEAR** 将分析计数器恢复到零，并且禁用过程分析。

    启用了分析后，可以使用 sa_procedure_profile_summary 和 sa_procedure_profile 系统过程从数据库中检索分析信息。请参见“使用系统过程进行过程分析”一节 《SQL Anywhere 服务器 - SQL 的用法》。
- **ProfileFilterConn** 指示数据库服务器捕获指定连接 ID 的分析信息，而不阻止其它连接使用该数据库。启用连接过滤后，`SELECT PROPERTY( 'ProfileFilterConn' )` 返回的值是被监控连接的连接 ID。如果未指定 ID 或禁用连接过滤，则返回的值为 -1。
- **ProfileFilterUser** 指示数据库服务器捕获特定用户 ID 的分析信息。
- **QuittingTime** 指示数据库服务器在指定的时间关闭。请参见“-tq 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。
- **RememberLastPlan** 指示数据库服务器捕获在连接上执行的最后一个查询的详细文本计划。此设置还由 -zp 服务器选项控制。请参见“-zp 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

  当启用了 RememberLastPlan 时，可以通过查询 LastPlanText 连接属性的值来获取在连接上执行的最后一个查询计划的文本表示。

  ```
  SELECT CONNECTION_PROPERTY( 'LastPlanText' );
  ```
- **RememberLastStatement** 指示数据库服务器为在服务器上运行的每个数据库捕获最近预准备 SQL 语句。对于存储过程调用，仅显示最外面的过程调用，不显示过程中的语句。

  当 RememberLastStatement 启用时，可以通过查询 LastStatement 连接属性的值来为连接获取 LastStatement 的当前值：

  ```
  SELECT CONNECTION_PROPERTY( 'LastStatement' );
  ```

  如果启用客户端语句高速缓存并且重用了某个高速缓存的语句，则此属性返回空字符串。

  有关详细信息，请参见“数据库服务器属性”一节 《SQL Anywhere 服务器 - 数据库管理》和“-zl 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

  当 RememberLastStatement 打开时，下面的语句为指定连接返回最近预准备语句：

  ```
  SELECT CONNECTION_PROPERTY( 'LastStatement', connection-id );
  ```

  sa_conn_activity 系统过程为所有连接都返回此相同的信息。

小心
当指定 -zl 或启用 RememberLastStatement 服务器设置时，任何用户都可以调用 sa_conn_activity 系统过程或获取 LastStatement 连接属性的值，以找到为任何其他用户最新准备的 SQL 语句。应慎用此选项，并在不需要时将其关闭。

- **RequestFilterConn** 对请求记录信息进行过滤以便只记录特定连接的信息。在对具有多个活动连接或多个数据库的数据库服务器进行监控时，这有助于减小请求日志文件的大小。可通过执行以下语句来获得连接 ID：

  ```
  CALL sa_conn_info( );
  ```

  一旦获得了连接 ID 后，即可执行以下语句来指定要记录日志信息的特定连接：

  ```
  CALL sa_server_option( 'RequestFilterConn', connection-id );
  ```

  在显式地重置过滤或关闭数据库服务器之前，过滤将一直有效。要重置过滤，请使用以下语句：

  ```
  CALL sa_server_option( 'RequestFilterConn', -1 );
  ```

- **RequestFilterDB** 对请求记录信息进行过滤以便只记录特定数据库的信息。在对具有多个数据库的服务器进行监控时，这有助于减小请求日志文件的大小。在连接到所需的数据库后，可通过执行以下语句来获得数据库 ID：

  ```
  SELECT CONNECTION_PROPERTY( 'DBNumber' );
  ```

  要指定仅记录特定数据库的信息，请执行以下语句：

  ```
  CALL sa_server_option( 'RequestFilterDB', database-id );
  ```

  在显式地重置过滤或关闭数据库服务器之前，过滤将一直有效。要重置过滤，请使用以下语句：

  ```
  CALL sa_server_option( 'RequestFilterDB', -1 );
  ```

- **RequestLogFile** 用来记录请求信息的文件的名称。指定一个空字符串会停止将请求信息记录到请求日志文件。如果启用请求记录，但请求日志文件未指定或设置为空字符串，则服务器会将请求记录到数据库服务器消息窗口中。路径中的任何反斜线字符必须写两遍，因为这是 SQL 字符串。请参见“-zo 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **RequestLogging** 此调用启用对发送到数据库服务器的各 SQL 语句的记录，以便用于疑难解答，此选项设置同数据库服务器选项 -zr 和 -zo 一起使用。值可以是以下项的组合，这些项由加号 (+) 或逗号分隔：

  - **PLAN** 启用执行计划的记录（简单形式）。如果启用过程 (PROCEDURES) 的记录，还将记录过程的执行计划。
  - **HOSTVARS** 启用主机变量值的记录。如果指定 HOSTVARS，还将记录针对 SQL 列出的信息。
  - **PROCEDURES** 启用从过程中执行的语句的记录。
  - **TRIGGERS** 启用从触发器中执行的语句的记录。
  - **OTHER** 启用 SQL 不包含的其它请求类型（如 FETCH 和 PREFETCH）的记录。然而，如果您指定 OTHER 但未指定 SQL，则其等效于指定 SQL+OTHER。包括 OTHER 可导致日志文件迅速增大，并可能对服务器性能造成负面影响。

- **BLOCKS** 启用显示何时在另一个连接上阻塞和解除阻塞某个连接的详细信息的记录。
- **REPLACE** 在记录开始时，使用同名的新（空）请求日志来替换现有的请求日志。否则，将打开现有的请求日志，并将新条目附加到文件的结尾。
- **ALL** 记录所有支持的信息。这等效于指定 SQL+PLAN+HOSTVARS+PROCEDURES +TRIGGERS+OTHER+BLOCKS。此设置可导致日志文件迅速增大，会对服务器的性能产生负面影响。
- **NO 或 NONE** 禁用记录请求日志。

可以通过执行以下的查询来查看此属性的当前设置：

```
SELECT PROPERTY( 'RequestLogging' );
```

有关详细信息，请参见“-zr 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》和“数据库服务器属性”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **RequestLogMaxSize** 用来记录请求记录信息的文件的最大大小（以字节为单位）。如果指定为 0，则对请求记录文件而言没有最大大小，且该文件也决不会被重命名。这是缺省值。

  当请求日志文件达到 sa_server_option 系统过程或 -zs 服务器选项指定的大小后，会用附加的扩展名 *.old* 对该文件进行重命名（如果已经存在这样一个文件，则用相同名称替换现有的文件）。然后，重新启动请求日志文件。请参见“-zs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **RequestLogNumFiles** 要保留的请求日志文件副本的数目。

  如果在较长一段时间内启用了请求记录，则请求日志文件可能会变得很大。-zn 允许您指定要保留的请求日志文件副本的数量。请参见“-zn 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **RequestTiming** 指示数据库服务器维护每个连接的计时信息。缺省情况下，此功能关闭。此功能打开时，数据库服务器会为每个连接维护累计计时器，该计时器表明连接在服务器的各个状态中所花费的具体时间。可以使用 sa_performance_diagnostics 系统过程来获得此计时信息的汇总，或者通过检查以下连接属性来检索各个值：

  - ReqCountUnscheduled
  - ReqTimeUnscheduled
  - ReqCountActive
  - ReqTimeActive
  - ReqCountBlockIO
  - ReqTimeBlockIO
  - ReqCountBlockLock
  - ReqTimeBlockLock
  - ReqCountBlockContention
  - ReqTimeBlockContention

  请参见“连接属性”一节 《SQL Anywhere 服务器 - 数据库管理》。

  当 RequestTiming 服务器属性设置为 on 时，为每个请求维护附加计数器会产生少量开销。请参见“-zt 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》和“sa_performance_diagnostics 系统过程”一节第 867 页。

- **SecureFeatures** 用于启用或禁用正在运行的数据库的安全功能。*feature-list* 是以逗号分隔的功能名称或功能集的列表。通过将其添加到安全功能列表，您可以保护（阻止）某些功能。要从安全功能列表中删除项目，请在安全功能名称前指定一个减号 (-)。有关有效 *feature-list* 值的列表，请参见“-sf 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

  为启用或禁用功能而进行的任何更改都会立即对连接生效。这些设置不影响执行 sa_server_option 系统过程的连接；必须断开并重新连接才能使设置生效。

  注意
  要使用 sa_server_option 系统过程启用或禁用功能，必须在启动数据库服务器时使用 -sk 选项指定一个密钥，然后将 secure_feature_key 数据库选项的值设置为 -sk 所指定的密钥（例如，`SET TEMPORARY OPTION secure_feature_key = 'j978kls12'`）。将 secure_feature_key 数据库选项设置为 -sk 值使您可以更改安全功能的设置。请参见“-sk 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》和“secure_feature_key [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

  例如，要禁用两个功能并启用第三个功能，应使用以下语法：

  ```
  CALL sa_server_option('SecureFeatures', 'CONSOLE_LOG,WEBCLIENT_LOG,-
  REQUEST_LOG' );
  ```

  执行了该语句后，CONSOLE_LOG 和 WEBCLIENT_LOG 被添加到安全功能列表中，而 REQUEST_LOG 将从列表中删除。

- **WebClientLogFile** Web 服务客户端日志文件的名称。每次使用 -zoc 服务器选项或 WebClientLogFile 属性设置或重置文件名时，Web 服务客户端日志文件都将被截断。路径中的任何反斜线字符都必须双写，因为这是字符串。请参见“-zoc 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

- **WebClientLogging** 此选项可启用或禁用对 Web 服务客户端的记录。记录的信息包括 HTTP 请求和响应数据。指定 ON 将开始向 Web 服务客户端日志文件写入记录，而指定 OFF 可停止向该文件写入记录。请参见“-zoc 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》。

权限

以下选项与应用程序分析或请求记录有关，需要具有 DBA 或 PROFILE 权限：

- ProcedureProfiling
- ProfileFilterConn
- ProfileFilterUser
- RequestFilterConn
- RequestFilterDB
- RequestLogFile
- RequestLogging
- RequestLogMaxSize
- RequestLogNumFiles

所有其它选项都要求具有 DBA 权限。

副作用

无

示例

以下语句禁止与数据库服务器建立新连接：

```
CALL sa_server_option( 'ConnsDisabled', 'YES' );
```

以下语句禁止与当前数据库建立新连接：

```
CALL sa_server_option( 'ConnsDisabledForDB', 'YES' );
```

以下语句启用所有 SQL 语句、过程调用、计划、阻塞和取消阻塞事件的记录，并指定要启动一个新请求日志：

```
CALL dbo.sa_server_option( 'RequestLogging', 'SQL+PROCEDURES+BLOCKS+PLAN
+REPLACE' );
```

sa_set_http_header 系统过程

允许一个 Web 服务设置结果的 HTTP 标头。

语法

sa_set_http_header(
fldname,
val
)

参数

- **fldname** 使用此 CHAR(128) 参数指定一个字符串，该字符串包含其中一个 HTTP 标头字段的名称。
- **val** 使用此 LONG VARCHAR 参数指定应为指定参数设置的值。

注释

设置特殊标头字段 @HttpStatus 将会设置请求所返回的状态码。状态码也称为响应码。例如，以下命令将状态码设置为 [404 未找到]。

```
CALL dbo.sa_set_http_header( '@HttpStatus', '404' );
```

错误消息主体将会被自动插入。只能使用有效的 HTTP 错误码。将状态设置为无效代码时会导致 SQL 错误。

可以删除由数据库服务器自动生成的响应标头，但 Connection 响应标头例外。例如，下面的命令删除 Expires 响应标头：

```
CALL dbo.sa_set_http_header( 'Expires', NULL );
```

权限

无

副作用

无

另请参见

- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “HTTP_HEADER 函数 [HTTP]”一节第 214 页
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_http_header_info 系统过程”一节第 837 页
- “sa_http_variable_info 系统过程”一节第 840 页
- “sa_set_http_option 系统过程”一节第 895 页
- “sa_split_list 系统过程”一节第 900 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

示例

以下示例将 Content-Type 标头字段设置为 text/html。

```
CALL dbo.sa_set_http_header( 'Content-Type', 'text/html' );
```

sa_set_http_option 系统过程

允许一个 Web 服务在结果中设置一个 HTTP 选项。

语法

sa_set_http_option(
optname,
val
)

参数

- **optname** 使用此 CHAR(128) 参数指定一个字符串，该字符串包含其中一个 HTTP 选项的名称。
- **val** 使用此 LONG VARCHAR 参数指定应为指定选项设置的值。

注释

在处理 Web 服务的语句或过程中使用此过程在 HTTP 结果集中设置选项。

所支持的选项为：

- **CharsetConversion** 使用此选项控制是否自动将结果集从数据库的字符集转换为客户端的字符集。其值只允许为 ON 和 OFF。缺省值为 ON。请参见“使用字符集自动转换”一节 《SQL Anywhere 服务器 - 编程》。
- **AcceptCharset** 使用此选项指定用于 XML、SOAP 和 HTTP Web 服务的响应字符集。此选项的语法符合 RFC2616 超文本传输协议 (Hypertext Transfer Protocol) 中用于 HTTP Accept-Charset 请求标头字段说明的语法。

要指定可接受的字符集列表，HTTP 客户端可以在发送给数据库服务器的请求中包含 Accept-Charset 请求标头。Accept-Charset 请求标头指定数据库服务器在响应时应使用哪个字符集。发送响应时，会使用客户端和数据库服务器均支持的字符集。

未指定 Accept-Charset 请求标头时，则使用客户端最常用且受数据库服务器支持的字符集。如果客户端指定的所有字符集数据库服务器均不支持，或者未指定 Accept-Charset 请求标头，则使用数据库字符集。

如果客户端未发送 Accept-Charset 请求标头，则数据库服务器使用 AcceptCharset 选项指定的第一个字符集。

如果客户端指定了 Accept-Charset 请求标头，而 AcceptCharset HTTP 选项也指定了一个字符集列表，则使用客户端最常用且受数据库服务器支持的字符集。如果客户端指定的所有字符集数据库服务器均不支持，则使用 AcceptCharset HTTP 选项指定的第一个字符集。

AcceptCharset HTTP 选项中的加号 (+) 指示数据库服务器使用数据库字符集。例如，`( 'AcceptCharset', '+,UTF-8,*' )`。只要有可能，就会使用 dbcharset，即便客户端倾向于使用客户端和数据库服务器均通用的字符集。在响应时使用 dbcharset 可以不必进行转换，进而改善了数据库服务器的响应时间。

AcceptCharset HTTP 选项中的星号 (*) 指示数据库服务器使用客户端 Accept-Charset 列表中的字符集。如果不存在通用字符集，则使用 AcceptCharset 选项指定的第一个字符集。

- **SessionId** 使用此选项为 HTTP 会话提供一个名称。例如，`sa_set_http_option( 'SessionId', 'my_app_session_1' )` 将 HTTP 会话的 ID 设置为 my_app_session_1。SessionId 必须是非 NULL 字符串。有关 HTTP 会话的详细信息，请参见“使用 HTTP 会话”一节 《SQL Anywhere 服务器 - 编程》。
- **SessionTimeout** 使用此选项指定非活动期间 HTTP 会话持续的时间（单位为分钟）。当使用该会话的某个 HTTP 请求发出响应时，会重置 SessionTimeout 选项。

权限

无

副作用

无

另请参见

- “使用 HTTP 会话”一节 《SQL Anywhere 服务器 - 编程》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “sa_http_header_info 系统过程”一节第 837 页
- “sa_http_variable_info 系统过程”一节第 840 页
- “sa_set_http_header 系统过程”一节第 894 页
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

示例

以下示例指定如果可以接受 dbcharset，则使用它。如果无法接受 dbcharset，则指定 UTF-8 作为替代选项。如果 UTF-8 无法使用，则星号 (*) 指定客户端最常用且受数据库服务器支持的字符集：

```
sa_set_http_option( 'AcceptCharset', '+,UTF-8,*');
```

以下 pseudo 代码阐释了如何使用 AcceptCharset 选项：

```
if -- client sends an Accept-Charset header then
      if -- no AcceptCharset HTTP option set then
            if -- at least one of the client charsets are supported then
                  use most preferred client charset that is also supported by
the server
            else
                  use dbcharset
            end if;
      else -- an AcceptCharset HTTP option was set
               if -- at least one of the client charsets is also in the HTTP
option charset list
                     and is supported by the server then use most preferred
client charset that
                    is also in the HTTP option charset list and is supported
by the server(1)
               else
                     use the first charset listed in the AcceptCharset HTTP
option
            end if;
     end if
else -- client did not send an Accept-Charset header
     if -- no AcceptCharset HTTP option set then
           use dbcharset
        else -- an AcceptCharset HTTP option was set
                 use the first charset listed in the AcceptCharset HTTP option
        end if
end if;
```

以下示例将 HTTP 会话的超时时间设置为 5 分钟：

```
CALL sa_set_http_option('SessionTimeout', '5');
```

sa_set_soap_header 系统过程

允许设置 SOAP 响应的 SOAP 标头。此过程被用在从 SOAP Web 服务调用的存储过程内。

语法

sa_set_soap_header(
fldname,
val
)

参数

- **fldname** 使用此 VARCHAR 参数指定标头键，这是用来引用给定标头条目的唯一字符串（它不必与 *val* 的 localname 相同）。

- **val** 使用此 VARCHAR 参数指定在 SOAP 标头元素范围内顶级标头条目及其子项的原始 XML。

注释

当发送 SOAP 响应消息时，使用此过程设置的所有 SOAP 标头条目都在 SOAP 标头元素内进行序列化。内容为 NULL 的 *val* 不进行序列化。如果 SOAP 响应不存在标头条目，则不在 SOAP 封装内创建封闭的标头元素。

权限

无

副作用

无

另请参见

- “处理 SOAP 标头”一节 《SQL Anywhere 服务器 - 编程》
- “SQL Anywhere Web 服务” 《SQL Anywhere 服务器 - 编程》
- “-xs 服务器选项”一节 《SQL Anywhere 服务器 - 数据库管理》
- “Web 服务函数”一节第 124 页
- “Web 服务系统过程”一节第 786 页

示例

以下示例将 SOAP 标头 welcome 设置为 Hello：

```
sa_set_soap_header( 'welcome', '<welcome>Hello</welcome>' )
```

sa_set_tracing_level 系统过程

初始化要存储在诊断跟踪表中的跟踪信息的级别。

语法

sa_set_tracing_level(
level
[, *specified_scope*
, *specified_name*]
[, *do_commit*]
)

参数

- **level** 使用此 INTEGER 参数指定要执行的诊断跟踪的级别。可接受的值包括：
  - **0** 不生成任何跟踪数据。此级别使跟踪会话保持打开状态，但不将任何跟踪数据发送到诊断跟踪表。
  - **1** 设置跟踪的基本级别。
  - **2** 设置跟踪的中等级别。

- **3** 设置跟踪的高等级别。

- **specified_scope** 使用此可选的 LONG VARCHAR 参数指定跟踪范围；例如，USER、DATABASE、CONNECTION_NAME、TRIGGER 等等。
- **specified_name** 使用此可选的 LONG VARCHAR 参数为 *specified_scope* 中所指明的对象指定标识符。
- **do_commit** 使用此可选的 TINYINT 参数指定是否自动提交由此过程插入的行。指定 1（缺省值）自动提交行（建议使用），指定 0 不自动提交它们。

注释

此过程替换 sa_diagnostic_tracing_level 表中的行，将跟踪级别和范围更改为调用该过程时所指定的设置。

将级别设置为 0 不会停止跟踪会话。相反，跟踪会话仍附加到跟踪数据库上，但不发送跟踪数据。当级别为 0 时，跟踪会话仍处于活动状态。

必须从所分析的数据库调用此系统过程。

权限

需要 DBA 权限

副作用

无

另请参见

- “选择诊断跟踪级别”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “诊断跟踪范围”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “sa_diagnostic_tracing_level 表”一节第 780 页
- “使用诊断跟踪进行高级应用程序分析”一节 《SQL Anywhere 服务器 - SQL 的用法》

示例

以下示例将跟踪级别设置为 1。这意味着将针对性能计数器数据以及所执行语句的一些示例来分析整个数据库：

```
CALL sa_set_tracing_level( 1 );
```

以下示例将跟踪级别设置为 3，并指定用户 AG84756。这意味着将只跟踪与 AG84756 关联的活动：

```
CALL sa_set_tracing_level( 3, 'user', 'AG84756' );
```

sa_snapshots 系统过程

返回当前活动的快照的列表。

语法

sa_snapshots( )

结果集

列名	数据类型	说明
connection_num	INT	在其上正运行快照的连接的连接 ID。
start_sequence_num	UNSIGNED BIGINT	标识快照的唯一编号。
statement_level	BIT	如果使用语句快照或只读语句快照来创建快照，则为 true。否则为 false。

注释

一个连接上可能存在多个语句快照。就运行在语句快照隔离级别下的嵌套或交错语句而言，每个语句在第一个读取或更新时都开始一个不同的语句快照。

通常，每个连接仅有一个事务快照（statement_level=0 的 sa_snapshots 中每个连接有一个条目）。但是，与游标关联的快照在游标的第一次读取后从不更改，并且使用 WITH HOLD 打开的游标在提交或回退操作后仍处于打开状态。如果游标有关联的快照，则该快照也会保留。因此，同一个 connection_num 可能存在多个事务快照：一个当前事务快照，一个或多个由于 WITH HOLD 游标而保留的旧的事务快照。

权限

需要 DBA 权限

副作用

无

另请参见

- “sa_transactions 系统过程”一节第 909 页
- “快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》

sa_split_list 系统过程

采用由分隔符分隔的值的字符串，并返回行集—每个值一行。

语法

sa_split_list(
str
[, *delim*]
[, *maxlen*]
)

参数

- **str** 使用此 LONG VARCHAR 参数指定包含要拆分的以 *delim* 分隔的值的字符串。

- **delim** 使用此可选的 CHAR(10) 参数指定 *str* 中用来分隔值的分隔符。该分隔符可以是任何字符组成的字符串，最多为 10 个字节。如果未指定 *delim*，则缺省情况下使用逗号。
- **maxlen** 使用此可选的 INTEGER 参数指定返回值的最大长度。例如，如果 *maxlen* 设置为 3，则结果集中的值将被截断为 3 个字符长度。如果指定 0（缺省值），值可以为任意长度。

结果集

列名	数据类型	说明
line_num	INTEGER	行的顺序编号。
row_value	LONG VARCHAR	来自字符串的值，如果需要将被截断为 *maxlen*。

注释

sa_split_list 过程接受其值为分隔列表形式的字符串，并返回每行有一个值的结果集。这与 LIST 函数 [Aggregate] 执行的操作相反。如果字符串为以下形式，则 row_value 将返回空字符串：

- 以 *delim* 开头
- 字符串中包含两个连续的 *delim* 实例
- 以 *delim* 结尾

输入字符串内的白空格有意义。如果分隔符是空格字符，输入字符串中的额外空格将导致结果集中出现额外的行。如果分隔符不是空格字符，则不会从结果集的值中裁剪输入字符串中的空格。

权限

无

副作用

无

另请参见

- “LIST 函数 [Aggregate]”一节第 228 页

示例

以下查询返回黑色产品的列表。

```
SELECT list( Name )
  FROM Products
  WHERE Color = 'Black';
```

list (Products.Name)
T 恤衫、棒球帽、太阳帽、短裤

在以下示例中，使用 sa_split_list 过程返回来自集合列表的原始结果集。

```
SELECT *
  FROM sa_split_list( 'Tee Shirt,Baseball Cap,Visor,Shorts' );
```

line_num	row_value
1	Tee Shirt
2	Baseball Cap
3	Visor
4	Shorts

以下示例为每一词返回一行。为避免返回其 row_value 为空字符串的行，必须指定 WHERE 子句。

```
SELECT *
  FROM sa_split_list( 'one||three|four||six|', '|' )
  WHERE row_value <> '';
```

line_num	row_value
1	one
3	three
4	four
6	six

在以下示例中，将创建名为 ProductsWithColor 的过程。当调用 ProductsWithColor 过程时，它将使用 sa_split_list 来分析用户指定的颜色值，在 Products 表的 Color 列中查找，然后为与用户指定的颜色之一相匹配的每个产品返回名称、说明、大小和颜色。

以下过程调用的结果是所有白色或黑色产品的名称、说明、大小和颜色。

```
CREATE PROCEDURE ProductsWithColor( IN color_list LONG VARCHAR )
BEGIN
  SELECT Name, Description, Size, Color
  FROM Products
  WHERE Color IN ( SELECT row_value FROM sa_split_list( color_list ) )
END;
go

SELECT * from ProductsWithColor( 'white,black' );
```

sa_statement_text 系统过程

格式化 SELECT 语句，使各项显示在不同的行中。当从请求日志（所有的换行符都已被删除）中查看长语句时，此过程很有用。

语法

sa_statement_text(*txt* **)**

参数

- **txt** 使用此 LONG VARCHAR 参数指定 SELECT 语句。

注释

输入的 *txt* 必须是字符串（用单引号引起来）或字符串表达式。

权限

无

副作用

无

另请参见

- “sa_get_request_times 系统过程”一节第 833 页
- “sa_get_request_profile 系统过程”一节第 832 页

示例

以下调用格式化 SELECT 语句，使各项显示在不同的行中。

```
CALL sa_statement_text( 'SELECT * FROM car WHERE name=''Audi''' );
```

	stmt_text
1	SELECT *
2	FROM car
3	WHERE name = 'Audi'

sa_table_fragmentation 系统过程

报告数据库表的分段信息。

语法

sa_table_fragmentation(
[*tbl_name*
[, *owner_name*]]
)

参数

- **tbl_name** 使用此可选的 CHAR(128) 参数指定要检查其碎片的表的名称。
- **owner_name** 使用此可选的 CHAR(128) 参数指定 *tbl_name* 的所有者。

结果集

列名	数据类型	说明
TableName	CHAR(128)	表的名称。
rows	UNSIGNED INTEGER	表中的行数。
row_segments	UNSIGNED BIGINT	表中的行分段数。
segs_per_row	DOUBLE	每行的分段数。

注释

数据库管理员可以使用此过程获取数据库表的分段信息。如果不提供参数，则为数据库中的所有表返回结果。

当数据库表的碎片过多时，可以运行 REORGANIZE TABLE 或重建数据库以减少表碎片并提高性能。请参见“减少表碎片”一节 《SQL Anywhere 服务器 - SQL 的用法》 。

权限

需要 DBA 权限

副作用

无

另请参见

- “减少表碎片”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “重建数据库”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “REORGANIZE TABLE 语句”一节第 673 页

sa_table_page_usage 系统过程

报告有关数据库表的页面使用情况的信息。

语法

sa_table_page_usage( )

结果集

列名	数据类型	说明
TableId	UNSIGNED INTEGER	表 ID。
TablePages	INTEGER	表使用的表页的数目。
PctUsedT	INTEGER	已使用表页空间的百分比。

列名	数据类型	说明
IndexPages	INTEGER	表使用的索引页的数目。
PctUsedI	INTEGER	已使用索引页空间的百分比。
PctOfFile	INTEGER	表占用的全部数据库文件的百分比。
TableName	CHAR(128)	表名。

注释

结果包括由信息实用程序提供的同样信息。

权限

需要 DBA 权限

副作用

无

另请参见

- “信息实用程序 (dbinfo)”一节 《SQL Anywhere 服务器 - 数据库管理》

sa_table_stats 系统过程

报告有关每个表中已读取的页数的信息。

语法

sa_table_stats( )

结果集

列名	数据类型	说明
table_id	INT	表 ID。
creator	CHAR(128)	表的创建者的用户名。
table_name	CHAR(128)	表名。
count	UNSIGNED BIGINT	表中的预计行数，取自 SYSTAB。
table_page_count	UNSIGNED BIGINT	表使用的主页数。
table_page_cached	UNSIGNED BIGINT	当前存储在高速缓存中的表页数。

列名	数据类型	说明
table_page_reads	UNSIGNED BIGINT	为主表中页执行读取的页数。
ext_page_count	UNSIGNED BIGINT	表中的预计页数
ext_page_cached	UNSIGNED BIGINT	留作将来使用。
ext_page_reads	UNSIGNED BIGINT	留作将来使用。

注释

sa_table_stats 过程所返回的每个行都描述优化程序正为其维护页统计信息的表。sa_table_stats 过程可用来查找哪些表正在使用高速缓存以及为每个表执行的磁盘读取数。例如，可使用 sa_table_stats 过程查找进行的磁盘读取数最多的表。过程的结果表示预计结果，应仅用于诊断目的。

table_page_cached 列指明当前存储在高速缓存中的表页数，table_page_reads 列指明自从优化程序开始维护表的计数后已从磁盘读取的表页数。这些统计信息不是永久存储在数据库中；第一次将其装载到内存中后，它们表示表的活动状态。

权限

需要 DBA 权限

副作用

无

另请参见

- “SYSTAB 系统视图”一节第 973 页

sa_text_index_stats 系统过程

返回有关数据库中文本索引的统计信息。

语法

sa_text_index_stats( )

注释

sa_text_index_stats 系统过程用于查看数据库中各个文本索引的统计信息。下表说明了由 sa_text_index_stats 返回的信息。

列名	Type	说明
owner_id	UNSIGNED INT	表所有者的 ID
table_id	UNSIGNED INT	表 ID

列名	Type	说明
index_id	UNSIGNED INT	文本索引的 ID
text_config_id	UNSIGNED BIGINT	索引引用的文本配置对象的 ID
owner_name	CHAR(128)	所有者的名称
table_name	CHAR(128)	表的名称
index_name	CHAR(128)	文本索引的名称
text_config_name	CHAR(128)	文本配置对象的名称
doc_count	UNSIGNED BIGINT	文本索引中的索引列值总数
doc_length	UNSIGNED BIGINT	文本索引中数据的总长度
pending_length	UNSIGNED BIGINT	待执行更改的总长度
deleted_length	UNSIGNED BIGINT	待执行删除的总长度
last_refresh	TIMESTAMP	上次刷新的日期和时间

对于 IMMEDIATE REFRESH 文本索引，pending_length、deleted_length 和 last_refresh 的值为 NULL。

对于 MANUAL REFRESH 文本索引，可以使用 doc_length、pending_length 和 deleted_length 确定是否刷新文本索引，以及要执行的刷新类型（重建或增量更新）。请参见“REFRESH TEXT INDEX 语句”一节第 668 页。

权限

需要 DBA 权限

副作用

None

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “文本索引”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “DROP TEXT INDEX 语句”一节第 560 页
- “REFRESH TEXT INDEX 语句”一节第 668 页
- “TRUNCATE TEXT INDEX 语句”一节第 728 页
- “sa_refresh_text_indexes 系统过程”一节第 877 页
- “sa_text_index_vocab 系统过程”一节第 908 页
- “SYSTEXTIDX 系统视图”一节第 980 页

示例

以下语句返回数据库中各个文本索引的统计信息：

```
CALL sa_text_index_stats( );
```

sa_text_index_vocab 系统过程

列出所有在文本索引中出现的术语以及每个术语在其中出现的索引值的总数。

语法

sa_text_index_vocab(
'*text-index-name***',**
'*table-name***',**
'*table-owner***'**
)

参数

- **text-index-name** 此 CHAR(128) 参数用于指定文本索引的名称。
- **table-name** 此 CHAR(128) 参数用于指定在其上构建文本索引的表的名称。
- **table-owner** 此 CHAR(128) 参数用于指定表的所有者。

结果集

列名	数据类型	说明
term	VARCHAR(60)	文本索引中的术语。
freq	BIGINT	其中出现术语的索引值的数量。

注释

sa_text_index_vocab 系统过程返回所有在文本索引中出现的术语以及每个术语在其中出现的索引值的总数（如果术语在某些索引值中出现多次，它将少于出现的总次数）。

sa_text_index_vocab 系统过程具有以下限制：

- 不能与 CALL 语句一起使用。
- 不能用在过程内的语句中。
- 参数值不能是主机变量或表达式。参数 *text-index-name*、*table-name* 和 *table-owner* 必须是约束或变量。

权限

需要 DBA 权限，或对索引表的 SELECT 权限。

副作用

None

另请参见

- “全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “术语和短语搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》
- “DROP TEXT INDEX 语句”一节第 560 页
- “REFRESH TEXT INDEX 语句”一节第 668 页
- “TRUNCATE TEXT INDEX 语句”一节第 728 页
- “sa_refresh_text_indexes 系统过程”一节第 877 页
- “SYSTEXTIDX 系统视图”一节第 980 页

示例

以下示例在示例数据库的 Products.Description 列上构建了一个名为 VocabTxtIdx 的文本索引。以下语句执行 sa_text_index_vocab 系统过程，返回出现在文本索引中的所有术语。

```
CREATE TEXT INDEX VocabTxtIdx2 ON Products( Description );
SELECT *
   FROM sa_text_index_vocab( 'VocabTxtIdx2', 'Products', 'GROUPO' );
```

term	freq
Cap	2
Cloth	1
Cotton	2
Crew	1
Hooded	1
neck	2
...	...

sa_transactions 系统过程

返回当前活动的事务的列表。

语法

sa_transactions( )

结果集

列名	数据类型	说明
connection_num	INT	在其上正运行事务的连接的连接 ID。

列名	数据类型	说明
transaction _id	INT	只要数据库服务器对其进行跟踪就唯一标识事务的 ID。旧事务信息放弃后，即可重新使用 ID。
start_time	TIMESTAMP	何时启动事务的 TIMESTAMP。
start_sequence_num	UNSIGNED BIGINT	事务的起始序列号。
end_sequence_num	UNSIGNED BIGINT	如果事务已提交或回退，则为其结束序列号，否则为 NULL。
committed	bit	事务的状态：如果事务以 COMMIT 结束则为 true，如果以 ROLLBACK 结束则为 false，如果事务仍在活动则为 NULL。
version_entries	unsigned INT	事务已保存的行版本的计数。

注释

此过程提供有关当前针对数据库运行的事务的信息。

权限

需要 DBA 权限

副作用

无

另请参见

- “sa_snapshots 系统过程”一节第 899 页
- “快照隔离”一节 《SQL Anywhere 服务器 - SQL 的用法》

sa_unload_cost_model 系统过程

将当前开销模型卸载到指定文件。

语法

sa_unload_cost_model (*file_name* **)**

参数

- **file_name** 使用此 CHAR(256) 参数指定要在其中卸载数据的文件的名称。由于是数据库服务器执行该系统过程，因此 *file_name* 将指定数据库服务器计算机上的文件，而相对 *file_name* 将指定相对于数据库服务器启动目录的文件。

注释

优化程序使用开销模型来确定查询的最佳访问计划。数据库服务器维护每个数据库的开销模型。可随时使用 ALTER DATABASE 语句中的 CALIBRATE SERVER 子句来重新校准数据库的开销模型。例如，如果将数据库移动到非标准硬件上，您可能决定要重新校准开销模型。

sa_unload_cost_model 系统过程允许将开销模型保存到 ASCII 文件 (*file_name*)。然后，您可登录到另一个数据库，并使用 sa_load_cost_model 系统过程将第一个数据库中的开销模型装载到第二个数据库。这样就不必重新校准第二个数据库。

> **注意**
> sa_unload_cost_model 系统过程不会在该文件中包含 CALIBRATE PARALLEL READ 信息。

在存在大量相似的硬件安装的情况下，使用 sa_unload_cost_model 系统过程可以避免重复且耗时的重新校准活动。

权限

需要 DBA 权限

您必须对创建文件的位置具有写权限。

副作用

无

另请参见

- “ALTER DATABASE 语句”一节第 344 页
- “sa_load_cost_model 系统过程”一节第 845 页
- “查询优化与执行” 《SQL Anywhere 服务器 - SQL 的用法》

示例

以下示例将开销模型卸载到名为 costmodel8 的文件：

```
CALL sa_unload_cost_model( 'costmodel8' );
```

sa_external_library_unload 系统过程

卸载外部库。

语法

sa_external_library_unload(['*external-library*'])

参数

- **external-library** 此可选的 LONG VARCHAR 参数用于指定要卸载的库的名称。如果未指定任何库，则将卸载所有未使用的外部库。

注释

如果指定了正在使用的或未装载的外部库，则会返回错误。如果未指定参数，则在找不到任何已装载的外部库时会返回错误。

权限

需要 DBA 权限

副作用

None

另请参见

- “从过程调用外部库”一节 《SQL Anywhere 服务器 - 编程》

示例

以下示例卸载名为 myextlib.dll 的外部库：

```
CALL sa_external_library_unload( 'myextlib.dll' );
```

以下示例卸载所有当前未使用的库：

```
CALL sa_external_library_unload();
```

sa_validate 系统过程

校验全部或部分数据库。

语法

sa_validate(
 [[*owner_name*.]*tbl_name* | *owner_name*]
)

参数

- **tbl_name** 此可选的 VARCHAR(128) 参数用于指定要校验的表或实例化视图的名称。
- **owner_name** 此可选的 CHAR(128) 参数用于指定所有者。单独指定时，将校验所有者拥有的所有表和实例化视图。

权限

需要 DBA 权限

副作用

无

注释

如果指定了没有参数的 sa_validate()，数据库服务器将校验所有的表、实例化视图、索引、校验和以及数据库文件。

如果 *owner* 和 *tbl_name* 都未指定，将校验数据库中所有的表和实例化视图。另外，也会对数据库自身进行校验，包括校验和校验，以及各个表和实例化视图中的行数是否与各相关索引中的行数相匹配的校验。

tbl_name 和 *owner_name* 的值是字符串并且必须用引号括起来。

此过程返回一列，其名称为 Messages。校验期间返回的错误出现在列中。如果校验成功且未出现错误，则该列包含 [未检测到错误]。

小心
应在没有任何连接对数据库进行更改时对表或整个数据库进行校验；否则，可能会报告错误，指出某种形式的数据库损坏，而实际上并没有任何损坏。

示例

以下语句对 DBA 拥有的表和实例化视图执行校验：

```
CALL sa_validate( owner_name = 'DBA' );
```

sa_verify_password 系统过程

校验当前用户的口令。

语法

sa_verify_password(*curr_pwsd* **)**

参数

- **curr_pwsd** 使用此 CHAR(128) 参数指定当前数据库用户的口令。

注释

此过程由 sp_password 使用。如果口令匹配，则该过程直接返回。如果不匹配，则返回错误 [用户 ID 或口令无效]。

权限

无

副作用

无

另请参见

- “Adaptive Server Enterprise 系统过程”一节第 789 页

sp_login_environment 系统过程

设置用户登录时的连接选项。

语法

sp_login_environment( )

注释

sp_login_environment 是由 login_procedure 数据库选项调用的缺省过程。

建议不要编辑此过程。相反，如果要改变登录环境，请将 login_procedure 选项设置为指向另一个过程。

以下是 sp_login_environment 过程的文本：

```
CREATE PROCEDURE dbo.sp_login_environment( )
BEGIN
  IF connection_property( 'CommProtocol' ) = 'TDS' THEN
    CALL dbo.sp_tsql_environment( )
  END IF
END;
```

权限

无

副作用

无

另请参见

- “login_procedure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

sp_remote_columns 系统过程

生成远程表中的列的列表以及它们的数据类型的描述。

要使用此系统过程，必须用 CREATE SERVER 语句对服务器进行定义。

语法

sp_remote_columns(
@server_name,
@table_name
[, *@table_owner*
[, *@table_qualifier*]]
)

参数

- **@server_name** 使用此 CHAR(128) 参数指定包含 CREATE SERVER 语句所指定的服务器名的字符串。
- **@table_name** 使用此 CHAR(128) 参数指定远程表的名称。
- **@table_owner** 使用此可选的 CHAR(128) 参数指定 *@table_name* 的所有者。
- **@table_qualifier** 使用此可选的 CHAR(128) 参数指定 *@table_name* 所在的数据库的名称。

结果集

列名	数据类型	说明
database	CHAR(128)	数据库名称。
owner	CHAR(128)	数据库所有者名称。
table-name	CHAR(128)	表名。
column-name	CHAR(128)	列的名称。
domain-id	SMALLINT	指明列的数据类型的 INTEGER。
width	SMALLINT	此字段的含义取决于数据类型。对于字符类型，width 代表字符数。
scale	SMALLINT	此字段的含义取决于数据类型。对于 NUMERIC 数据类型，scale 是小数点后的位数。
nullable	SMALLINT	如果允许列值为空，则此字段为 1。否则 nullable 为 0。

注释

如果您输入 CREATE EXISTING 语句并且指定列列表，则对于获取远程表上可用列的列表可能会有帮助。sp_remote_columns 生成远程表上的列的列表以及它们的数据类型的描述。如果指定了数据库，则必须指定所有者或提供 NULL 值。

标准和兼容性

- **Sybase** 受 Open Client/Open Server 支持。

权限

无

副作用

无

另请参见

- “访问远程数据” 《SQL Anywhere 服务器 - SQL 的用法》
- “用于远程数据访问的服务器类” 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE SERVER 语句”一节第 481 页

示例

以下示例返回 SYSOBJECTS 表中的列，该表位于名为 asetest 的 Adaptive Server Enterprise 服务器上的 production 数据库中。未指定所有者。

```
CALL sp_remote_columns( 'asetest', 'sysobjects', null, 'production' );
```

sp_remote_exported_keys 系统过程

提供其外键属于指定主表的表信息。

要使用此系统过程，必须用 CREATE SERVER 语句对服务器进行定义。

语法

```
sp_remote_exported_keys(
  @server_name
  , @sp_name
  [, @sp_owner
  [, @sp_qualifier ] ]
)
```

参数

- **@server_name** 使用此 CHAR(128) 参数指定主表所在的服务器。此参数需要值。
- **@sp_name** 使用此 CHAR(128) 参数指定包含主键的表。此参数需要值。
- **@sp_owner** 使用此可选的 CHAR(128) 参数指定主表的所有者。
- **@sp_qualifier** 使用此可选的 CHAR(128) 参数指定包含主表的数据库。

结果集

列名	数据类型	说明
pk_database	CHAR(128)	包含主键表的数据库。
pk_owner	CHAR(128)	主键表的所有者。
pk_table	CHAR(128)	主键表。
pk_column	CHAR(128)	主键列的名称。
fk_database	CHAR(128)	包含外键表的数据库。
fk_owner	CHAR(128)	外键表的所有者。
fk_table	CHAR(128)	外键表。
fk_column	CHAR(128)	外键列的名称。
key_seq	SMALLINT	键序列号。
fk_name	CHAR(128)	外键名称。
pk_name	CHAR(128)	主键名称。

注释

此过程提供外键属于特定主表的远程表的信息。sp_remote_exported_keys 系统过程的结果集包含数据库、所有者、表、列、主键和外键的名称以及外键列的外键序列。由于基础 ODBC 和 JDBC 调用的缘故，结果集可能不同，但总是返回外键的表信息和列信息。

权限

无

副作用

无

另请参见

- “CREATE SERVER 语句”一节第 481 页
- “外键”一节 《SQL Anywhere 11 - 简介》

示例

获取有关引用外健的远程表的信息，这些外键位于名为 asetest 的服务器上的 production 数据库中的 SYSOBJECTS 表中：

```
CALL sp_remote_exported_keys(
     @server_name='asetest',
     @sp_name='sysobjects',
     @sp_qualifier='production' );
```

sp_remote_imported_keys 系统过程

提供有关远程表的信息，这些表包含对应于指定外键的主键。

要使用此系统过程，必须用 CREATE SERVER 语句对服务器进行定义。

语法

sp_remote_imported_keys(
@server_name
, *@sp_name*
[, *@sp_owner*
[, *@sp_qualifier*]]
)

参数

- **@server_name** 使用此可选的 CHAR(128) 参数指定外键表所在的服务器。此参数需要值。
- **@sp_name** 使用此可选的 CHAR(128) 参数指定包含外键的表。此参数需要值。
- **@sp_owner** 使用此可选的 CHAR(128) 参数指定外键表的所有者。
- **@sp_qualifier** 使用此可选的 CHAR(128) 参数指定包含外键表的数据库。

结果集

列名	数据类型	说明
pk_database	CHAR(128)	包含主键表的数据库。
pk_owner	CHAR(128)	主键表的所有者。
pk_table	CHAR(128)	主键表。
pk_column	CHAR(128)	主键列的名称。
fk_database	CHAR(128)	包含外键表的数据库。
fk_owner	CHAR(128)	外键表的所有者。
fk_table	CHAR(128)	外键表。
fk_column	CHAR(128)	外键列的名称。
key_seq	SMALLINT	键序列号。
fk_name	CHAR(128)	外键名称。
pk_name	CHAR(128)	主键名称。

注释

外键引用包含相应主键的单独表中的一行。使用此过程可以获取其主键对应于特定外表的远程表的列表。sp_remote_imported_keys 结果集包含数据库、所有者、表、列、主键和外键的名称以及外键列的外键序列。由于基础 ODBC 和 JDBC 调用的缘故，结果集可能不同，但总是返回主键的表信息和列信息。

权限

无

副作用

无

另请参见

- “CREATE SERVER 语句”一节第 481 页
- “外键”一节 《SQL Anywhere 11 - 简介》

示例

获取主键对应于 asetest 服务器上 SYSOBJECTS 表的外键的表：

```
CALL sp_remote_imported_keys(
     @server_name='asetest',
     @sp_name='sysobjects',
     @sp_qualifier='production' );
```


sp_remote_primary_keys 系统过程

使用远程数据访问提供远程表的主键信息。

语法

sp_remote_primary_keys(
@server_name
[, *@table_name*
[, *@table_owner*
[, *@table_qualifier*]]]
)

参数

- **@server_name** 使用此 CHAR(128) 参数指定远程表所在的服务器。
- **@table_name** 使用此可选的 CHAR(128) 参数指定远程表。
- **@table_owner** 使用此可选的 CHAR(128) 参数指定远程表的所有者。
- **@table_qualifier** 使用此可选的 CHAR(128) 参数指定远程数据库的名称。

结果集

列名	数据类型	说明
database	CHAR(128)	远程数据库的名称。
owner	CHAR(128)	远程表的所有者。
table-name	CHAR(128)	远程表。
column-name	CHAR(128)	列名称。
key-seq	SMALLINT	主键序列号。
pk-name	CHAR(128)	主键名称。

注释

此系统过程使用远程数据访问提供远程表的主键信息。

由于基础 ODBC/JDBC 调用的差异，返回的信息在目录/数据库值方面会稍有不同（这取决于为服务器所指定的远程数据访问类）。不过，重要的信息（如列名称）与预期一致。

标准和兼容性

- **Sybase** 受 Open Client/Open Server 支持。

权限

无

副作用

无

sp_remote_tables 系统过程

返回服务器上表的列表。

要使用此系统过程，必须用 CREATE SERVER 语句对服务器进行定义。

语法

```
sp_remote_tables(
 @server_name
 [, @table_name
 [, @table_owner
 [, @table_qualifier
 [, @with_table_type ] ] ] ]
)
```

参数

- **@server_name** 使用此 CHAR(128) 参数指定远程表所在的服务器。
- **@table_name** 使用此 CHAR(128) 参数指定远程表。
- **@table_owner** 使用此 CHAR(128) 参数指定远程表的所有者。
- **@table_qualifier** 使用此 CHAR(128) 参数指定 *table_name* 所在的数据库。
- **@with_table_type** 使用此可选的 BIT 参数指定远程表的类型。此参数是位类型，接受两个值：0（缺省值）和 1。如果希望结果集中包含列出表类型的列，则必须输入值 1。

结果集

列名	数据类型	说明
database	CHAR(128)	远程数据库的名称。
owner	CHAR(128)	远程数据库所有者的名称。
table-name	CHAR(128)	远程表。
table-type	CHAR(128)	指定表的类型。此字段的值取决于远程服务器的类型。例如，可能的值有 TABLE、VIEW、SYS 和 GBL TEMP。

注释

当您配置数据库服务器以获取特定服务器上可用的远程表的列表时，此过程可能会有帮助。此过程返回服务器上表的列表。

此过程接受五个参数。如果给定了表、所有者或数据库名，则表的列表仅包含与这些参数匹配的表。

标准和兼容性

- **Sybase** 受 Open Client/Open Server 支持。

权限

无

副作用

无

另请参见

- “访问远程数据” 《SQL Anywhere 服务器 - SQL 的用法》
- “用于远程数据访问的服务器类” 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE SERVER 语句”一节第 481 页

示例

获取名为 excel 的服务器引用的 ODBC 数据源中可用的所有 Microsoft Excel 工作表的列表：

```
CALL sp_remote_tables( 'excel' );
```

获取 production 数据库（在名为 asetest 的 Adaptive Server Enterprise 服务器中）中由 fred 拥有的所有表的列表：

```
CALL sp_remote_tables( 'asetest', null, 'fred', 'production' );
```

sp_servercaps 系统过程

显示远程服务器的容量信息。

要使用此系统过程，必须用 CREATE SERVER 语句对服务器进行定义。

语法

sp_servercaps(@*sname*)

参数

- **@sname** 使用此 CHAR(64) 参数指定使用 CREATE SERVER 语句定义的服务器。指定的 *@sname* 必须与 CREATE SERVER 语句中使用的服务器名相同。

注释

此过程显示有关远程服务器的容量的信息。SQL Anywhere 使用此容量信息来确定可以向远程服务器转发多少 SQL 语句。直到 SQL Anywhere 第一次连接到远程服务器后，才填充列出服务器功能的 ISYSCAPABILITY 系统表。

标准和兼容性

- **Sybase** 受 Open Client/Open Server 支持。

权限

无

副作用

无

另请参见

- “SYSCAPABILITY 系统视图”一节第 939 页
- “SYSCAPABILITYNAME 系统视图”一节第 940 页
- “访问远程数据” 《SQL Anywhere 服务器 - SQL 的用法》
- “用于远程数据访问的服务器类” 《SQL Anywhere 服务器 - SQL 的用法》
- “CREATE SERVER 语句”一节第 481 页

示例

显示有关远程服务器 testasa 的信息：

```
CALL sp_servercaps( 'testasa' );
```

sp_tsql_environment 系统过程

设置用户从 jConnect 或 Open Client 应用程序连接时的连接选项。

语法

sp_tsql_environment( )

注释

sp_login_environment 过程是由 login_procedure 数据库选项指定的缺省过程。对于每个新连接，均调用由 login_procedure 指定的过程。如果连接使用的是 TDS 通信协议（即，如果是 Open Client 或 jConnect 连接），则 sp_login_environment 接着调用 sp_tsql_environment。

此过程设置数据库选项，使它们与缺省 Sybase Adaptive Server Enterprise 行为兼容。

如果希望改变缺省行为，建议创建新过程并改变 login_procedure 选项，使其指向这些新过程。

权限

无

副作用

无

另请参见

- “sp_login_environment 系统过程”一节第 913 页
- “login_procedure 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》

示例

以下是 sp_tsql_environment 过程的文本：

```
CREATE PROCEDURE dbo.sp_tsql_environment()
BEGIN
    IF db_property( 'IQStore' ) = 'Off' THEN
        -- SQL Anywhere datastore
        SET TEMPORARY OPTION close_on_endtrans='OFF';
    END IF;
    SET TEMPORARY OPTION ansinull='OFF';
    SET TEMPORARY OPTION tsql_variables='ON';
    SET TEMPORARY OPTION ansi_blanks='ON';
    SET TEMPORARY OPTION chained='OFF';
    SET TEMPORARY OPTION quoted_identifier='OFF';
    SET TEMPORARY OPTION allow_nulls_by_default='OFF';
    SET TEMPORARY OPTION on_tsql_error='CONTINUE';
    SET TEMPORARY OPTION isolation_level='1';
    SET TEMPORARY OPTION date_format='YYYY-MM-DD';
    SET TEMPORARY OPTION timestamp_format='YYYY-MM-DD HH:NN:SS.SSS';
    SET TEMPORARY OPTION time_format='HH:NN:SS.SSS';
    SET TEMPORARY OPTION date_order='MDY';
    SET TEMPORARY OPTION escape_character='OFF';
END
```

xp_cmdshell 系统过程

从过程中执行操作系统命令。

语法

xp_cmdshell(
command
[, **'no_output'**])

参数

- **command** 使用此 CHAR(8000) 参数指定一个系统命令。
- **'no_output'** 使用此可选的 CHAR(254) 参数指定是否显示输出。缺省行为是显示输出。如果此参数为字符串 'no_output'，则不显示输出。

注释

xp_cmdshell 执行系统命令，然后将控制返回给调用环境。由 xp_cmdshell 返回的值是所执行的 shell 进程的退出代码。如果在开始子进程时发生错误，返回值为 2。

第二个参数只影响 Windows 操作系统上的命令行应用程序。对于 Unix，无论如何设置第二个参数，都不显示任何输出。

但对于 Windows Mobile，无论如何设置第二个参数，执行的任何命令都会在数据库服务器消息日志中显示。运行该过程时需要控制台 shell *\\windows\cmd.exe*。

权限

需要 DBA 权限

另请参见

- “CALL 语句”一节第 400 页

示例

以下语句列出文件 *c:\temp.txt* 中当前目录下的文件：

```
xp_cmdshell( 'dir > c:\\temp.txt' )
```

以下语句执行相同的操作，但在执行时不显示 [**命令**] 窗口。

```
xp_cmdshell( 'dir > c:\\temp.txt', 'no_output' )
```

xp_msver 系统过程

检索有关数据库服务器的版本和名称信息。

语法

xp_msver(*string* **)**

- ***string*** 此字符串必须是以下字符串之一，并且用字符串分隔符括起来。

参数	说明
ProductName	产品名 (SQL Anywhere)。
ProductVersion	版本号，其后是内部版本号。格式如下： `11.0.0.3512`
CompanyName	返回以下字符串： `iAnywhere Solutions, Inc.`
FileDescription	返回产品名，其后是操作系统的名称。
LegalCopyright	返回软件的版权字符串。
LegalTrademarks	返回软件的商标信息。

注释

xp_msver 返回产品、公司、版本和其它信息。

权限

None

另请参见

- “系统函数”一节第 127 页

示例

以下语句请求版本和操作系统说明：

```
SELECT xp_msver( 'ProductVersion') Version,
   xp_msver( 'FileDescription' ) Description;
```

示例输出如下。具体系统上的版本值可能会有所不同。

版本	说明
11.0.0.3512	SQL Anywhere Windows XP

xp_read_file 系统过程

读取文件并以 LONG BINARY 变量的形式返回文件的内容。

语法

xp_read_file(*filename* **)**

参数

- **filename** 使用此 LONG VARCHAR 参数指定为其返回内容的文件的名称。

注释

此函数读取指定文件的内容，然后以 LONG BINARY 值的形式返回结果。

filename 是相对于数据库服务器的起始目录的。

对于将文件中存储的整个文档或图像插入到表中，此函数非常有用。如果无法读取文件，此函数则返回 NULL。

如果数据文件是以不同的字符集进行编码的，可以使用 CSCONVERT 函数来进行转换。请参见“CSCONVERT 函数 [String]”一节第 161 页。

CSCONVERT 函数还能够用来解决使用 xp_read_file 系统过程时遇到的字符集转换问题。请参见“CSCONVERT 函数 [String]”一节第 161 页。

权限

需要 DBA 权限

另请参见

- “CSCONVERT 函数 [String]”一节第 161 页
- “xp_write_file 系统过程”一节第 934 页
- “CALL 语句”一节第 400 页
- “搭配使用 openxml 和 xp_read_file”一节 《SQL Anywhere 服务器 - SQL 的用法》

示例

以下语句将一个图像插入到表 t1 的名为 picture 的列中（假设其它所有列都接受 NULL）：

```
INSERT INTO t1 ( picture )
   SELECT xp_read_file( 'portrait.gif' );
```

xp_scanf 系统过程

从输入字符串和格式字符串中抽取子串。

语法

xp_scanf(
input_buffer,
format,
parm [, *parm2*, ...]
)

参数

- **input_buffer** 使用此 CHAR(254) 参数指定输入字符串。
- **format** 使用此 CHAR(254) 参数指定输入字符串的格式，为每个 *parm* 参数使用占位符 (%s)。*format* 参数中最多可以有五十个占位符，而且必须与 *parm* 参数中的占位符数量相同。
- **parm** 使用这些 CHAR(254) 参数中的一个或多个来指定从 *input_buffer* 中抽取的子字符串。最多可以有 50 个这样的参数。

注释

xp_scanf 系统过程使用指定的 *format* 从输入字符串中抽取子字符串，并将结果放在指定的 *parm* 值中。

权限

无

另请参见

- “CALL 语句”一节第 400 页

示例

以下语句从输入缓冲区 Hello World! 中抽取子字符串 Hello 和 World!，并将它们分别放在变量 string1 和 string2 中，然后再选择它们：

```
CREATE VARIABLE string1 CHAR(254);
CREATE VARIABLE string2 CHAR(254);
CALL xp_scanf( 'Hello World!', '%s %s', string1, string2 );
SELECT string1, string2;
```

xp_sendmail 系统过程

发送电子邮件消息。

语法

```
xp_sendmail(
 recipient = mail-address
 [, subject = subject ]
 [, cc_recipient = mail-address ]
 [, bcc_recipient = mail-address ]
 [, query = sql-query ]
 [, "message" = message-body ]
 [, attachname = attach-name ]
 [, attach_result = attach-result ]
 [, echo_error = echo-error ]
 [, include_file = filename ]
 [, no_column_header = no-column-header ]
 [, no_output = no-output ]
 [, width = width ]
 [, separator = separator-char ]
 [, dbuser = user-name ]
 [, dbname = db-name ]
 [, type = type ]
 [, include_query = include-query ]
 [, content_type = content-type ]
)
```

参数

一些参数提供固定值且可用来确保 Transact-SQL 兼容，如下所述。

- **recipient** 此 LONG VARCHAR 参数指定接收者的邮件地址。当指定多个接收者时，必须用分号分隔每个邮件地址。
- **subject** 此 LONG VARCHAR 参数指定消息的主题字段。缺省值为 NULL。
- **cc_recipient** 此 LONG VARCHAR 参数指定 cc 接收者的邮件地址。当指定多个 cc 接收者时，必须用分号分隔每个邮件地址。缺省值为 NULL。
- **bcc_recipient** 此 LONG VARCHAR 参数指定 bcc 接收者的邮件地址。当指定多个 bcc 接收者时，必须用分号分隔每个邮件地址。缺省值为 NULL。
- **query** 此 LONG VARCHAR 参数与 Transact-SQL 一起使用。缺省值为 NULL。
- **"message"** 此 LONG VARCHAR 参数指定消息内容。缺省值为 NULL。"message" 参数名需要用双引号括起来，因为 MESSAGE 是保留字。请参见“保留字”一节第 4 页。
- **attachname** 此 LONG VARCHAR 参数与 Transact-SQL 一起使用。缺省值为 NULL。
- **attach_result** 此 INT 参数与 Transact-SQL 一起使用。缺省值为 0。
- **echo_error** 此 INT 参数与 Transact-SQL 一起使用。缺省值为 1。
- **include_file** 此 LONG VARCHAR 参数指定附件文件。缺省值为 NULL。
- **no_column_header** 此 INT 参数与 Transact-SQL 一起使用。缺省值为 0。
- **no_output** 此 INT 参数与 Transact-SQL 一起使用。缺省值为 0。
- **width** 此 INT 参数与 Transact-SQL 一起使用。缺省值为 80。
- **separator** 此 CHAR(1) 参数与 Transact-SQL 一起使用。缺省值为 CHAR(9)。

- **dbuser** 此 LONG VARCHAR 参数与 Transact-SQL 一起使用。缺省值为 guest。
- **dbname** 此 LONG VARCHAR 参数与 Transact-SQL 一起使用。缺省值为 master。
- **type** 此 LONG VARCHAR 参数与 Transact-SQL 一起使用。缺省值为 NULL。
- **include_query** 此 INT 参数与 Transact-SQL 一起使用。缺省值为 0。
- **content_type** 此 LONG VARCHAR 参数指定 "message" 参数的内容类型（例如，text/html、ASIS 等）。缺省值为 NULL。content_type 的值不经过校验；设置无效的内容类型导致发送无效或不可理解的电子邮件。

  如果要手动设置标头，则可将 content_type 参数设为 ASIS。执行此操作时，xp_sendmail 过程假定传递给消息参数的数据已经是带有标头的格式正确的电子邮件，且不添加任何附加标头。指定 ASIS 时，必须手动设置消息参数中的所有标头，也包括通常会通过将数据传递给其它参数而进行填充的标头。

权限

需要 DBA 权限

必须执行了 xp_startmail 才能使用 MAPI 启动电子邮件会话，或执行了 xp_startsmtp 才能使用 SMTP 启动电子邮件会话。

如果使用 MAPI 发送邮件，则不支持 content_type 参数。

注释

xp_sendmail 是一个系统过程，可在使用 xp_startmail 或 xp_startsmtp 启动了会话后，将电子邮件消息发送给指定的接收者。此过程接受任何长度的消息。xp_sendmail 的参数值为字符串。每个参数的长度受系统可用内存量的限制。

content_type 参数旨在用于那些了解 MIME 电子邮件要求的用户。xp_sendmail 接受 ASIS 作为 content_type。当 content_type 设置为 ASIS 时，xp_sendmail 假定消息主体 ("message") 是带有标头的格式正确的电子邮件，且不添加任何附加标头。指定 ASIS 以发送包含多个内容类型的多部分消息。有关 MIME 的详细信息，请参见 RFC 2045-2049 (http://www.ietf.org/)。

由 include_file 参数指定的附件将作为 application/octet-stream MIME 类型进行发送（采用 base64 编码），这些附件必须存在于数据库服务器上。

在 SQL Anywhere 10.0.0 及更高版本中，如果主题行含有非 7 位 ASCII 码的字符，系统会对 SMTP 电子邮件系统发送的邮件进行编码。此外，如果主题行含有非 7 位 ASCII 码的字符，电子邮件发送到具有 SMS 功能的设备时，可能无法正确解码。

返回代码

请参见“MAPI 和 SMTP 系统过程的返回代码”一节第 787 页。

另请参见

- “MAPI 和 SMTP 过程”一节第 786 页
- “xp_startmail 系统过程”一节第 931 页
- “xp_startsmtp 系统过程”一节第 932 页
- “xp_stopmail 系统过程”一节第 933 页
- “xp_stopsmtp 系统过程”一节第 934 页
- “CALL 语句”一节第 400 页

示例

以下调用向用户 ID 为 Sales Group 的用户发送一条包含文件 *prices.doc* （作为邮件附件）的消息：

```
CALL xp_sendmail( recipient='Sales Group',
      subject='New Pricing',
      include_file = 'C:\\DOCS\\PRICES.DOC' );
```

以下示例程序显示 xp_sendmail 系统过程的不同用法，如示例本身所述：

```
BEGIN
DECLARE to_list LONG VARCHAR;
DECLARE email_subject CHAR(256);
DECLARE content LONG VARCHAR;
DECLARE uid CHAR(20);

SET to_list='test_account@mytestdomain.com';
SET email_subject='This is a test';
SET uid='test_sender@mytestdomain.com';

// Call xp_startsmtp to start an SMTP email session
CALL xp_startsmtp( uid, 'mymailserver.mytestdomain.com' );

// Basic email example
SET content='This text is the body of my email.\n';
CALL xp_sendmail( recipient=to_list,
     subject=email_subject,
     "message"=content );

// Send email containing HTML using the content_type parameter,
// as well as including an attachment with the include_file
// parameter
SET content='Plain text.<BR><BR><B>Bold text.</B><BR><BR><a
href="www.iAnywhere.com">iAnywhere
 Home Page</a></B><BR><BR>';
CALL xp_sendmail( recipient=to_list,
     subject=email_subject,
     "message"=content,
     content_type = 'text/html',
     include_file = 'test.zip' );

// Send email "ASIS".  Here the content-type has been specified
// by the user as part of email body.  Note the attachment can
// also be done separately
SET content='Content-Type: text/html;\nContent-Disposition: inline; \n\nThis
text
 is not bold<BR><BR><B>This text is bold</B><BR><BR><a
href="www.iAnywhere.com">iAnywhere Home
 Page</a></B><BR><BR>';
CALL xp_sendmail( recipient=to_list,
     subject=email_subject,
     "message"=content,
```

```
        content_type = 'ASIS',
        include_file = 'test.zip' );

    // Send email "ASIS" along with an include file.  Note that
    // "message" contains the information for another attachment
    SET content = 'Content-Type: multipart/mixed; boundary="xxxxx";\n';
    SET content = content || 'This part of the email should not be shown.  If this
    is shown
     then the email client is not MIME compatible\n\n';
    SET content = content || '--xxxxx\n';
    SET content = content || 'Content-Type: text/html;\n';
    SET content = content || 'Content-Disposition: inline;\n\n';
    SET content = content || 'This text is not bold<BR><BR><B>This text is bold</
    B><BR>
     <BR><a href="www.iAnywhere.com">iAnywhere Home Page</a></B><BR><BR>\n\n';
    SET content = content || '--xxxxx\n';
    SET content = content || 'Content-Type: application/zip; name="test.zip"\n';
    SET content = content || 'Content-Transfer-Encoding: base64\n';
    SET content = content || 'Content-Disposition: attachment;
    filename="test.zip"\n\n';

    // Encode the attachment yourself instead of adding this one in
    // the include_file parameter
    SET content = content || base64_encode( xp_read_file( 'othertest.zip' ) ) ||
    '\n\n';
    SET content = content || '--xxxxx--\n';
    CALL xp_sendmail( recipient=to_list,
          subject=email_subject,
          "message"=content,
          content_type = 'ASIS',
          include_file = 'othertest.zip' );

    // End the SMTP session
        CALL xp_stopsmtp();
    END
```

xp_sprintf 系统过程

用一组输入字符串建立结果字符串。

语法

xp_sprintf(
output_buffer,
format,
parm [**,** *parm2*, ...]
)

参数

- **output_buffer** 使用此 CHAR(254) 参数指定包含结果字符串的输出缓冲区。
- **format** 使用此 CHAR(254) 参数指定结果字符串的格式，为每个 *parm* 参数使用占位符 (%s)。*format* 参数中最多可以有五十个占位符，而且应与 *parm* 参数中的占位符数量相同。
- **parm** 这些是结果字符串中使用的输入字符串。最多可以指定 50 个这样的 CHAR(254) 参数。

注释

xp_sprintf 系统过程使用 *format* 参数和 *parm* 参数创建字符串，并将结果放在 *output-buffer* 中。

权限

无

另请参见

- “CALL 语句”一节第 400 页

示例

以下语句将字符串 Hello World! 放在结果变量中。

```
CREATE VARIABLE result CHAR(254);
Call xp_sprintf( result, '%s %s', 'Hello', 'World!' );
```

xp_startmail 系统过程

在 MAPI 下启动电子邮件会话。

语法

xp_startmail(
[mail_user = *mail-login-name*]
[, mail_password = *mail-password*] **)**

参数

- **mail_user** 使用此 LONG VARCHAR 参数指定 MAPI 登录名。
- **mail_password** 使用此 LONG VARCHAR 参数指定 MAPI 口令。

权限

需要 DBA 权限

在 Unix 上不受支持。

注释

xp_startmail 是一个启动电子邮件会话的系统过程。

如果您使用的是 Microsoft Exchange，则 *mail-login-name* 参数是 Exchange 配置文件名，在过程调用中不应该包含口令。

返回代码

请参见“MAPI 和 SMTP 系统过程的返回代码”一节第 787 页。

另请参见

- “MAPI 和 SMTP 过程”一节第 786 页
- “xp_stopmail 系统过程”一节第 933 页
- “xp_sendmail 系统过程”一节第 926 页
- “xp_startsmtp 系统过程”一节第 932 页
- “xp_stopsmtp 系统过程”一节第 934 页
- “CALL 语句”一节第 400 页

xp_startsmtp 系统过程

在 SMTP 下启动电子邮件会话。

语法

xp_startsmtp(
smtp_sender = *email-address*
, smtp_server = *smtp-server*
[, smtp_port = *port-number* **]**
[, timeout = *timeout* **]**
[, smtp_sender_name = *username* **]**
[, smtp_auth_username = *auth-username*
[, smtp_auth_password = *auth-password*
)

参数

- **smtp_sender** 此 LONG VARCHAR 参数指定发送者的电子邮件地址。
- **smtp_server** 此 LONG VARCHAR 参数指定要使用的 SMTP 服务器，可以是服务器名或 IP 地址。
- **smtp_port** 此可选 INTEGER 参数指定要连接的 SMTP 服务器上的端口号。缺省值为 25。
- **timeout** 此可选 INTEGER 参数指定中止对 xp_sendmail 的当前调用前等待数据库服务器响应的时间（以秒为单位）。缺省值是 60 秒。
- **smtp_sender_name** 此可选 LONG VARCHAR 参数指定发送者的电子邮件地址的别名。例如，'JSmith' 代替 '*email-address*'。
- **smtp_auth_username** 此可选 LONG VARCHAR 参数指定要提供到需要验证的 SMTP 服务器的口令。
- **smtp_auth_password** 此可选 LONG VARCHAR 参数指定要提供到需要验证的 SMTP 服务器的口令。

权限

需要 DBA 权限

注释

xp_startsmtp 是一个系统过程，它通过连接到 SMTP 服务器来启动指定电子邮件地址的邮件会话。此连接可能会超时。因此，建议在执行 xp_sendmail 前才调用 xp_start_smtp。

病毒扫描程序会影响 xp_startsmtp，导致其返回错误代码 100。对于 McAfee VirusScan 版本 8.0.0 和更高版本，防止电子邮件蠕虫大量散播邮件的设置也会妨碍 xp_sendmail 的正确执行。如果您的病毒扫描软件允许您指定可绕过大量邮件散播保护的进程，请指定 *dbeng11.exe* 和 *dbsrv11.exe*。例如，以 McAfee VirusScan 为例，您可以将这两个进程添加到防止大量邮件散播的 [Properties] 区域的 [Excluded Processes] 列表。

返回代码

请参见“MAPI 和 SMTP 系统过程的返回代码”一节第 787 页。

另请参见

- “MAPI 和 SMTP 过程”一节第 786 页
- “xp_startmail 系统过程”一节第 931 页
- “xp_stopmail 系统过程”一节第 933 页
- “xp_sendmail 系统过程”一节第 926 页
- “xp_stopsmtp 系统过程”一节第 934 页
- “CALL 语句”一节第 400 页

xp_stopmail 系统过程

关闭 MAPI 电子邮件会话。

语法

xp_stopmail( )

权限

需要 DBA 权限

在 Unix 上不受支持。

注释

xp_stopmail 是一个结束电子邮件会话的系统过程。

返回代码

请参见“MAPI 和 SMTP 系统过程的返回代码”一节第 787 页。

另请参见

- “MAPI 和 SMTP 过程”一节第 786 页
- “xp_startmail 系统过程”一节第 931 页
- “xp_sendmail 系统过程”一节第 926 页
- “xp_startsmtp 系统过程”一节第 932 页
- “xp_stopsmtp 系统过程”一节第 934 页
- “CALL 语句”一节第 400 页

xp_stopsmtp 系统过程

关闭 SMTP 电子邮件会话。

语法

xp_stopsmtp( )

权限

需要 DBA 权限

注释

xp_stopsmtp 是一个结束电子邮件会话的系统过程。

返回代码

请参见“MAPI 和 SMTP 系统过程的返回代码”一节第 787 页。

另请参见

- “MAPI 和 SMTP 过程”一节第 786 页
- “xp_startmail 系统过程”一节第 931 页
- “xp_stopmail 系统过程”一节第 933 页
- “xp_sendmail 系统过程”一节第 926 页
- “xp_startsmtp 系统过程”一节第 932 页
- “CALL 语句”一节第 400 页

xp_write_file 系统过程

从 SQL 语句将数据写入文件。

语法

xp_write_file(
filename,
file_contents
)

参数

- **filename** 使用此 LONG VARCHAR 参数指定文件名。
- **file_contents** 使用此 LONG BINARY 参数指定要写入到文件中的内容。

注释

此函数将 *file_contents* 写入 *filename* 文件。如果成功则返回 0，如果失败则返回非零值。

filename 值前面可以带有绝对或相对路径。如果 *filename* 前面带有相对路径，则文件名是相对于数据库服务器的当前工作目录的。如果文件已存在，其内容将被覆盖。

此函数可用于将长的二进制数据卸载到文件中。

CSCONVERT 函数还能够用来解决使用 xp_write_file 系统过程时遇到的字符集转换问题。请参见“CSCONVERT 函数 [String]”一节第 161 页。

权限

需要 DBA 权限

另请参见

- “CSCONVERT 函数 [String]”一节第 161 页
- “xp_read_file 系统过程”一节第 925 页
- “CALL 语句”一节第 400 页

示例

以下示例使用 xp_write_file 创建一个包含数据 123456 的文件 *accountnum.txt*：

```
CALL xp_write_file( 'accountnum.txt', '123456' );
```

以下示例查询示例数据库的 Contacts 表，然后为居住在新泽西的每个联系人创建一个文本文件。每个文本文件都使用联系人的名字 (GivenName)、姓氏 (Surname) 和字符串 .txt 的连接来命名（例如，*Reeves_Scott.txt*），并在单独行上包含联系人的街道地址 (Street)、城市 (City) 和州 (State)。

```
SELECT xp_write_file(
Surname || '_' ||  GivenName || '.txt',
Street || '\n' || City || '\n' || State )
FROM Contacts WHERE State = 'NJ';
```

以下示例使用 xp_write_file 为 Products 表中的每个产品创建一个图像文件 (JPG)。ID 列的每个值成为具有 Photo 列对应值内容的文件的文件名：

```
SELECT xp_write_file( ID || '.jpg' , Photo ) FROM Products;
```

在上面示例中，ID 是具有 UNIQUE 约束的行。这对于确保文件不被后续行的内容覆盖非常重要。此外，必须指定适用于列中所存储的数据的文件扩展名。在本例中，Products.Photo 存储图像数据 (JPG)。

视图

目录

系统视图

目录包含通过键和索引链接在一起的系统表。在 SQL Anywhere 中，系统表是隐藏的。但每个表都有一个系统视图。在某些情况下，一个系统视图可能还包括多个系统表中的列，以满足通常需要的连接。

为确保与以后版本的 SQL Anywhere 目录兼容，应确保您的应用程序使用系统视图而不是可能会更改的基础系统表。

♦ 查看详细的系统信息视图和定义 (Sybase Central)

1. 以具有 DBA 权限的用户身份连接到数据库。
2. 右键单击数据库并选择 [**配置所有者过滤**]。
3. 单击 [**SYS**]，然后单击 [**确定**]。
4. 在左窗格中，双击 [**视图**]。
5. 在左窗格中单击 [视图]，然后在右窗格中单击 [**SQL**] 选项卡。

   单击 [**数据**] 选项卡查看有关已选视图的详细信息。

SYSARTICLE 系统视图

SYSARTICLE 系统视图的每一行都描述了发布中的一个项目。该视图的基础系统表为 ISYSARTICLE。

列名	列类型	说明
publication_id	UNSIGNED INT	该项目所属的发布。
table_id	UNSIGNED INT	每个项目都由来自单个表的列和行组成。该列包含该表的表 ID。
where_expr	LONG VARCHAR	对于包含 WHERE 子句定义的行的子集的项目，该列包含搜索条件。
subscribe_by_expr	LONG VARCHAR	对于包含 SUBSCRIBE BY 表达式定义的行的子集的项目，该列包含表达式。
query	CHAR(1)	为数据库服务器指示项目类型信息。
alias	VARCHAR(256)	项目的别名。

基础系统表上的约束

PRIMARY KEY (publication_id, table_id)

FOREIGN KEY (publication_id) 引用 SYS.ISYSPUBLICATION (publication_id)

FOREIGN KEY (table_id) 引用 SYS.ISYSTAB (table_id)

SYSARTICLECOL 系统视图

SYSARTICLECOL 系统视图的每一行都标识项目中的一列。该视图的基础系统表为 ISYSARTICLECOL。

列名	列类型	说明
publication_id	UNSIGNED INT	该列所属发布的唯一标识符。
table_id	UNSIGNED INT	该列所属的表。
column_id	UNSIGNED INT	列标识符，来自 SYSTABCOL 系统视图。

基础系统表上的约束

PRIMARY KEY (publication_id, table_id, column_id)

FOREIGN KEY (publication_id, table_id) 引用 SYS.ISYSARTICLE (publication_id, table_id)

FOREIGN KEY (table_id, column_id) 引用 SYS.ISYSTABCOL (table_id, column_id)

SYSCAPABILITY 系统视图

SYSCAPABILITY 系统视图的每一行都标识远程服务器的一种功能。该视图的基础系统表为 ISYSCAPABILITY。

列名	列类型	说明
capid	INTEGER	功能的 ID，如 SYSCAPABILITYNAME 系统视图中所列。
srvid	UNSIGNED INT	在其中应用了该功能的服务器，如 SYSSERVER 系统视图中所列。
capvalue	CHAR(128)	功能的值。

基础系统表上的约束

PRIMARY KEY (capid, srvid)

FOREIGN KEY (srvid) 引用 SYS.ISYSSERVER (srvid)

另请参见

- “SYSCAPABILITYNAME 系统视图”一节第 940 页

SYSCAPABILITYNAME 系统视图

SYSCAPABILITYNAME 系统视图中的每一行都命名 SYSCAPABILITY 系统视图中定义的一种功能。

列名	列类型	说明
capid	INTEGER	唯一标识功能的编号。
capname	CHAR(128)	功能的名称。

注释

SYSCAPABILITYNAME 系统视图是使用 sa_rowgenerator 和以下服务器属性的组合进行定义：

- RemoteCapability
- MaxRemoteCapability

另请参见

- “数据库服务器属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SYSCAPABILITY 系统视图”一节第 939 页

SYSCHECK 系统视图

SYSCHECK 系统视图中的每一行为表中一个指定的检查约束提供定义。该视图的基础系统表为 ISYSCHECK。

列名	列类型	说明
check_id	UNSIGNED INT	唯一标识数据库中的约束的编号。
check_defn	LONG VARCHAR	CHECK 表达式。

基础系统表上的约束

PRIMARY KEY (check_id)

FOREIGN KEY (check_id) 引用 SYS.ISYSCONSTRAINT (constraint_id)

SYSCOLPERM 系统视图

GRANT 语句可以将 UPDATE、SELECT 或 REFERENCES 权限赋予表中的单个列。SYSCOLPERM 系统视图的一行中记录了每个具有 UPDATE、SELECT 或 REFERENCES 权限的列。该视图的基础系统表为 ISYSCOLPERM。

列名	列类型	说明
table_id	UNSIGNED INT	包含该列的表的表号。
grantee	UNSIGNED INT	用户 ID 的用户号，该用户 ID 被赋予该列的权限。如果 grantee 是特殊 PUBLIC 用户 ID 的用户号，则将该权限赋予所有用户 ID。
grantor	UNSIGNED INT	授予权限的用户 ID 的用户号。
column_id	UNSIGNED INT	此列编号与 table_id 一同标识被授予权限的列。
privilege_type	SMALLINT	此列中的编号指示列权限的种类（16=REFERENCES、1=SELECT 或 8=UPDATE）。
is_grantable	CHAR(1)	表示是否将 WITH GRANT OPTION 授予了该列权限。

基础系统表上的约束

PRIMARY KEY (table_id, grantee, grantor, column_id, privilege_type)

FOREIGN KEY (table_id, column_id) 引用 SYS.ISYSTABCOL (table_id, column_id)

FOREIGN KEY (grantor) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY (grantee) 引用 SYS.ISYSUSER (user_id)

SYSCOLSTAT 系统视图

SYSCOLSTAT 系统视图包含由优化程序使用的列统计数据，其中包括直方图。使用 sa_get_histogram 存储过程或直方图实用程序可以最有效地检索该视图的内容。该视图的基础系统表为 ISYSCOLSTAT。

列名	列类型	说明
table_id	UNSIGNED INT	唯一标识该列所属的表或实例化视图的编号。
column_id	UNSIGNED INT	与 table_id 一同唯一标识该列的编号。
format_id	SMALLINT	仅供系统使用。
update_time	TIMESTAMP	上次更新列统计数据的时间。
density	FLOAT	估计的列的单值平均选择性，不计算存储在行中的大的单值选择性。

列名	列类型	说明
max_steps	SMALLINT	仅供系统使用。
actual_steps	SMALLINT	仅供系统使用。
step_values	LONG BINARY	仅供系统使用。
frequencies	LONG BINARY	仅供系统使用。

注意
对于使用 SQL Anywhere 11.0.0 版及更高版本创建的数据库，会始终加密此视图的基础系统表，以防止他人对数据进行未经授权的访问。

基础系统表上的约束

PRIMARY KEY (table_id, column_id)

FOREIGN KEY (table_id, column_id) 引用 SYS.ISYSTABCOL (table_id, column_id)

SYSCONSTRAINT 系统视图

SYSCONSTRAINT 系统视图的每一行描述数据库中的一个命名约束。该视图的基础系统表为 ISYSCONSTRAINT。

列名	列类型	说明
constraint_id	UNSIGNED INT	约束的唯一 ID。
constraint_type	CHAR(1)	约束类型： ● C - 列检查约束。 ● T - 表约束。 ● P - 主键。 ● F - 外键。 ● U - 唯一约束。
ref_object_id	UNSIGNED BIGINT	要应用约束的列、表或索引的对象 ID。
table_object_id	UNSIGNED BIGINT	要应用约束的表的表 ID。
constraint_name	CHAR(128)	约束名称。

基础系统表上的约束

PRIMARY KEY (constraint_id)

FOREIGN KEY (ref_object_id) 引用 SYS.ISYSOBJECT (object_id)

FOREIGN KEY (table_object_id) 引用 SYS.ISYSOBJECT (object_id)

SYSDBFILE 系统视图

SYSDBFILE 系统视图中的每一行都描述一个 dbspace 文件。该视图的基础系统表为 ISYSDBFILE。

列名	列类型	说明
dbfile_id	SMALLINT	仅供内部使用。
dbspace_id	SMALLINT	数据库中的每个 dbspace 文件都有一个指定的唯一编号。system dbspace 包含全部系统对象，且其 dbspace_id 为 0。
dbfile_name	CHAR(128)	dbspace 的文件名。对于 system 和 TEMPORARY 之外的 dbspace，可以用以下语句更改文件名： `ALTER DBSPACE dbspace RENAME 'new-filename';`
file_name	LONG VARCHAR	dbspace 的唯一名称。该名称用在 CREATE TABLE 命令中。
lob_map	LONG VARBIT	仅供内部使用。

基础系统表上的约束

PRIMARY KEY (dbfile_id)

FOREIGN KEY (dbspace_id) 引用 SYS.ISYSDBSPACE (dbspace_id)

SYSDBSPACE 系统视图

SYSDBSPACE 系统视图中的每一行都描述了一个 dbspace 文件。该视图的基础系统表为 ISYSDBSPACE。

列名	列类型	说明
dbspace_id	SMALLINT	标识 dbspace 的唯一编号。system dbspace 包含全部系统对象，且其 dbspace_id 为 0。

列名	列类型	说明
object_id	UNSIGNED BIGINT	dbspace 的文件名。对于 SYSTEM dbspace，该值是创建数据库时的数据库文件的名称，它仅供参考且不能更改。对于其它 dbspace，可以用以下语句更改文件名： `ALTER DBSPACE dbspace RENAME 'new-filename';`
dbspace_name	CHAR(128)	dbspace 的唯一名称。该名称用在 CREATE TABLE 命令中。
store_type	TINYINT	仅供内部使用。

基础系统表上的约束

PRIMARY KEY (dbspace_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT MATCH UNIQUE FULL

SYSDBSPACEPERM 系统视图

SYSDBSPACEPERM 系统视图中的每一行都描述了 dbspace 文件的权限。该视图的基础系统表为 ISYSDBSPACEPERM。

列名	列类型	说明
dbspace_id	SMALLINT	标识 dbspace 的唯一编号。system dbspace 包含全部系统对象，且其 dbspace_id 为 0。
grantee	UNSIGNED INT	获得权限的用户的用户 ID。
privilege_type	SMALLINT	授予被授予者的权限。例如，CREATE 授予被授予者在 dbspace 上创建对象的权限。

基础系统表上的约束

FOREIGN KEY (dbspace_ID) 引用 SYS.ISYSDBSPACE

FOREIGN KEY (grantee) 引用 SYS.ISYSUSER (user_id)

另请参见

- “GRANT 语句”一节第 595 页
- “数据库权限和特权概述”一节 《SQL Anywhere 服务器 - 数据库管理》

SYSDEPENDENCY 系统视图

SYSDEPENDENCY 系统视图中的各行用来描述两个数据库对象间的依赖性。该视图的基础系统表为 ISYSDEPENDENCY。

当一个数据库对象引用其定义中的另一个对象时，这两个数据库对象之间就存在依赖性。例如，如果视图的查询说明引用一个表，则该视图就被认为是依赖于该表。数据库服务器将跟踪视图对表、视图、实例化视图和列的依赖性。

列名	列类型	说明
ref_object_id	UNSIGNED BIGINT	被引用对象的对象 ID。
dep_object_id	UNSIGNED BIGINT	引用对象的 ID。

基础系统表上的约束

PRIMARY KEY (ref_object_id, dep_object_id)

FOREIGN KEY (ref_object_id) 引用 SYS.ISYSOBJECT (object_id)

FOREIGN KEY (dep_object_id) 引用 SYS.ISYSOBJECT (object_id)

另请参见

- “sa_dependent_views 系统过程”一节第 817 页
- “视图依赖性”一节 《SQL Anywhere 服务器 - SQL 的用法》

SYSDOMAIN 系统视图

SYSDOMAIN 系统视图记录有关内置数据类型（也称为域）的信息。在正常操作期间，该视图的内容不发生更改。该视图的基础系统表为 ISYSDOMAIN。

列名	列类型	说明
domain_id	SMALLINT	指派给每种数据类型的唯一编号。无法更改这些编号。
domain_name	CHAR(128)	通常在 CREATE TABLE 命令中使用的数据类型（如 CHAR 或 INTEGER）的名称。
type_id	SMALLINT	ODBC 数据类型。该值与 Transact-SQL 兼容性 dbo.SYSTYPES 表中的 data_type 的值相对应。
"precision"	SMALLINT	使用该数据类型可以存储的有效位数。对于非数字数据类型，列值为 NULL。

基础系统表上的约束

PRIMARY KEY (domain_id)

SYSEVENT 系统视图

SYSEVENT 系统视图中的每一行都描述一个使用 CREATE EVENT 创建的事件。该视图的基础系统表为 ISYSEVENT。

列名	列类型	说明
event_id	UNSIGNED INT	指派给每个事件的唯一编号。
object_id	UNSIGNED BIGINT	事件的内部 ID，在数据库中唯一标识事件。
creator	UNSIGNED INT	事件所有者的用户号。通过查看 SYSUSER 系统视图中可以找到的用户的名称。
event_name	VARCHAR(128)	事件的名称。
enabled	CHAR(1)	指出是否允许触发事件。
location	CHAR(1)	将要触发事件的位置： ● C = 统一 ● R = 远程 ● A = 全部
event_type_id	UNSIGNED INT	对于系统事件，指 SYSEVENTTYPE 中所列的事件类型。
action	LONG VARCHAR	事件处理程序定义。模糊值指示隐藏事件。
external_action	LONG VARCHAR	仅供系统使用。
condition	LONG VARCHAR	用于控制事件处理程序触发的条件。
remarks	LONG VARCHAR	事件的注释；该列来自 ISYSREMARK。
source	LONG VARCHAR	事件的原始来源；该列来自 ISYSSOURCE。

基础系统表上的约束

PRIMARY KEY (event_id)

FOREIGN KEY (creator) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

另请参见

● “SYSEVENTTYPE 系统视图”一节第 947 页

SYSEVENTTYPE 系统视图

SYSEVENTTYPE 系统视图定义可由 CREATE EVENT 引用的系统事件类型。

列名	列类型	说明
event_type_id	UNSIGNED INT	指派给每种事件类型的唯一编号。
name	VARCHAR(128)	系统事件类型的名称。
description	LONG VARCHAR	系统事件类型的说明。

注释

SYSEVENTTYPE 系统视图是使用 sa_rowgenerator 和以下服务器属性的组合进行定义：

- EventTypeName
- EventTypeDesc
- MaxEventType

另请参见

- “数据库服务器属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “SYSEVENT 系统视图”一节第 946 页

SYSEXTERNENV 系统视图

SQL Anywhere 支持六种外部运行时环境。它们包括以 C/C++ 编写的嵌入式 SQL 和 ODBC 应用程序，以及以 Java、Perl、PHP 或 C# 和 Visual Basic 等基于 Microsoft .NET Framework 公共语言运行库 (CLR) 的语言编写的应用程序。

SYSEXTERNENV 系统视图中的各行描述了标识和启动每个外部环境所需的信息。该视图的基础系统表为 ISYSEXTERNENV。

列名	列类型	说明
object_id	unsigned bigint	外部环境的唯一标识符。
name	varchar(128)	外部环境或语言的名称。
scope	char(1)	标识外部环境是以 one-per-connection (C) 还是 one-per-database (D) 启动。
support_result_sets	char(1)	标识可以将结果集返回给用户的外部环境。
location	long varchar	标识可以找到环境的主要可执行文件的位置。
options	long varchar	标识启动外部环境的命令行所需的选项。

列名	列类型	说明
user_id	unsigned int	标识数据库中具有 DBA 权限的用户。

基础系统表上的约束

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (user_id) 引用 SYS.ISYSUSER (user_id)

SYSEXTERNENVOBJECT 系统视图

SQL Anywhere 支持六种外部运行时环境。它们包括以 C/C++ 编写的嵌入式 SQL 和 ODBC 应用程序，以及以 Java、Perl、PHP 或 C# 和 Visual Basic 等基于 Microsoft .NET Framework 公共语言运行库 (CLR) 的语言编写的应用程序。

SYSEXTERNENVOBJECT 系统视图中的每一行都描述已安装的外部对象。该视图的基础系统表为 ISYSEXTERNENVOBJECT。

列名	列类型	说明
object_id	unsigned bigint	外部对象的唯一标识符。
extenv_id	unsigned bigint	外部环境的唯一标识符 (SYSEXTERNENV.object_id)。
owner	unsigned int	标识外部对象的创建者/所有者。
name	long varchar	标识在 INSTALL 语句中指定的外部对象的名称。
contents	long binary	外部对象的内容。
update_time	timestamp	标识上次修改（或安装）对象的时间。

基础系统表上的约束

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (extenv_id) 引用 SYS.ISYSEXTERNENV (object_id)

FOREIGN KEY (owner) 引用 SYS.ISYSUSER (user_id)

SYSEXTERNLOGIN 系统视图

SYSEXTERNLOGIN 系统视图的每行都描述一个用于远程数据访问的外部登录。该视图的基础系统表为 ISYSEXTERNLOGIN。

注意
以前版本的目录中包含一个 SYSEXTERNLOGINS 系统表。该表已重命名为 ISYSEXTERNLOGIN（无 'S'），并且是本视图的基础表。

列名	列类型	说明
user_id	UNSIGNED INT	本地数据库上的用户 ID。
srvid	UNSIGNED INT	SYSSERVER 系统视图中所列的远程服务器。
remote_login	VARCHAR(128)	用户登录远程服务器的登录名。
remote_password	VARBINARY(128)	用户登录远程服务器的口令。

注意
对于使用 SQL Anywhere 11.0.0 版及更高版本创建的数据库，会始终加密此视图的基础系统表，以防止他人对数据进行未经授权的访问。

基础系统表上的约束

PRIMARY KEY (user_id, srvid)

FOREIGN KEY (user_id) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY (srvid) 引用 SYS.ISYSSERVER (srvid)

SYSFKEY 系统视图

SYSFKEY 系统视图的每行描述系统中的一个外键约束。该视图的基础系统表为 ISYSFKEY。

列名	列类型	说明
foreign_table_id	UNSIGNED INT	外表的表号。
foreign_index_id	UNSIGNED INT	外键的索引号。
primary_table_id	UNSIGNED INT	主表的表号。
primary_index_id	UNSIGNED INT	主键的索引号。

列名	列类型	说明
match_type	TINYINT	约束的匹配类型。匹配类型包括： ● 0 - 使用缺省匹配 ● 1 - SIMPLE ● 2 - FULL ● 129 - SIMPLE UNIQUE ● 130 - FULL UNIQUE 有关匹配类型的详细信息，请参见“CREATE TABLE 语句”一节第 497 页的 MATCH 子句。
check_on_commit	CHAR(1)	指出 INSERT 和 UPDATE 语句是否应等到 COMMIT 时才检查外键是否仍然有效。
nulls	CHAR(1)	指出是否允许外键中的列包含 NULL 值。注意该设置与外键中所包含列的 nulls 设置无关。

基础系统表上的约束

PRIMARY KEY (foreign_table_id, foreign_index_id)

FOREIGN KEY (foreign_table_id, foreign_index_id) 引用 SYS.ISYSIDX (table_id, index_id)

FOREIGN KEY (primary_table_id, primary_index_id) 引用 SYS.ISYSIDX (table_id, index_id)

SYSGROUP 系统视图

对于每组的每一个成员，在 SYSGROUP 系统视图中都有与之对应的一行。该视图介绍了组与成员之间的多对多关系。一个组可以有许多成员，而一个用户也可以是许多组的成员。该视图的基础系统表为 ISYSGROUP。

列名	列类型	说明
group_id	UNSIGNED INT	组的用户号。
group_member	UNSIGNED INT	成员的用户号。

基础系统表上的约束

PRIMARY KEY (group_id, group_member)

FOREIGN KEY group_id (group_id) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY group_member (group_member) 引用 SYS.ISYSUSER (user_id)

SYSHISTORY 系统视图

SYSHISTORY 系统视图的每行记录对数据库的一个系统操作，例如数据库启动、数据库校准等。该视图的基础系统表为 ISYSHISTORY。

列名	列类型	说明
operation	CHAR(128)	对数据库文件执行的操作类型。操作必须是以下值之一： ● INIT - 有关何时创建数据库的信息。 ● UPGRADE - 有关何时升级数据库的信息。 ● START - 有关何时在特定操作系统下使用指定数据库服务器版本启动数据库的信息。 ● LAST_START - 有关最近一次启动数据库服务器的时间的信息。启动数据库时，如果所使用的数据库服务器版本和/或所在的操作系统与当前存储在 LAST_START 行中的值不同，LAST_START 操作将会被转换成 START 操作。 ● DTT - 有关*倒数第二次*对 dbspace 执行的*磁盘传送时间*（Disk Transfer Time，简称 DTT）校准操作的信息。即有关倒数第二次执行 ALTER DATABASE CALIBRATE 或 ALTER DATABASE RESTORE DEFAULT CALIBRATION 语句的信息。 ● LAST_DTT - 有关*最新*对 dbspace 执行的 DTT 校准操作的信息。即有关最新执行 ALTER DATABASE CALIBRATE 或 ALTER DATABASE RESTORE DEFAULT CALIBRATION 语句的信息。 ● LAST_BACKUP - 有关上次备份的信息，包括备份的日期和时间、备份类型、备份的文件及执行备份的数据库服务器的版本。
object_id	UNSIGNED INT	对于 DTT 和 LAST_DTT 以外的任何操作，该列的值都将为 0。对于 DTT 和 LAST_DTT 操作，这是在 SYSDBSPACE 系统视图中定义的 dbspace 的 dbspace_id。请参见“SYSDBSPACE 系统视图”一节第 943 页。
sub_operation	CHAR(128)	对于 DTT 和 LAST_DTT 以外的任何操作，该列的值将是一组空单引号 ('')。对于 DTT 和 LAST_DTT 操作，该列包含对 dbspace 执行的子操作的类型。值包括： ● DTT_SET - 已设置 dbspace 校准。 ● DTT_UNSET - 已将 dbspace 校准恢复为缺省设置。
version	CHAR(128)	用于执行此操作的数据库服务器版本与编译版本。

列名	列类型	说明
platform	CHAR(128)	执行该操作的操作系统。
first_time	TIMESTAMP	第一次在某一特定操作系统下以特定软件版本启动数据库的日期和时间。
last_time	TIMESTAMP	最近一次在某一特定操作系统下以特定软件版本启动数据库的日期和时间。
details	LONG VARCHAR	此列存储例如启动数据库服务器所使用的命令行选项或数据库启用的功能位等信息。该信息供技术支持使用。

基础系统表上的约束

PRIMARY KEY (operation, object_id, version, platform)

SYSIDX 系统视图

SYSIDX 系统视图的每行都定义数据库中的一个逻辑索引。该视图的基础系统表为 ISYSIDX。

列名	列类型	说明
table_id	UNSIGNED INT	唯一标识包含该索引的表。
index_id	UNSIGNED INT	标识表中索引的唯一编号。
object_id	UNSIGNED BIGINT	索引的内部 ID，在数据库中唯一标识索引。
phys_index_id	UNSIGNED INT	标识用于实现逻辑索引的基础物理索引。对于临时表或远程表上的索引，该值为 NULL。否则，该值与 SYSPHYSIDX 系统视窗中物理索引的 object_id 相对应。请参见“SYSPHYSIDX 系统视图”一节第 960 页。
dbspace_id	SMALLINT	包含索引的文件的 ID。该值对应 SYSDBSPACE 系统视图中的一个条目。请参见“SYSDBSPACE 系统视图”一节第 943 页。
file_id	SMALLINT	不建议使用。此列存在于 SYSVIEW 中，但不存在于基础系统表 ISYSIDX 中。此列的内容与 dbspace_id 相同，并且是为了兼容而提供的。请改用 dbspace_id。

列名	列类型	说明
index_category	TINYINT	索引的类型。值包括： ● 1 - 主键 ● 2 - 外键 ● 3 - 次级索引（包括唯一约束） ● 4 - 文本索引
"unique"	TINYINT	指出索引是唯一索引 (1)、非唯一索引 (4) 还是唯一约束 (2)。唯一索引可以避免索引表中的两行在索引列中具有相同的值。
index_name	CHAR(128)	索引的名称。
not_enforced	CHAR(1)	仅供系统使用。
file_id	SMALLINT	仅供系统使用。

基础系统表上的约束

PRIMARY KEY (table_id, index_id)

FOREIGN KEY (table_id) 引用 SYS.ISYSTAB (table_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (table_id, phys_index_id) 引用 SYS.ISYSPHYSIDX (table_id, phys_index_id)

另请参见

- “SYSIDXCOL 系统视图”一节第 953 页
- “SYSPHYSIDX 系统视图”一节第 960 页
- “SYSDBSPACE 系统视图”一节第 943 页

SYSIDXCOL 系统视图

SYSIDXCOL 系统视图的每行描述在 SYSIDX 系统视图中描述的索引的一列。该视图的基础系统表为 ISYSIDXCOL。

列名	列类型	说明
table_id	UNSIGNED INT	标识对其应用了该索引的表。
index_id	UNSIGNED INT	标识对其应用该列的索引。table_id 与 index_id 共同标识 SYSIDX 系统视图中描述的一个索引。

列名	列类型	说明
sequence	SMALLINT	索引中的每一列都有一个指定的从 0 开始的唯一编号。这些编号的顺序决定了列在索引中的相对重要性。最重要的列的序列号为 0。
column_id	UNSIGNED INT	标识表中的哪列被索引。table_id 与 column_id 共同标识 SYSCOLUMN 系统视图中描述的一列。
"order"	CHAR(1)	指出该列在索引中是升序 (A) 还是降序 (D)。对于文本索引此值为 NULL。
primary_column_id	UNSIGNED INT	与该外键列相对应的主键列的 ID。对于非外键列，值为 NULL。

基础系统表上的约束

PRIMARY KEY (table_id, index_id, column_id)

FOREIGN KEY (table_id, index_id) 引用 SYS.ISYSIDX (table_id, index_id)

FOREIGN KEY (table_id, column_id) 引用 SYS.ISYSTABCOL (table_id, column_id)

另请参见

- “SYSIDX 系统视图”一节第 952 页

SYSJAR 系统视图

SYSJAR 系统视图的每行定义在数据库中存储的一个 JAR 文件。该视图的基础系统表为 ISYSJAR。

列名	列类型	说明
jar_id	INTEGER	标识 JAR 文件的唯一编号。
object_id	UNSIGNED BIGINT	JAR 文件的内部 ID，在数据库中唯一标识该文件。
creator	UNSIGNED INT	JAR 文件创建者的用户号。
jar_name	LONG VARCHAR	JAR 文件的名称。
jar_file	LONG VARCHAR	数据库中 JAR 文件的外部文件名。
update_time	TIMESTAMP	上次更新 JAR 文件的时间。

基础系统表上的约束

PRIMARY KEY (jar_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

另请参见

- “SYSJARCOMPONENT 系统视图”一节第 955 页

SYSJARCOMPONENT 系统视图

SYSJAR 系统视图中的每一行都定义一个 JAR 文件组件。该视图的基础系统表为 ISYSJARCOMPONENT。

列名	列类型	说明
component_id	INTEGER	包含组件 ID 的主键。
jar_id	INTEGER	包含 JAR 的 ID 号的字段。
component_name	LONG VARCHAR	组件的名称。
component_type	CHAR(1)	组件的类型。
contents	LONG BINARY	JAR 文件的字节代码。

基础系统表上的约束

PRIMARY KEY (component_id)

FOREIGN KEY (jar_id) 引用 SYS.ISYSJAR (jar_id)

另请参见

- “SYSJAR 系统视图”一节第 954 页

SYSJAVACLASS 系统视图

SYSJAVACLASS 系统视图的每行描述在数据库中存储的一个 Java 类。该视图的基础系统表为 ISYSJAVACLASS。

列名	列类型	说明
class_id	INTEGER	Java 类的唯一编号。也是表的主键。
object_id	UNSIGNED BIGINT	JAR 类的内部 ID，在数据库中唯一标识该类。
creator	UNSIGNED INT	类的创建者的用户号。
jar_id	INTEGER	产生类的 JAR 文件的 ID。

列名	列类型	说明
class_name	LONG VARCHAR	Java 类的名称。
public	CHAR(1)	指出该类是公共类 (Y) 还是专用类 (N)。
component_id	INTEGER	SYSJARCOMPONENT 系统视图中组件的 ID。
update_time	TIMESTAMP	类的上次更新时间。

基础系统表上的约束

PRIMARY KEY (class_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (creator) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY (component_id) 引用 SYS.ISYSJARCOMPONENT (component_id)

SYSLOGINMAP 系统视图

SYSLOGINMAP 系统视图为可使用集成登录或 Kerberos 登录连接到数据库的每个用户都包含一行。作为安全措施，只有具有 DBA 权限的用户才能查看该视图的内容。该视图的基础系统表为 ISYSLOGINMAP。

列名	列类型	说明
login_mode	TINYINT	登录的类型：1 表示集成登录，2 表示 Kerberos 登录。
login_id	VARCHAR(1024)	集成登录用户配置文件名或映射到 database_uid 的 Kerberos 主体。
object_id	UNSIGNED BIGINT	唯一标识符，用户 ID 与数据库用户 ID 间的每个映射各一个。
database_uid	UNSIGNED INT	登录 ID 所映射到的数据库用户 ID。

基础系统表上的约束

PRIMARY KEY (login_mode, login_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (database_uid) 引用 SYS.ISYSUSER (user_id)

SYSLOGINPOLICY 系统视图

该视图的基础系统表为 ISYSLOGINPOLICY。

列名	列类型	说明
login_policy_id	UNSIGNED BIGINT	登录策略的唯一标识符。
login_policy_name	CHAR(128)	登录策略的名称。

基础系统表上的约束

PRIMARY KEY (login_policy_id)

FOREIGN KEY (login_policy_id) 引用 SYS.ISYSOBJECT (object_id)

UNIQUE (login_policy_name)

另请参见

- “SYSLOGINPOLICYOPTION 系统视图”一节第 957 页
- “SYSUSER 系统视图”一节第 983 页

SYSLOGINPOLICYOPTION 系统视图

该视图的基础系统表为 ISYSLOGINPOLICYOPTION。

列名	列类型	说明
login_policy_id	UNSIGNED BIGINT	登录策略的唯一标识符。
login_option_name	CHAR(128)	登录策略的名称。
login_option_value	LONG VARCHAR	登录策略被创建时的值。

基础系统表上的约束

PRIMARY KEY (login_policy_id, login_option_name)

FOREIGN KEY (login_policy_id) 引用 SYS.ISYSLOGINPOLICY (login_policy_id)

另请参见

- “SYSLOGINPOLICY 系统视图”一节第 957 页
- “SYSUSER 系统视图”一节第 983 页

SYSMVOPTION 系统视图

SYSMVOPTION 系统视图的每行描述了实例化视图在其创建时一个选项值的设置。但该描述并不包含选项的选项名称。该视图的基础系统表为 ISYSMVOPTION。

列名	列类型	说明
view_object_id	UNSIGNED BIGINT	实例化视图的对象 ID。
option_id	UNSIGNED INT	标识数据库中的选项的唯一编号。要查看选项名称，请参见 SYSMVOPTIONNAME 系统视图。
option_value	LONG VARCHAR	创建实例化视图时的选项值。

基础系统表上的约束

PRIMARY KEY (view_object_id, option_id)

FOREIGN KEY (view_object_id) 引用 SYS.ISYSOBJECT (object_id)

FOREIGN KEY (option_id) 引用 SYS.ISYSMVOPTIONNAME (option_id)

另请参见

- “SYSMVOPTIONNAME 系统视图”一节第 958 页

SYSMVOPTIONNAME 系统视图

SYSMVOPTIONNAME 系统视图的每行包含 SYSMVOPTION 系统视图中定义的选项的名称。该视图的基础系统表为 ISYSMVOPTIONNAME。

列名	列类型	说明
option_id	UNSIGNED INT	在数据库中唯一标识选项的编号。
option_name	CHAR(128)	选项的名称。

基础系统表上的约束

PRIMARY KEY (option_id)

另请参见

- “SYSMVOPTION 系统视图”一节第 958 页

SYSOBJECT 系统视图

SYSOBJECT 系统视图中的每行都描述一个数据库对象。该视图的基础系统表为 ISYSOBJECT。

列名	列类型	说明
object_id	UNSIGNED BIGINT	对象的内部 ID，在数据库中唯一标识该对象。
status	TINYINT	对象的状态。值包括： ● 1（有效）- 该对象可供数据库服务器使用。此状态与 [已启用] 同义。即，如果启用一个对象，则其状态变为 [有效]。 ● 2（无效）- 在一项内部操作之后尝试重新编译对象失败，例如，在对其依赖的某个对象进行模式变更修改之后。只要语句中一引用对象，数据库服务器就继续尝试重新编译该对象。 ● 4（禁用）- 该对象已被用户显式禁用，例如使用 ALTER TABLE...DISABLE VIEW DEPENDENCIES 语句。
object_type	TINYINT	对象的类型。值包括： ● 1 - 表 ● 2 - 视图 ● 3 - 实例化视图 ● 4 - 列 ● 5 - 索引 ● 6 - 过程 ● 7 - 触发器 ● 8 - 事件 ● 9 - 用户 ● 10 - 发布 ● 11 - 远程类型 ● 12 - 登录映射 ● 13 - JAR ● 14 - Java 类 ● 16 - 服务 ● 17 - 文本配置 ● 18 - Dbspace
creation_time	TIMESTAMP	对象创建的日期和时间。

基础系统表上的约束

PRIMARY KEY (object_id)

SYSOPTION 系统视图

SYSOPTION 系统视图包含一些选项，每个存储在数据库中的选项设置对应一行。对于给定的选项，每个用户都可以有自己的设置。此外，PUBLIC 用户 ID 的设置定义了缺省设置，以用于没有自己的设置的用户。该视图的基础系统表为 ISYSOPTION。

列名	列类型	说明
user_id	UNSIGNED INT	对其应用该选项设置的用户号。
"option"	CHAR(128)	选项的名称。
"setting"	LONG VARCHAR	选项的当前设置。

基础系统表上的约束

PRIMARY KEY (user_id, "option")

FOREIGN KEY (user_id) 引用 SYS.ISYSUSER (user_id)

SYSOPTSTAT 系统视图

SYSOPTSTAT 系统视图存储开销模型校准信息（由 ALTER DATABASE CALIBRATE 语句计算）。该视图的内容仅供内部使用并可通过 sa_get_dtt 系统过程最有效地访问。该视图的基础系统表为 ISYSOPTSTAT。

列名	列类型	说明
stat_id	UNSIGNED INT	仅供系统使用。
group_id	UNSIGNED INT	仅供系统使用。
format_id	SMALLINT	仅供系统使用。
data	LONG BINARY	仅供系统使用。

基础系统表上的约束

PRIMARY KEY (stat_id, group_id, format_id)

SYSPHYSIDX 系统视图

SYSPHYSIDX 系统视图的每行定义数据库中的一个物理索引。该视图的基础系统表为 SYSPHYSIDX。

列名	列类型	说明
table_id	UNSIGNED INT	索引对应的表的对象 ID。
phys_index_id	UNSIGNED INT	表中物理索引的唯一编号。
root	INTEGER	标识数据库文件中物理索引的根页的位置。
key_value_count	UNSIGNED INT	索引中不同键值的数量。
leaf_page_count	UNSIGNED INT	叶索引页的数量。
depth	UNSIGNED SMALLINT	物理索引的深度（层数）。
max_key_distance	UNSIGNED INT	仅供系统使用。
seq_transitions	UNSIGNED INT	仅供系统使用。
rand_transitions	UNSIGNED INT	仅供系统使用。
rand_distance	UNSIGNED INT	仅供系统使用。
allocation_bitmap	LONG VARBIT	仅供系统使用。
long_value_bitmap	LONG VARBIT	仅供系统使用。

基础系统表上的约束

PRIMARY KEY (table_id, phys_index_id)

另请参见

- “SYSIDXCOL 系统视图”一节第 953 页
- “SYSIDX 系统视图”一节第 952 页

SYSPROCEDURE 系统视图

SYSPROCEDURE 系统视图的每行描述数据库中的一个过程。该视图的基础系统表为 ISYSPROCEDURE。

列名	列类型	说明
proc_id	UNSIGNED INT	每个过程都有一个指定的唯一编号（过程号）。
creator	UNSIGNED INT	过程的所有者。

列名	列类型	说明
object_id	UNSIGNED BIGINT	过程的内部 ID，在数据库中唯一标识该过程。
proc_name	CHAR(128)	过程的名称。一个创建者不能有两个同名过程。
proc_defn	LONG VARCHAR	过程的定义。
remarks	LONG VARCHAR	有关过程的注释。该值存储在 ISYSREMARK 系统表中。
replicate	CHAR(1)	指出过程是否为复制服务器安装中的主数据源。
srvid	UNSIGNED INT	如果该过程是远程数据库服务器上过程的代理，则表示该远程服务器。
source	LONG VARCHAR	过程的保留源。该值存储在 ISYSSOURCE 系统表中。
avg_num_rows	FLOAT	当该过程出现在 FROM 子句中时，收集到的用于查询优化的信息。
avg_cost	FLOAT	当该过程出现在 FROM 子句中时，收集到的用于查询优化的信息。
stats	LONG BINARY	当该过程出现在 FROM 子句中时，收集到的用于查询优化的信息。

基础系统表上的约束

PRIMARY KEY (proc_id)

FOREIGN KEY (srvid) 引用 SYS.ISYSSERVER (srvid)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (creator) 引用 SYS.ISYSUSER (user_id)

SYSPROCPARM 系统视图

SYSPROCPARM 系统视图的每行描述数据库中过程的一个参数。该视图的基础系统表为 ISYSPROCPARM。

列名	列类型	说明
proc_id	UNSIGNED INT	唯一标识该参数所属的过程。
parm_id	SMALLINT	每个过程都从 1 开始对参数进行编号。参数编号的顺序对应于定义参数时采取的顺序。对于函数，第一个参数具有该函数的名称并代表该函数的返回值。
parm_type	SMALLINT	参数类型将为下列类型之一： ● 0 - 普通参数（变量） ● 1 - 结果变量 - 与返回结果集的过程一起使用 ● 2 - SQLSTATE 错误值 ● 3 - SQLCODE 错误值 ● 4 - 函数的返回值
parm_mode_in	CHAR(1)	指出该参数是否向过程（IN 或 INOUT 参数）提供值。
parm_mode_out	CHAR(1)	指出该参数从过程（OUT 或 INOUT 参数）返回值还是在 RESULT 子句中返回列。
domain_id	SMALLINT	用 SYSDOMAIN 系统视图中列出的数据类型号标识参数的数据类型。
width	UNSIGNED INT	包含字符串参数的长度、数字参数的精度或任何其它数据类型的存储字节数。
scale	SMALLINT	对于数字数据类型是小数点后的位数。对于所有其它数据类型，该列的值为 1。
user_type	SMALLINT	参数的用户类型（如果适用）。
parm_name	CHAR(128)	过程参数的名称。
"default"	LONG VARCHAR	参数的缺省值。仅供参考。
remarks	LONG VARCHAR	始终返回 NULL。前提是允许将上一版本的 ODBC 驱动程序与较新的个人数据库服务器配合使用。

基础系统表上的约束

PRIMARY KEY (proc_id, parm_id)

FOREIGN KEY (proc_id) 引用 SYS.ISYSPROCEDURE (proc_id)

FOREIGN KEY (domain_id) 引用 SYS.ISYSDOMAIN (domain_id)

FOREIGN KEY (user_type) 引用 SYS.ISYSUSERTYPE (type_id)

SYSPROCPERM 系统视图

SYSPROCPERM 系统视图的每行都描述一个被授权执行过程的用户。只有得到授权的用户才能执行过程。该视图的基础系统表为 ISYSPROCPERM。

列名	列类型	说明
proc_id	UNSIGNED INT	过程号唯一标识被授予权限的过程。
grantee	UNSIGNED INT	接收权限的用户的用户号。

基础系统表上的约束

PRIMARY KEY (proc_id, grantee)

FOREIGN KEY (grantee) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY (proc_id) 引用 SYS.ISYSPROCEDURE (proc_id)

SYSPROXYTAB 系统视图

SYSPROXYTAB 系统视图的每行描述一个代理表的远程参数。该视图的基础系统表为 ISYSPROXYTAB。

列名	列类型	说明
table_object_id	UNSIGNED BIGINT	代理表的对象 ID。
existing_obj	CHAR(1)	指出远程服务器上先前是否存在代理表。
srvid	UNSIGNED INT	与代理表相关的远程服务器的唯一 ID。
remote_location	LONG VARCHAR	远程服务器上代理表的位置。

基础系统表上的约束

PRIMARY KEY (table_object_id)

FOREIGN KEY (table_object_id) 引用 ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (srvid) 引用 SYS.ISYSSERVER (srvid)

SYSPUBLICATION 系统视图

SYSPUBLICATION 系统视图中的每一行都描述一个 SQL Remote 或 MobiLink 发布。该视图的基础系统表为 ISYSPUBLICATION。

列名	列类型	说明
publication_id	UNSIGNED INT	唯一标识发布的编号。
object_id	UNSIGNED BIGINT	发布的内部 ID，在数据库中唯一标识该发布。
creator	UNSIGNED INT	发布的所有者。
publication_name	CHAR(128)	发布的名称。
remarks	LONG VARCHAR	有关发布的注释。该值存储在 ISYSREMARK 系统表中。
type	CHAR(1)	不建议使用此列。
sync_type	UNSIGNED INT	发布的同步类型。值包括： ● 日志扫描 - 这是一个常规发布，它使用事务日志上载自上次上载以来发生更改的全部相关数据。 ● 脚本式上载 - 对于该发布将忽略事务日志并由用户使用存储过程定义上载。有关存储过程的信息存储在 ISYSSYNCSCRIPT 系统表中。 ● 仅下载 - 这是一个仅下载发布；不上载任何数据。

基础系统表上的约束

PRIMARY KEY (publication_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (creator) 引用 SYS.ISYSUSER (user_id)

另请参见

- “脚本式上载” 《MobiLink - 客户端管理》
- “SYSSYNCSCRIPT 系统视图”一节第 973 页

SYSREMARK 系统视图

SYSREMARK 系统视图的每行描述一个对象的一个注释。该视图的基础系统表为 ISYSREMARK。

列	数据类型	说明
object_id	UNSIGNED BIGINT	具有关联注释的对象的内部 ID。
remarks	LONG VARCHAR	与对象关联的注释。

基础系统表上的约束

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

SYSREMOTEOPTION 系统视图

SYSREMOTEOPTION 系统视图的每一行都描述一个 SQL Remote 消息链接参数的值。该视图的基础系统表为 ISYSREMOTEOPTION。

该视图中的一些列包含可能比较敏感的数据。为此，仅限具有 DBA 权限的用户能够访问该视图。SYSREMOTEOPTION2 视图提供了访问此视图中除可能的敏感列以外的数据的公共方法。

列	数据类型	说明
option_id	UNSIGNED INT	消息链接参数的标识号。
user_id	UNSIGNED INT	为其设置参数的用户 ID。
"setting"	VARCHAR(255)	消息链接参数的值。

基础系统表上的约束

PRIMARY KEY (option_id, user_id)

FOREIGN KEY (option_id) 引用 SYS.ISYSREMOTEOPTIONTYPE (option_id)

FOREIGN KEY (user_id) 引用 SYS.ISYSUSER (user_id)

SYSREMOTEOPTIONTYPE 系统视图

SYSREMOTEOPTIONTYPE 系统视图的每一行都描述一个 SQL Remote 消息链接参数。该视图的基础系统表为 ISYSREMOTEOPTIONTYPE。

列	数据类型	说明
option_id	UNSIGNED INT	消息链接参数的标识号。
type_id	SMALLINT	使用该参数的消息类型的标识号。

列	数据类型	说明
"option"	VARCHAR(128)	消息链接参数的名称。

基础系统表上的约束

PRIMARY KEY (option_id)

FOREIGN KEY (type_id) 引用 SYS.ISYSREMOTETYPE (type_id)

SYSREMOTETYPE 系统视图

SYSREMOTETYPE 系统视图包含有关 SQL Remote 的信息。该视图的基础系统表为 ISYSREMOTETYPE。

列名	列类型	说明
type_id	SMALLINT	标识使用 SQL Remote 所支持的哪种消息系统向该用户发送消息。
object_id	UNSIGNED BIGINT	远程类型的内部 ID，在数据库中唯一标识该远程类型。
type_name	CHAR(128)	SQL Remote 所支持的消息系统的名称。
publisher_address	LONG VARCHAR	远程数据库发布者的地址。
remarks	LONG VARCHAR	有关远程类型的注释。该值存储在 ISYSREMARK 系统表中。

基础系统表上的约束

PRIMARY KEY (type_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

SYSREMOTEUSER 系统视图

SYSREMOTEUSER 系统视图的每一行都描述一个具有 REMOTE 权限的用户 ID（预订者），以及发送自/至该用户的 SQL Remote 消息的状态。该视图的基础系统表为 ISYSREMOTEUSER。

列名	列类型	说明
user_id	UNSIGNED INT	具有 REMOTE 权限的用户的用户号。

列名	列类型	说明
consolidate	CHAR(1)	指出是授予了用户 CONSOLIDATE 权限 (Y) 还是 REMOTE 权限 (N)。
type_id	SMALLINT	标识使用 SQL Remote 所支持的哪种消息系统向此用户发送消息。
address	LONG VARCHAR	SQL Remote 消息要发送到的地址。该地址必须符合 address_type。
frequency	CHAR(1)	发送 SQL Remote 消息的频率。
send_time	TIME	下一次将消息发送给该用户的时间。
log_send	UNSIGNED BIGINT	仅将消息发送给 log_send 大于 log_sent 的预订者。
time_sent	TIMESTAMP	将最新的消息发送到该预订者的时间。
log_sent	UNSIGNED BIGINT	最近的发送操作的日志偏移。
confirm_sent	UNSIGNED BIGINT	此预订者最近一次确认的操作的日志偏移。
send_count	INTEGER	已发送的 SQL Remote 消息数。
resend_count	INTEGER	用于确保在预订者数据库上只应用一次消息的计数器。
time_received	TIMESTAMP	收到来自该预订者的最新消息的时间。
log_received	UNSIGNED BIGINT	预订者数据库中当前数据库上最近收到的操作的日志偏移。
confirm_received	UNSIGNED BIGINT	预订者数据库中已发送确认消息的最近操作的日志偏移。
receive_count	INTEGER	已接收到的消息数量。
rereceive_count	INTEGER	用于确保在当前数据库上只应用一次消息的计数器。

基础系统表上的约束

PRIMARY KEY(user_id)

FOREIGN KEY (user_id) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY (type_id) 引用 SYS.ISYSREMOTETYPE (type_id)

SYSSCHEDULE 系统视图

SYSSCHEDULE 系统视图中的每一行都描述事件的触发时间（由 CREATE EVENT 的 SCHEDULE 子句指定）。该视图的基础系统表为 ISYSSCHEDULE。

列名	列类型	说明
event_id	UNSIGNED INT	指派给每个事件的唯一编号。
sched_name	VARCHAR(128)	与事件的调度相关联的名称。
recurring	TINYINT	指出调度是否重复。
start_time	TIME	调度开始时间。
stop_time	TIME	调度停止时间（如果使用了 BETWEEN）。
start_date	DATE	调度事件执行的第一个日期。
days_of_week	TINYINT	用于指出在星期几调度事件的位掩码： ● x01 = 星期日 ● x02 = 星期一 ● x04 = 星期二 ● x08 = 星期三 ● x10 = 星期四 ● x20 = 星期五 ● x40 = 星期六
days_of_month	UNSIGNED INT	用于指出在该月中的哪天调度事件的位掩码。一些示例包括： ● x01 = 1 号 ● x02 = 2 号 ● x40000000 = 31 号 ● x80000000 = 每月的最后 1 天
interval_units	CHAR(10)	EVERY 指定的间隔单位： ● HH = 小时 ● NN = 分钟 ● SS = 秒
interval_amt	INTEGER	EVERY 指定的周期。

基础系统表上的约束

PRIMARY KEY (event_id, sched_name)

FOREIGN KEY (event_id) 引用 SYS.ISYSEVENT (event_id)

SYSSERVER 系统视图

SYSSERVER 系统视图中的每一行都描述一个远程服务器。该视图的基础系统表为 ISYSSERVER。

注意
以前版本的目录中包含一个 SYSSERVERS 系统表。该表已重命名为 ISYSSERVER（无 'S'），并且是本视图的基础表。

列名	列类型	说明
srvid	UNSIGNED INT	远程服务器的标识符。
srvname	VARCHAR(128)	远程服务器的名称。
srvclass	LONG VARCHAR	服务器类（在 CREATE SERVER 语句中指定）。
srvinfo	LONG VARCHAR	服务器信息。
srvreadonly	CHAR(1)	服务器是否为只读。

基础系统表上的约束

PRIMARY KEY (srvid)

SYSSOURCE 系统视图

SYSSOURCE 系统视图中的每一行都包含 SYSOBJECT 系统视图中所列的一个对象的源代码（如果适用）。该视图的基础系统表为 ISYSSOURCE。

列名	列类型	说明
object_id	UNSIGNED BIGINT	要定义其源代码的对象的内部 ID。
source	LONG VARCHAR	当创建对象时，如果 preserve_source_format 数据库选项为 On，则该列包含该对象的原始源代码。有关详细信息，请参见“preserve_source_format 选项 [数据库]”一节 《SQL Anywhere 服务器 - 数据库管理》。

基础系统表上的约束

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

SYSSQLSERVERTYPE 系统视图

SYSSQLSERVERTYPE 系统视图包含有关与 Adaptive Server Enterprise 兼容的信息。该视图的基础系统表为 ISYSSQLSERVERTYPE。

列名	列类型	说明
ss_user_type	SMALLINT	Adaptive Server Enterprise 用户类型。
ss_domain_id	SMALLINT	Adaptive Server Enterprise 域 ID。
ss_type_name	VARCHAR(30)	Adaptive Server Enterprise 类型名称。
primary_sa_domain_id	SMALLINT	相应的 SQL Anywhere 主域 ID。
primary_sa_user_type	SMALLINT	相应的 SQL Anywhere 主用户类型。

基础系统表上的约束

PRIMARY KEY (ss_user_type)

SYSSUBSCRIPTION 系统视图

SYSSUBSCRIPTION 系统视图中的每一行都描述一个用户 ID（必须有 REMOTE 权限）对一个发布的预订。该视图的基础系统表为 ISYSSUBSCRIPTION。

列名	列类型	说明
publication_id	UNSIGNED INT	此用户 ID 所预订的发布的标识符。
user_id	UNSIGNED INT	发布预订者的用户 ID。
subscribe_by	CHAR(128)	预订的 SUBSCRIBE BY 表达式（如果有）的值。
created	UNSIGNED BIGINT	事务日志中创建预订的位置所在的偏移。
started	UNSIGNED BIGINT	事务日志中启动预订的位置所在的偏移。

基础系统表上的约束

PRIMARY KEY (publication_id, user_id, subscribe_by)

FOREIGN KEY (publication_id) 引用 SYS.ISYSPUBLICATION (publication_id)

FOREIGN KEY (user_id) 引用 SYS.ISYSUSER (user_id)

SYSSYNC 系统视图

SYSSYNC 系统视图包含与 MobiLink 同步相关的信息。该视图中的一些列包含可能比较敏感的数据。为此，仅限具有 DBA 权限的用户能够访问该视图。SYSSYNC2 视图提供了访问此视图中除可能的敏感列以外的数据的公共方法。该视图的基础系统表为 ISYSSYNC。

列名	列类型	说明
sync_id	UNSIGNED INT	唯一标识行的编号。
type	CHAR(1)	同步对象的类型：D 表示定义，T 表示模板，S 表示站点。
publication_id	UNSIGNED INT	在 SYSPUBLICATION 系统视图中找到的 publication_id。
progress	UNSIGNED BIGINT	上次成功上载的日志偏移。
site_name	CHAR(128)	MobiLink 用户名。
"option"	LONG VARCHAR	同步选项。
server_connect	LONG VARCHAR	MobiLink 服务器的地址或 URL。
server_conn_type	LONG VARCHAR	进行同步时使用的通信协议（如 TCP/IP）。
last_download_time	TIMESTAMP	指出上次从 MobiLink 服务器接收下载流的时间。
last_upload_time	TIMESTAMP	指出上次成功上载信息的时间（在 MobiLink 服务器上测量的时间）。缺省值为 jan-1-1900。
created	UNSIGNED BIGINT	创建预订的位置所在的日志偏移。
log_sent	UNSIGNED BIGINT	日志进度，到该进度为止的信息已被上载。不必接收到上载确认即可更新该列中的条目。
generation_number	INTEGER	对于基于文件的下载，是指接收到的此预订的上一个世代号。缺省值为 0。
extended_state	VARCHAR(1024)	仅供内部使用。

基础系统表上的约束

PRIMARY KEY (sync_id)

FOREIGN KEY (publication_id) 引用 SYS.ISYSPUBLICATION (publication_id)

SYSSYNCSCRIPT 系统视图

SYSSYNCSCRIPT 系统视图的每一行都标识一个 MobiLink 脚本式上载的存储过程。该视图几乎与 SYSSYNCSCRIPTS 视图完全相同，只是该视图的值以其原始格式显示。要查看以人工可读格式显示的这些值，请参见“SYSSYNCSCRIPTS 统一视图”一节第 1000 页。

有关哪些发布使用脚本式上载的信息，请参见“SYSPUBLICATION 系统视图”一节第 965 页。

有关存储过程定义的信息，请参见“SYSPROCEDURE 系统视图”一节第 961 页。

该视图的基础系统表为 ISYSSYNCSCRIPT。

列名	列类型	说明
pub_object_id	UNSIGNED BIGINT	脚本所属的发布的对象 ID。
table_object_id	UNSIGNED BIGINT	应用脚本的表的对象 ID。
type	UNSIGNED INT	上载过程的类型。
proc_object_id	UNSIGNED BIGINT	用于发布的存储过程的对象 ID。

基础系统表上的约束

PRIMARY KEY (pub_object_id, table_object_id, type)

FOREIGN KEY (pub_object_id) 引用 SYS.ISYSOBJECT (object_id)

FOREIGN KEY (table_object_id) 引用 SYS.ISYSOBJECT (object_id)

FOREIGN KEY (proc_object_id) 引用 SYS.ISYSOBJECT (object_id)

另请参见

- “脚本式上载” 《MobiLink - 客户端管理》
- “SYSPUBLICATION 系统视图”一节第 965 页
- “SYSPROCEDURE 系统视图”一节第 961 页
- “SYSSYNCSCRIPTS 统一视图”一节第 1000 页

SYSTAB 系统视图

SYSTAB 系统视图的每一行都描述数据库中的一个表或视图。视图的附加信息可在 SYSVIEW 系统视图中找到。该视图的基础系统表为 ISYSTAB。

列名	列类型	说明
table_id	UNSIGNED INT	每个表都有一个指定的唯一编号（表号）。
dbspace_id	SMALLINT	指出哪个 dbspace 包含该表的值。

列名	列类型	说明
file_id	SMALLINT	不建议使用。此列存在于 SYSVIEW 中，但不存在于基础系统表 ISYSTAB 中。此列的内容与 dbspace_id 相同，并且是为了兼容而提供的。请改用 dbspace_id。
count	UNSIGNED BIGINT	表或实例化视图中的行数。该值在每次成功的检查点操作中都会更新。SQL Anywhere 在优化数据库访问时使用这个数字。对于非实例化视图或远程表，count 始终为 0。
creator	UNSIGNED INT	表或视图的所有者的用户号。
table_page_count	INTEGER	该基础表使用的主页总数。
ext_page_count	INTEGER	该基础表使用的扩展页总数。
commit_action	INTEGER	对于全局临时表，0 表示创建表时指定了 ON COMMIT PRESERVE ROWS 子句，1 表示创建表时指定了 ON COMMIT DELETE ROWS 子句（临时表的缺省行为），而 3 表示创建表时指定了 NOT TRANSACTIONAL 子句。对于非临时表，commit_action 始终为 0。
share_type	INTEGER	对于全局临时表，4 表示创建表时指定了 SHARE BY ALL 子句，而 5 表示创建表时*未*指定 SHARE BY ALL 子句。对于非临时表，share_type 始终为 5，因为在创建非临时表时无法指定 SHARE BY ALL 子句。
object_id	UNSIGNED BIGINT	表的对象 ID。
last_modified_at	TIMESTAMP	上次修改表中数据的时间。该列仅在检查点时间更新。
last_modified_tsn	UNSIGNED BIGINT	分配给修改表的事务的序列号。
table_name	CHAR(128)	表或视图的名称。一个创建者不能有两个同名的表或视图。
table_type	TINYINT	表或视图的类型。值包括： ● 1 - 基表 ● 2 - 实例化视图 ● 3 - 全局临时表 ● 4 - 本地临时表 ● 5 - 文本索引基表 ● 6 - 文本索引全局临时表 ● 21 - 视图

列名	列类型	说明
replicate	CHAR(1)	指出基础表是否为复制服务器安装中主数据源的值。
server_type	TINYINT	基础表的数据位置。值包括： ● 1 - 本地服务器 (SQL Anywhere) ● 2 - 远程服务器
tab_page_list	LONG VARBIT	仅供内部使用。包含表的信息的页集，用位图表示。
ext_page_list	LONG VARBIT	仅供内部使用。包含表的行扩展和大对象 (LOB) 页的页集，用位图表示。
pct_free	UNSIGNED INT	表的 PCT_FREE 规范（如果已经指定了一个）；否则为 NULL。
clustered_index_id	UNSIGNED INT	表的聚簇索引的 ID。如果没有聚簇索引，则该字段为 NULL。
encrypted	CHAR(1)	是否加密表或实例化视图。
file_id	SMALLINT	
table_type_str	CHAR(9)	table_type 的可读值。值包括： ● BASE - 基表 ● MAT VIEW - 实例化视图 ● GBL TEMP - 全局临时表 ● VIEW - 视图

基础系统表上的约束

FOREIGN KEY (dbspace_id) 引用 SYS.ISYSDBSPACE (dbspace_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id)

PRIMARY KEY(table_id)

FOREIGN KEY (creator) 引用 SYS.ISYSUSER (user_id)

另请参见

● “SYSVIEW 系统视图”一节第 986 页

SYSTABCOL 系统视图

SYSTABCOL 系统视图为数据库中每个表和视图的每列都包含一行。该视图的基础系统表为 ISYSTABCOL。

列名	列类型	说明
table_id	UNSIGNED INT	该列所属的表或视图的对象 ID。
column_id	UNSIGNED INT	列的 ID。各个表的列的编号都是从 1 开始。
domain_id	SMALLINT	该列的数据类型，由 SYSDOMAIN 系统视图中所列的数据类型编号标识。
nulls	CHAR(1)	指出该列中是否允许 NULL 值。
width	UNSIGNED INT	字符串列的长度、数字列的精度或任何其它数据类型的存储字节数。
scale	SMALLINT	对于 NUMERIC 或 DECIMAL 数据类型的列，是指小数点后的位数。对于字符串列，值 1 表示字符长度语义，而 0 表示字节长度语义。
object_id	UNSIGNED BIGINT	表列的对象 ID。
max_identity	BIGINT	如果该列为 AUTOINCREMENT、IDENTITY 或 GLOBAL AUTOINCREMENT 列，则为列的最大值。
column_name	CHAR(128)	列的名称。
"default"	LONG VARCHAR	列的缺省值。如果指定，则仅在 INSERT 语句没有指定列值时才使用该值。
user_type	SMALLINT	使用用户定义的数据类型定义列时的数据类型。
column_type	CHAR(1)	列的类型（C 表示计算列，而 R 表示其它)。
compressed	TINYINT	该列是否以压缩格式存储。
collect_stats	TINYINT	系统是否自动收集并更新该列的统计信息。

列名	列类型	说明
inline_max	SMALLINT	行中存储 BLOB 的最大字节数。NULL 值表示已应用缺省值，或该列不是字符或二进制类型。非 NULL 的 inline_max 值与使用 CREATE TABLE 或 ALTER TABLE 语句为列指定的 INLINE 值相对应。有关 INLINE 子句的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。
inline_long	SMALLINT	当 BLOB 的大小超过 inline_max 值时，在行中存储的 BLOB 的重复字节数。NULL 值表示已应用缺省值，或该列不是字符或二进制类型。非 NULL 的 inline_long 值与使用 CREATE TABLE 或 ALTER TABLE 语句为列指定的 PREFIX 值相对应。有关 PREFIX 子句的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。
lob_index	TINYINT	是否对列中超过内部阈值大小（约八个数据库页）的 BLOB 值建立索引。NULL 值表示已应用缺省值，或该列不是 BLOB 类型。值 1 表示将建立索引。值 0 表示不建立建索引。非 NULL 的 lob_index 值与使用 CREATE TABLE 或 ALTER TABLE 语句为列指定了 INDEX 还是 NO INDEX 相对应。有关 [NO] INDEX 子句的详细信息，请参见“CREATE TABLE 语句”一节第 497 页。

基础系统表上的约束

PRIMARY KEY (table_id, column_id)

FOREIGN KEY (table_id) 引用 SYS.ISYSTAB (table_id)

FOREIGN KEY (domain_id) 引用 SYS.ISYSDOMAIN (domain_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (user_type) 引用 SYS.ISYSUSERTYPE (type_id)

SYSTABLEPERM 系统视图

GRANT 语句所赋予的权限存储在 SYSTABLEPERM 系统视图中。该视图中的每一行都对应于一个表、一个授予权限的用户 ID (grantor) 和一个被授予权限的用户 ID (grantee)。该视图的基础系统表为 ISYSTABLEPERM。

列名	列类型	说明
stable_id	UNSIGNED INT	对其应用了权限的表或视图的表号。
grantee	UNSIGNED INT	接收权限的用户 ID 的用户号。

列名	列类型	说明
grantor	UNSIGNED INT	授予权限的用户 ID 的用户号。
selectauth	CHAR(1)	指出是否已授予 SELECT 权限。可能的值为 Y、N 或 G。有关这些值的含义的进一步信息，请参见下面的注释区域。
insertauth	CHAR(1)	指出是否已授予 INSERT 权限。可能的值为 Y、N 或 G。有关这些值的含义的进一步信息，请参见下面的注释区域。
deleteauth	CHAR(1)	指出是否已授予 DELETE 权限。可能的值为 Y、N 或 G。有关这些值的含义的进一步信息，请参见下面的注释区域。
updateauth	CHAR(1)	指出是否已为表中的所有列授予 UPDATE 权限。可能的值为 Y、N 或 G。有关这些值的含义的进一步信息，请参见下面的注释区域。
updatecols	CHAR(1)	指出是否仅为基础表中的某些列授予了 UPDATE 权限。如果 updatecols 的值为 Y，则 SYSCOLPERM 系统视图中将有一行或多行为该列授予 UPDATE 权限。
alterauth	CHAR(1)	指出是否已授予 ALTER 权限。可能的值为 Y、N 或 G。有关这些值的含义的进一步信息，请参见下面的注释区域。
referenceauth	CHAR(1)	指出是否已授予 REFERENCE 权限。可能的值为 Y、N 或 G。有关这些值的含义的进一步信息，请参见下面的注释区域。

注释

有多种可以授予的权限。每种权限都可以有下列三个值之一。

- **N** 否，授予者没有授予被授予者该权限。
- **Y** 是，授予者已授予被授予者该权限。
- **G** 被授予者已被授予该权限并能将同样的权限授予其他用户。请参见“GRANT 语句”一节 第 595 页。

权限
另一个授予者可能已授予被授予者操作同一个表的权限。如果是这样，可在 SYSTABLEPERM 系统视图的不同行中找到该信息。

基础系统表上的约束

PRIMARY KEY (stable_id, grantee, grantor)

FOREIGN KEY (stable_id) 引用 SYS.ISYSTAB (table_id)

FOREIGN KEY (grantor) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY (grantee) 引用 SYS.ISYSUSER (user_id)

SYSTEXTCONFIG 系统视图

SYSTEXTCONFIG 系统视图中的每一行都描述了一个文本配置对象，以与全文搜索功能配合使用。该视图的基础系统表为 ISYSTEXTCONFIG。

有关各配置设置含义的详细信息，请参见“文本配置对象设置”一节 《SQL Anywhere 服务器 - SQL 的用法》。

有关全文搜索功能的详细信息，请参见“全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》。

列名	列类型	说明
object_id	UNSIGNED BIGINT	文本配置对象的对象 ID。
creator	UNSIGNED INT	文本配置对象的创建者。
term_breaker	TINYINT	用于将字符串分隔为术语或字的算法。对于 GENERIC，值为 0；对于 NGRAM，值为 1。GENERIC 算法会将所有由一个或多个字母数字字符构成并由非字母数字分隔的字符串视为一个术语。NGRAM 用于近似匹配或不使用空格分隔术语的文档。
stemmer	TINYINT	仅供内部使用。
min_term_length	TINYINT	术语所允许的最小长度（以字符数表示）。忽略长度小于 min_term_length 的术语。 MINIMUM TERM LENGTH 设置仅对 GENERIC 术语断开器有意义。对于 NGRAM 文本索引，将忽略该设置。
max_term_length	TINYINT	对于 GENERIC 文本索引，术语所允许的最大长度（以字符数表示）。忽略长度大于 max_term_length 的术语。 对于 NGRAM 文本索引，该设置是用于断开术语的 n 元语法词的长度。
collation	CHAR(128)	仅供内部使用。
text_config_name	CHAR(128)	文本配置对象的名称。
prefilter	CHAR(128)	仅供内部使用。
postfilter	CHAR(128)	仅供内部使用。

列名	列类型	说明
char_stoplist	LONG VARCHAR	在 CHAR 列上执行全文搜索时要忽略的术语。还可以从文本索引中忽略这些术语。在从 default_char 创建文本配置对象时使用此列。
nchar_stoplist	LONG NVARCHAR	在 NCHAR 列上执行全文搜索时要忽略的术语。还可以从文本索引中忽略这些术语。在从 default_nchar 创建文本配置对象时使用此列。

基础系统表上的约束

PRIMARY KEY (object_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (creator) 引用 SYS.ISYSUSER (user_id)

SYSTEXTIDX 系统视图

SYSTEXTIDX 系统视图中的每一行都描述一个文本索引。该视图的基础系统表为 ISYSTEXTIDX。

有关全文搜索功能的详细信息，请参见“全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》。

列名	列类型	说明
index_id	UNSIGNED BIGINT	SYSIDX 中文本索引的对象 ID。
sequence	UNSIGNED INT	仅供内部使用。
status	UNSIGNED INT	仅供内部使用。
text_config	UNSIGNED BIGINT	SYSTEXTCONFIG 中文本配置对象的对象 ID。
next_handle	UNSIGNED INT	仅供内部使用。
last_handle	UNSIGNED INT	仅供内部使用。
deleted_length	UNSIGNED BIGINT	文本索引中已删除的索引值的总大小。
pending_length	UNSIGNED BIGINT	将要在下次刷新时添加到文本索引的索引值的总大小。
refresh_type	TINYINT	刷新的类型。以下值之一： ● 1 - MANUAL ● 2 - AUTO ● 3 - IMMEDIATE

列名	列类型	说明
refresh_interval	UNSIGNED INT	AUTO REFRESH 间隔（以分钟为单位）。
last_refresh	TIMESTAMP	上次刷新的时间。

基础系统表上的约束

PRIMARY KEY (index_id, sequence)

FOREIGN KEY (index_id) 引用 SYS.ISYSOBJECT (object_id)

FOREIGN KEY (text_config) 引用 SYS.ISYSTEXTCONFIG (object_id)

SYSTEXTIDXTAB 系统视图

SYSTEXTIDXTAB 系统视图中的每一行都描述了已生成的表，而该表是文本索引的一部分。该视图的基础系统表为 ISYSTEXTIDXTAB。

有关全文搜索功能的详细信息，请参见“全文搜索”一节 《SQL Anywhere 服务器 - SQL 的用法》。

列名	列类型	说明
index_id	UNSIGNED BIGINT	仅供内部使用。
sequence	UNSIGNED INT	仅供内部使用。
table_type	UNSIGNED INT	仅供内部使用。
table_id	UNSIGNED INT	仅供内部使用。

基础系统表上的约束

PRIMARY KEY (index_id, sequence, table_type)

FOREIGN KEY (index_id, sequence) 引用 SYS.ISYSTEXTIDX (index_id, sequence)

FOREIGN KEY (table_id) 引用 SYS.ISYSTAB (table_id)

SYSTRIGGER 系统视图

SYSTRIGGER 系统视图的每一行都描述数据库中的一个触发器。该视图还包含自动为具有参照触发操作（例如 ON DELETE CASCADE）的外键定义创建的触发器。该视图的基础系统表为 ISYSTRIGGER。

列名	列类型	说明
trigger_id	UNSIGNED INT	每个触发器都有一个指定的唯一编号（触发器号）。
table_id	UNSIGNED INT	触发器所属的表的表 ID。
object_id	UNSIGNED BIGINT	导致触发器触发的事件。该单字符值对应于创建触发器时指定的触发事件。 ● A - INSERT, DELETE ● B - INSERT, UPDATE ● C - UPDATE COLUMNS ● D - DELETE ● E - DELETE, UPDATE ● I - INSERT ● U - UPDATE ● M - INSERT, DELETE, UPDATE
事件	CHAR(1)	触发器的触发时间。该单字符值对应于创建触发器时指定的触发器时间。
trigger_time	CHAR(1)	● A - AFTER（行级触发器） ● B - BEFORE（行级触发器） ● I - INSTEAD OF（行级触发器） ● K - INSTEAD OF（语句级触发器） ● R - RESOLVE ● S - AFTER（语句级触发器）
trigger_order	SMALLINT	触发器的触发顺序。这决定了当有相同类型（INSERT、UPDATE 或 DELETE）的触发器同时（BEFORE 或者 AFTER）触发时触发器的触发顺序。
foreign_table_id	UNSIGNED INT	包含有参照触发操作（如 ON DELETE CASCADE）的外键定义的表的表号。
foreign_key_id	UNSIGNED INT	foreign_table_id 所引用表的外键的外键号。
referential_action	CHAR(1)	外键所定义的操作。该单字符值对应于创建外键时指定的操作。 ● C - CASCADE ● D - SET DEFAULT ● N - SET NULL ● R - RESTRICT

列名	列类型	说明
trigger_name	CHAR(128)	触发器的名称。一个表中不能有两个同名的触发器。
trigger_defn	LONG VARCHAR	用于创建触发器的命令。
remarks	LONG VARCHAR	有关触发器的注释。该值存储在 ISYSREMARK 系统表中。
source	LONG VARCHAR	触发器的 SQL 源。该值存储在 ISYSSOURCE 系统表中。

基础系统表上的约束

PRIMARY KEY (trigger_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (table_id) 引用 SYS.ISYSTAB (table_id)

FOREIGN KEY fkey_index (foreign_table_id, foreign_key_id) 引用 SYS.ISYSIDX (table_id, index_id)

SYSTYPEMAP 系统视图

SYSTYPEMAP 系统视图包含 SYSSQLSERVERTYPE 系统视图中条目的兼容性映射值。该视图的基础系统表为 ISYSTYPEMAP。

列名	列类型	说明
ss_user_type	SMALLINT	包含 Adaptive Server Enterprise 用户类型。
sa_domain_id	SMALLINT	包含相应的 SQL Anywhere domain_id。
sa_user_type	SMALLINT	包含相应的 SQL Anywhere 用户类型。
nullable	CHAR(1)	该类型是否允许 NULL 值。

基础系统表上的约束

FOREIGN KEY (sa_domain_id) 引用 SYS.ISYSDOMAIN (domain_id)

SYSUSER 系统视图

SYSUSER 系统视图的每一行都描述系统中的一个用户。该视图的基础系统表为 ISYSUSER。

列名	列类型	说明
user_id	UNSIGNED INT	被分配至登录策略的用户的唯一标识符。
object_id	UNSIGNED BIGINT	用户在数据库中的唯一标识符。
user_name	CHAR(128)	用户的登录名。
password	BINARY(128)	用户的口令。
login_policy_id	UNSIGNED BIGINT	登录策略的唯一标识符。
expired_password_on_login	TINYINT	指示下次登录时用户的口令是否到期。
password_creation_time	TIMESTAMP	用户口令的创建时间。
failed_login_attempts	UNSIGNED INT	用户在帐户被锁定前可以登录失败的次数。
last_login_time	TIMESTAMP	用户上次登录的时间。

注意
对于使用 SQL Anywhere 11.0.0 版及更高版本创建的数据库，会始终加密此视图的基础系统表，以防止他人对数据进行未经授权的访问。

基础系统表上的约束

PRIMARY KEY(user_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

FOREIGN KEY (login_policy_id) 引用 SYS.ISYSLOGINPOLICY (login_policy_id)

另请参见

- “SYSLOGINPOLICY 系统视图”一节第 957 页
- “SYSLOGINPOLICYOPTION 系统视图”一节第 957 页

SYSUSERAUTHORITY 系统视图

SYSUSERAUTHORITY 系统视图的每一行都描述授予一个用户 ID 的权限。该视图的基础系统表为 ISYSUSERAUTHORITY。

列名	列类型	说明
user_id	UNSIGNED INT	权限所属的用户的 ID。
auth	VARCHAR(20)	授予用户的权限。

基础系统表上的约束

PRIMARY KEY (user_id, auth)

FOREIGN KEY (user_id) 引用 SYS.ISYSUSER (user_id)

SYSUSERMESSAGE 系统视图

SYSUSERMESSAGE 系统视图中的每一行都包含有关错误情况的用户定义消息。该视图的基础系统表为 ISYSUSERMESSAGE。

注意
以前版本的目录中包含一个 SYSUSERMESSAGES 系统表。该表已重命名为 ISYSUSERMESSAGE（无 'S'），并且是本视图的基础表。

列名	列类型	说明
error	INTEGER	错误情况的唯一标识号。
uid	UNSIGNED INT	定义消息的用户号。
description	VARCHAR(255)	对应于错误情况的消息。
langid	SMALLINT	保留。

基础系统表上的约束

FOREIGN KEY (uid) 引用 SYS.ISYSUSER (user_id)

SYSUSERTYPE 系统视图

SYSUSERTYPE 系统视图的每一行都包含对一种用户定义数据类型的描述。该视图的基础系统表为 ISYSUSERTYPE。

列名	列类型	说明
type_id	SMALLINT	用户定义数据类型的唯一标识号。
creator	UNSIGNED INT	数据类型所有者的用户号。
domain_id	SMALLINT	该用户定义数据类型所依据的数据类型，由 SYSDOMAIN 系统视图中列出的数据类型号指出。
nulls	CHAR(1)	用户定义的数据类型是否允许空值。可能的值为 Y、N 或 U。值为 U 表示未指定空性。

列名	列类型	说明
width	UNSIGNED INT	字符串列的长度、数字列的精度或任何其它数据类型的存储字节数。
scale	SMALLINT	数字数据类型列的小数点后的位数，对于所有其它数据类型该值为零。
type_name	CHAR(128)	数据类型的名称。
"default"	LONG VARCHAR	数据类型的缺省值。
"check"	LONG VARCHAR	数据类型的 CHECK 条件。

基础系统表上的约束

PRIMARY KEY (type_id)

FOREIGN KEY (creator) 引用 SYS.ISYSUSER (user_id)

FOREIGN KEY (domain_id) 引用 SYS.ISYSDOMAIN (domain_id)

SYSVIEW 系统视图

SYSVIEW 系统视图的每一行都描述数据库中的一个视图。有关视图的附加信息也可在 SYSTAB 系统视图中找到。该视图的基础系统表为 ISYSVIEW。

还可以使用 sa_materialized_view_info 系统过程获取有关实例化视图的格式更加易读的信息。请参见“sa_materialized_view_info 系统过程”一节第 852 页。

列名	列类型	说明
view_object_id	UNSIGNED BIGINT	视图的对象 ID。
view_def	LONG VARCHAR	视图的定义（查询说明）。
mv_build_type	TINYINT	目前未使用。
mv_refresh_type	TINYINT	为视图定义的刷新类型。可能的值为 IMMEDIATE 和 MANUAL。请参见“手动和快速实例化视图”一节《SQL Anywhere 服务器 - SQL 的用法》。

列名	列类型	说明
mv_use_in_optimization	TINYINT	实例化视图是否可用于查询优化（0 表示不能用于优化，1 表示可以用于优化）。请参见“允许和禁止优化程序使用实例化视图”一节 《SQL Anywhere 服务器 - SQL 的用法》。
mv_last_refreshed_at	TIMESTAMP	表示上次刷新实例化视图的日期和时间。
mv_last_refreshed_tsn	UNSIGNED BIGINT	分配给刷新实例化视图的事务的序列号。
mv_known_stale_at	TIMESTAMP	实例化视图失效的时间。该值对应于检测到一个基表发生更改的时间。值为 0 表示该视图为最新的，或已失效但数据库服务器尚未对此进行标记，因为自该视图失效以来一直未使用。使用 sa_materialized_view_info 系统过程来确定实例化视图的状态。请参见“sa_materialized_view_info 系统过程”一节第 852 页。

注释

以 SNAPSHOT 隔离刷新实例化视图时，mv_last_refreshed_at 和 mv_last_refreshed_tsn 引用修改用于实例化视图内容计算的所有行的最早事务。

基础系统表上的约束

PRIMARY KEY (view_object_id)

FOREIGN KEY (view_object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

另请参见

- “SYSTAB 系统视图”一节第 973 页
- “CREATE MATERIALIZED VIEW 语句”一节第 455 页
- “REFRESH MATERIALIZED VIEW 语句”一节第 665 页
- “CREATE VIEW 语句”一节第 520 页

SYSWEBSERVICE 系统视图

SYSWEBSERVICE 系统视图的每一行都包含对一种 Web 服务的描述。该视图的基础系统表为 ISYSWEBSERVICE。

列名	列类型	说明
service_id	UNSIGNED INT	Web 服务的唯一标识号。

列名	列类型	说明
object_id	UNSIGNED BIGINT	Web 服务的 ID。
service_name	CHAR(128)	分配给 Web 服务的名称。
service_type	VARCHAR(40)	服务类型，例如 RAW、HTTP、XML、SOAP 或 DISH。
auth_required	CHAR(1)	是否所有请求都必须包含有效用户名和口令。
secure_required	CHAR(1)	是否接受不安全的连接（例如 HTTP），或仅接受安全连接（例如 HTTPS）。
url_path	CHAR(1)	控制对 URL 的解释。
user_id	UNSIGNED INT	如果启用验证，则标识有权使用该服务的用户或用户组。如果禁用验证，则指定处理请求时使用的帐户。
parameter	LONG VARCHAR	标识要包含在 DISH 服务中的 SOAP 服务的前缀。
statement	LONG VARCHAR	响应请求时始终要执行的 SQL 语句。如果为 NULL，则执行在每个请求中包含的任意语句。DISH 类型的服务将忽略此参数。
remarks	LONG VARCHAR	有关 Web 服务的注释。该值存储在 ISYSREMARK 系统表中。
enabled	CHAR(1)	

基础系统表上的约束

PRIMARY KEY (service_id)

FOREIGN KEY (object_id) 引用 SYS.ISYSOBJECT (object_id) MATCH UNIQUE FULL

统一视图

统一视图以用户更为频繁要求的形式提供数据。例如，统一视图经常提供通常需要的连接。统一视图与系统视图不同，因为它们不仅仅是基础系统表中原始数据的直接视图。例如，系统视图中的很多列都是无法理解的 ID 值，而在统一视图中，它们是一些可读的名称。

SYSARTICLECOLS 统一视图

SYSARTICLECOLS 视图中的每一行都标识项目中的一列。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定的表和列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSARTICLECOLS"
  as select p.publication_name,t.table_name,c.column_name
    from SYS.ISYSARTICLECOL as ac
      join SYS.ISYSPUBLICATION as p on p.publication_id = ac.publication_id
      join SYS.ISYSTAB as t on t.table_id = ac.table_id
      join SYS.ISYSTABCOL as c on c.table_id = ac.table_id
      and c.column_id = ac.column_id;
```

另请参见

- “SYSARTICLECOL 系统视图”一节第 939 页
- “SYSPUBLICATION 系统视图”一节第 965 页
- “SYSTAB 系统视图”一节第 973 页
- “SYSTABCOL 系统视图”一节第 976 页

SYSARTICLES 统一视图

SYSARTICLES 视图中的每一行都描述发布中的一个项目。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSARTICLES"
  as select u1.user_name as publication_owner,p.publication_name,
    u2.user_name as table_owner,t.table_name,
    a.where_expr,a.subscribe_by_expr,a.alias
    from SYS.ISYSARTICLE as a
      join SYS.ISYSPUBLICATION as p on(a.publication_id = p.publication_id)
      join SYS.ISYSTAB as t on(a.table_id = t.table_id)
      join SYS.ISYSUSER as u1 on(p.creator = u1.user_id)
      join SYS.ISYSUSER as u2 on(t.creator = u2.user_id);
```

另请参见

- “SYSARTICLE 系统视图”一节第 938 页
- “SYSPUBLICATION 系统视图”一节第 965 页
- “SYSTAB 系统视图”一节第 973 页
- “SYSUSER 系统视图”一节第 983 页

SYSCAPABILITIES 统一视图

SYSCAPABILITIES 视图中的每一行都描述一种功能。该视图从 ISYSCAPABILITY 和 ISYSCAPABILITYNAME 系统表中获取数据。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSCAPABILITIES"
  as select ISYSCAPABILITY.capid,
            ISYSCAPABILITY.srvid,
            property('RemoteCapability', ISYSCAPABILITY.capid) as capname,
            ISYSCAPABILITY.capvalue
    from SYS.ISYSCAPABILITY;
```

另请参见

- “SYSCAPABILITY 系统视图”一节第 939 页
- “SYSCAPABILITYNAME 系统视图”一节第 940 页

SYSCATALOG 统一视图

SYSCATALOG 视图中的每一行都描述一个系统表。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSCATALOG"( creator,
  tname,dbspacename,tabletype,ncols,primary_key,"check",
  remarks )
  as select u.user_name,tab.table_name,dbs.dbspace_name,
    if tab.table_type_str = 'BASE' then 'TABLE' else tab.table_type_str
endif,
    (select count() from SYS.ISYSTABCOL
      where ISYSTABCOL.table_id = tab.table_id),
    if ix.index_id is null then 'N' else 'Y' endif,
    null,
    rmk.remarks
    from SYS.SYSTAB as tab
      join SYS.ISYSDBSPACE as dbs on(tab.dbspace_id = dbs.dbspace_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator
      left outer join SYS.ISYSIDX as ix on(tab.table_id = ix.table_id and
ix.index_id = 0)
     left outer join SYS.ISYSREMARK as rmk on(tab.object_id = rmk.object_id);
```

另请参见

- “SYSTAB 系统视图”一节第 973 页
- “SYSTABCOL 系统视图”一节第 976 页
- “SYSDBSPACE 系统视图”一节第 943 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSIDX 系统视图”一节第 952 页
- “SYSREMARK 系统视图”一节第 965 页

SYSCOLAUTH 统一视图

SYSCOLAUTH 视图的每一行都描述针对列授予的一组权限（UPDATE、SELECT 或 REFERENCES）。SYSCOLAUTH 视图对“SYSCOLPERM 系统视图”一节第 940 页中的数据提供了用户友好的显示。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSCOLAUTH"( grantor,grantee,creator,tname,colname,
  privilege_type,is_grantable )
  as select u1.user_name,u2.user_name,u3.user_name,tab.table_name,
    col.column_name,cp.privilege_type,cp.is_grantable
    from SYS.ISYSCOLPERM as cp
      join SYS.ISYSUSER as u1 on u1.user_id = cp.grantor
      join SYS.ISYSUSER as u2 on u2.user_id = cp.grantee
      join SYS.ISYSTAB as tab on tab.table_id = cp.table_id
      join SYS.ISYSUSER as u3 on u3.user_id = tab.creator
      join SYS.ISYSTABCOL as col on col.table_id = cp.table_id
      and col.column_id = cp.column_id;
```

另请参见

- “SYSCOLPERM 系统视图”一节第 940 页
- “SYSTABCOL 系统视图”一节第 976 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSTAB 系统视图”一节第 973 页

SYSCOLSTATS 统一视图

SYSCOLSTATS 视图中包含有以直方图的形式存储并由优化程序使用的列统计数据。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSCOLSTATS"
  as select u.user_name,t.table_name,c.column_name,
    s.format_id,s.update_time,s.density,s.max_steps,
    s.actual_steps,s.step_values,s.frequencies
    from SYS.ISYSCOLSTAT as s
      join SYS.ISYSTABCOL as c on(s.table_id = c.table_id
      and s.column_id = c.column_id)
      join SYS.ISYSTAB as t on(t.table_id = c.table_id)
      join SYS.ISYSUSER as u on(u.user_id = t.creator)
```

另请参见

- “SYSCOLSTAT 系统视图”一节第 941 页
- “SYSTABCOL 系统视图”一节第 976 页
- “SYSTAB 系统视图”一节第 973 页
- “SYSUSER 系统视图”一节第 983 页

SYSCOLUMNS 统一视图

SYSCOLUMNS 视图的每一行都描述目录中每个表的一个列和视图。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSCOLUMNS"( creator,cname,tname,coltype,nulls,length,
  syslength,in_primary_key,colno,default_value,
  column_kind,remarks )
  as select u.user_name,col.column_name,tab.table_name,dom.domain_name,
    col.nulls,col.width,col.scale,if ixcol.sequence is null then 'N' else 'Y'
endif,col.column_id,
    col."default",col.column_type,rmk.remarks
    from SYS.ISYSTABCOL as col
     left outer join SYS.ISYSIDXCOL as ixcol on(col.table_id = ixcol.table_id
and col.column_id = ixcol.column_id and ixcol.index_id = 0)
      join SYS.ISYSTAB as tab on(tab.table_id = col.table_id)
      join SYS.ISYSDOMAIN as dom on(dom.domain_id = col.domain_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator
      left outer join SYS.ISYSREMARK as rmk on(col.object_id = rmk.object_id)
```

另请参见

- “SYSTABCOL 系统视图”一节第 976 页
- “SYSIDXCOL 系统视图”一节第 953 页
- “SYSTAB 系统视图”一节第 973 页
- “SYSDOMAIN 系统视图”一节第 945 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSREMARK 系统视图”一节第 965 页

SYSFOREIGNKEYS 统一视图

SYSFOREIGNKEYS 视图的每一行都描述了目录中每个表的一个外键。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSFOREIGNKEYS"( foreign_creator,
  foreign_tname,
  primary_creator,primary_tname,role,columns )
  as select fk_up.user_name,fk_tab.table_name,pk_up.user_name,
    pk_tab.table_name,ix.index_name,
    (select list(string(fk_col.column_name,' IS ',
      pk_col.column_name))
      from SYS.ISYSIDXCOL as fkc
        join SYS.ISYSTABCOL as fk_col on(
        fkc.table_id = fk_col.table_id
        and fkc.column_id = fk_col.column_id)
        ,SYS.ISYSTABCOL as pk_col
      where fkc.table_id = fk.foreign_table_id
      and fkc.index_id = fk.foreign_index_id
      and pk_col.table_id = fk.primary_table_id
      and pk_col.column_id = fkc.primary_column_id)
    from SYS.ISYSFKEY as fk
      join SYS.ISYSTAB as fk_tab on fk_tab.table_id = fk.foreign_table_id
```

```
join SYS.ISYSUSER as fk_up on fk_up.user_id = fk_tab.creator
join SYS.ISYSTAB as pk_tab on pk_tab.table_id = fk.primary_table_id
join SYS.ISYSUSER as pk_up on pk_up.user_id = pk_tab.creator
join SYS.ISYSIDX as ix on ix.table_id = fk.foreign_table_id
   and ix.index_id = fk.foreign_index_id
```

另请参见

- “SYSIDXCOL 系统视图”一节第 953 页
- “SYSTABCOL 系统视图”一节第 976 页
- “SYSFKEY 系统视图”一节第 949 页
- “SYSTAB 系统视图”一节第 973 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSIDX 系统视图”一节第 952 页
- “SYSDOMAIN 系统视图”一节第 945 页
- “SYSREMARK 系统视图”一节第 965 页

SYSGROUPS 统一视图

对于每组的每一个成员，在 SYSGROUPS 视图中都有与之对应的一行。该视图介绍了组与成员之间的多对多关系。一个组可以有许多成员，而一个用户也可以是许多组的成员。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSGROUPS"( group_name,
  member_name )
  as select g.user_name,u.user_name
    from SYS.ISYSGROUP, SYS.ISYSUSER as g, SYS.ISYSUSER as u
    where ISYSGROUP.group_id = g.user_id
      and ISYSGROUP.group_member = u.user_id;
```

另请参见

- “SYSGROUP 系统视图”一节第 950 页
- “SYSUSER 系统视图”一节第 983 页

SYSINDEXES 统一视图

SYSINDEXES 视图的每一行都描述数据库中的一个索引。您可以使用 SYSIDX 和 SYSIDXCOL 系统视图替代该视图。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSINDEXES"( icreator,
  iname,fname,creator,tname,indextype,
  colnames,interval,level_num )
  as select u.user_name,idx.index_name,dbs.dbspace_name,u.user_name,
    tab.table_name,
    case idx.index_category
    when 1 then 'Primary Key'
    when 2 then 'Foreign Key'
```

```
when 3 then(
  if idx."unique" = 4 then 'Non-unique'
  else if idx."unique" = 2 then 'UNIQUE constraint'
    else 'Unique'
    endif
  endif) when 4 then 'Text Index' end,
(select list(string(c.column_name,
  if ixc."order" = 'A' then ' ASC' else ' DESC' endif) order by
  ixc.table_id asc,ixc.index_id asc,ixc.sequence asc)
  from SYS.ISYSIDXCOL as ixc
    join SYS.ISYSTABCOL as c on(
    c.table_id = ixc.table_id
    and c.column_id = ixc.column_id)
  where ixc.index_id = idx.index_id
  and ixc.table_id = idx.table_id),
0,0
from SYS.ISYSTAB as tab
  join SYS.ISYSDBSPACE as dbs on(tab.dbspace_id = dbs.dbspace_id)
  join SYS.ISYSIDX as idx on(idx.table_id = tab.table_id)
  join SYS.ISYSUSER as u on u.user_id = tab.creator;
```

另请参见

- “SYSIDXCOL 系统视图”一节第 953 页
- “SYSTABCOL 系统视图”一节第 976 页
- “SYSTAB 系统视图”一节第 973 页
- “SYSDBSPACE 系统视图”一节第 943 页
- “SYSIDX 系统视图”一节第 952 页
- “SYSUSER 系统视图”一节第 983 页

SYSOPTIONS 统一视图

SYSOPTIONS 视图的每一行都描述了用 SET 命令创建的一个选项。对于每个选项，每个用户都可以有自己的设置。此外，PUBLIC 用户的设置定义了缺省设置，以用于没有自己的设置的用户。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSOPTIONS"( user_name,"option",setting )
  as select u.user_name,opt."option",opt.setting
    from SYS.ISYSOPTION as opt
      join SYS.ISYSUSER as u on opt.user_id = u.user_id
```

另请参见

- “SYSOPTION 系统视图”一节第 960 页
- “SYSUSER 系统视图”一节第 983 页

SYSPROCAUTH 统一视图

SYSPROCAUTH 视图的每一行都描述了针对过程授予的一组权限。您可以使用 SYSPROCPERM 系统视图替代该视图。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSPROCAUTH"( grantee,
  creator,procname )
  as select u1.user_name,u2.user_name,p.proc_name
    from SYS.ISYSPROCEDURE as p
      join SYS.ISYSPROCPERM as pp on(p.proc_id = pp.proc_id)
      join SYS.ISYSUSER as u1 on u1.user_id = pp.grantee
      join SYS.ISYSUSER as u2 on u2.user_id = p.creator
```

另请参见

- “SYSPROCEDURE 系统视图”一节第 961 页
- “SYSPROCPERM 系统视图”一节第 964 页
- “SYSUSER 系统视图”一节第 983 页

SYSPROCPARMS 统一视图

SYSPROCPARMS 视图中的每一行都描述数据库中某个过程的一个参数。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSPROCPARMS"( creator,
  procname,parmname,parm_id,parmtype,parmmode,parmdomain,
  length,scale,"default",user_type )
  as select up.user_name,p.proc_name,pp.parm_name,pp.parm_id,pp.parm_type,
    if pp.parm_mode_in = 'Y' and pp.parm_mode_out = 'N' then 'IN'
    else if pp.parm_mode_in = 'N' and pp.parm_mode_out = 'Y' then 'OUT'
      else 'INOUT'
      endif
    endif,dom.domain_name,pp.width,pp.scale,pp."default",ut.type_name
    from SYS.ISYSPROCPARM as pp
      join SYS.ISYSPROCEDURE as p on p.proc_id = pp.proc_id
      join SYS.ISYSUSER as up on up.user_id = p.creator
      join SYS.ISYSDOMAIN as dom on dom.domain_id = pp.domain_id
      left outer join SYS.ISYSUSERTYPE as ut on ut.type_id = pp.user_type;
```

另请参见

- “SYSPROCPARM 系统视图”一节第 962 页
- “SYSPROCEDURE 系统视图”一节第 961 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSDOMAIN 系统视图”一节第 945 页
- “SYSUSERTYPE 系统视图”一节第 985 页

SYSPROCS 统一视图

SYSPROCS 视图显示过程或函数的名称、其创建者的名称以及为过程或函数记录的任何注释。

在以下的 ALTER VIEW 语句中提供了组成该视图的表和列。

```
ALTER VIEW "SYS"."SYSPROCS"( creator,
  procname,remarks )
  as select u.user_name,p.proc_name,r.remarks
    from SYS.ISYSPROCEDURE as p
      join SYS.ISYSUSER as u on u.user_id = p.creator
      left outer join SYS.ISYSREMARK as r on(p.object_id = r.object_id);
```

另请参见

- “SYSPROCEDURE 系统视图”一节第 961 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSREMARK 系统视图”一节第 965 页

SYSPUBLICATIONS 统一视图

SYSPUBLICATIONS 视图中的每一行都描述一个 SQL Remote 或 MobiLink 发布。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSPUBLICATION"
  as select b.publication_id,
    b.object_id,
    b.creator,
    b.publication_name,
    r.remarks,
    b.type,
    b.sync_type
    from SYS.ISYSPUBLICATION as b
      left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
```

另请参见

- “SYSPUBLICATION 系统视图”一节第 965 页
- “SYSREMARK 系统视图”一节第 965 页

SYSREMOTEOPTION2 统一视图

以一种可读性更强的形式显示不包含敏感数据的 SYSREMOTEOPTION 和 SYSREMOTEOPTIONTYPE 中的列。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSREMOTEOPTION2"
  as select ISYSREMOTEOPTION.option_id,
    ISYSREMOTEOPTION.user_id,
    SYS.HIDE_FROM_NON_DBA(ISYSREMOTEOPTION.setting) as setting
    from SYS.ISYSREMOTEOPTION
```

另请参见

- “SYSREMOTEOPTION 系统视图”一节第 966 页

SYSREMOTEOPTIONS 统一视图

SYSREMOTEOPTIONS 视图的每一行都描述 SQL Remote 消息链接参数的值。该视图中的一些列包含可能比较敏感的数据。为此，仅限具有 DBA 权限的用户能够访问该视图。SYSREMOTEOPTION2 视图提供了访问非敏感数据的公共方法。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSREMOTEOPTIONS"
  as select srt.type_name,
    sup.user_name,
    srot."option",
    SYS.HIDE_FROM_NON_DBA(sro.setting) as setting
    from SYS.ISYSREMOTETYPE as srt
      ,SYS.ISYSREMOTEOPTIONTYPE as srot
      ,SYS.ISYSREMOTEOPTION as sro
      ,SYS.ISYSUSER as sup
    where srt.type_id = srot.type_id
    and srot.option_id = sro.option_id
    and sro.user_id = sup.user_id
```

另请参见

- “SYSREMOTETYPE 系统视图”一节第 967 页
- “SYSREMOTEOPTIONTYPE 系统视图”一节第 966 页
- “SYSREMOTEOPTION 系统视图”一节第 966 页
- “SYSUSER 系统视图”一节第 983 页

SYSREMOTETYPES 统一视图

SYSREMOTETYPES 视图的每一行都描述 SQL Remote 消息类型之一，包括发布者地址。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSREMOTETYPES"
  as select rt.type_id,rt.type_name,rt.publisher_address,rm.remarks
    from SYS.ISYSREMOTETYPE as rt
      left outer join SYS.ISYSREMARK as rm on(rt.object_id = rm.object_id)
```

另请参见

- “SYSREMOTETYPE 系统视图”一节第 967 页
- “SYSREMARK 系统视图”一节第 965 页

SYSREMOTEUSERS 统一视图

SYSREMOTEUSERS 视图的每一行都描述一个具有 REMOTE 权限的用户 ID（预订者），以及发送自/至该用户的 SQL Remote 消息的状态。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSREMOTEUSERS"
  as select u.user_name,r.consolidate,t.type_name,r.address,r.frequency,
    r.send_time,
    (if r.frequency = 'A' then null else if r.frequency = 'P' then
        if r.time_sent is null then current timestamp
        else(select min(minutes(a.time_sent,60*hour(a.send_time)
            +minute(seconds(a.send_time,59))))
            from SYS.ISYSREMOTEUSER as a where a.frequency = 'P'
            and a.send_time = r.send_time)
        endif
      else if current date+r.send_time
         > coalesce(r.time_sent,current timestamp) then
          current date+r.send_time else current date+r.send_time+1 endif
      endif endif) as next_send,
    r.log_send,r.time_sent,r.log_sent,r.confirm_sent,r.send_count,
    r.resend_count,r.time_received,r.log_received,
    r.confirm_received,r.receive_count,r.rereceive_count
    from SYS.ISYSREMOTEUSER as r
      join SYS.ISYSUSER as u on(u.user_id = r.user_id)
      join SYS.ISYSREMOTETYPE as t on(t.type_id = r.type_id)
```

另请参见

- “SYSREMOTEUSER 系统视图”一节第 967 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSREMOTETYPE 系统视图”一节第 967 页

SYSSUBSCRIPTIONS 统一视图

每行均介绍一个用户 ID（必须有 REMOTE 权限）对一个发布的预订。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSSUBSCRIPTIONS"
  as select p.publication_name,u.user_name,s.subscribe_by,s.created,
    s.started
    from SYS.ISYSSUBSCRIPTION as s
      join SYS.ISYSPUBLICATION as p on(p.publication_id = s.publication_id)
      join SYS.ISYSUSER as u on u.user_id = s.user_id
```

另请参见

- “SYSSUBSCRIPTION 系统视图”一节第 971 页
- “SYSPUBLICATION 系统视图”一节第 965 页
- “SYSUSER 系统视图”一节第 983 页

SYSSYNC2 统一视图

SYSSYNC2 视图提供了访问 SYSSYNC 系统视图中可找到的数据（与 MobiLink 同步相关的信息）的公共方法—而不会公开可能比较敏感的数据。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSSYNC2"
  as select ISYSSYNC.sync_id,
    ISYSSYNC.type,
    ISYSSYNC.publication_id,
    ISYSSYNC.progress,
    ISYSSYNC.site_name,
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC."option") as "option",
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC.server_connect) as server_connect,
    ISYSSYNC.server_conn_type,
    ISYSSYNC.last_download_time,
    ISYSSYNC.last_upload_time,
    ISYSSYNC.created,
    ISYSSYNC.log_sent,
    ISYSSYNC.generation_number,
    ISYSSYNC.extended_state
    from SYS.ISYSSYNC
```

另请参见

- “SYSSYNC 系统视图”一节第 972 页

SYSSYNCPUBLICATIONDEFAULTS 统一视图

SYSSYNCPUBLICATIONDEFAULTS 视图提供与 MobiLink 同步中涉及的发布关联的缺省同步设置。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSSYNCPUBLICATIONDEFAULTS"
  as select s.sync_id,
    p.publication_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
    s.server_conn_type
    from SYS.ISYSSYNC as s join SYS.ISYSPUBLICATION as p
       on(p.publication_id = s.publication_id) where s.site_name is null
```

另请参见

- “SYSSYNC 系统视图”一节第 972 页
- “SYSPUBLICATION 系统视图”一节第 965 页

SYSSYNCS 统一视图

SYSSYNCS 视图包含与 MobiLink 同步相关的信息。该视图中的一些列包含可能比较敏感的数据。为此，仅限具有 DBA 权限的用户能够访问该视图。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSSYNCS"
  as select p.publication_name,s.progress,s.site_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
    s.server_conn_type,s.last_download_time,
    s.last_upload_time,s.created,s.log_sent,s.generation_number,
    s.extended_state
    from SYS.ISYSSYNC as s
      left outer join SYS.ISYSPUBLICATION as p
      on p.publication_id = s.publication_id
```

另请参见

- “SYSSYNC 系统视图”一节第 972 页
- “SYSPUBLICATION 系统视图”一节第 965 页

SYSSYNCSCRIPTS 统一视图

SYSSYNCSCRIPTS 视图的每一行都标识一个 MobiLink 脚本式上载的存储过程。该视图几乎与 SYSSYNCSCRIPT 视图完全相同，只是其值以人工可读的格式显示，而不是显示原始数据。

```
ALTER VIEW "SYS"."SYSSYNCSCRIPTS"
  as select p.publication_name,
    t.table_name,
    case s.type
    when 0 then 'upload insert'
    when 1 then 'upload delete'
    when 2 then 'upload update'
    else 'unknown'
    end as type,
    c.proc_name
    from SYS.ISYSSYNCSCRIPT as s
      join SYS.ISYSPUBLICATION as p on p.object_id = s.pub_object_id
      join SYS.ISYSTAB as t on t.object_id = s.table_object_id
      join SYS.ISYSPROCEDURE as c on c.object_id = s.proc_object_id
```

另请参见

- “SYSSYNCSCRIPT 系统视图”一节第 973 页
- “SYSPUBLICATION 系统视图”一节第 965 页
- “SYSTAB 系统视图”一节第 973 页
- “SYSPROCEDURE 系统视图”一节第 961 页
- “脚本式上载” 《MobiLink - 客户端管理》

SYSSYNCSUBSCRIPTIONS 统一视图

SYSSYNCSUBSCRIPTIONS 视图包含与 MobiLink 同步预订关联的同步设置。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSSYNCSUBSCRIPTIONS"
  as select s.sync_id,
    p.publication_name,
```



```
    s.progress,
    s.site_name,
    SYS.HIDE_FROM_NON_DBA(s."option") as "option",
    SYS.HIDE_FROM_NON_DBA(s.server_connect) as server_connect,
    s.server_conn_type,
    s.last_download_time,
    s.last_upload_time,
    s.created,
    s.log_sent,
    s.generation_number,
    s.extended_state
    from SYS.ISYSSYNC as s join SYS.ISYSPUBLICATION as p on(p.publication_id
= s.publication_id)
    where s.publication_id is not null and
    s.site_name is not null and exists
    (select 1 from SYS.SYSSYNCUSERS as u
      where s.site_name = u.site_name)
```

另请参见

- “SYSSYNC 系统视图”一节第 972 页
- “SYSPUBLICATION 系统视图”一节第 965 页
- “SYSSYNCUSERS 统一视图”一节第 1001 页

SYSSYNCUSERS 统一视图

与 MobiLink 同步用户关联的同步设置视图。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSSYNCUSERS"
  as select ISYSSYNC.sync_id,
    ISYSSYNC.site_name,
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC."option") as "option",
    SYS.HIDE_FROM_NON_DBA(ISYSSYNC.server_connect) as server_connect,
    ISYSSYNC.server_conn_type
    from SYS.ISYSSYNC where
    ISYSSYNC.publication_id is null
```

另请参见

- “SYSSYNC 系统视图”一节第 972 页

SYSTABAUTH 统一视图

SYSTABAUTH 视图包含来自 SYSTABLEPERM 系统视图的信息，但采用了更加易读的格式。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSTABAUTH"( grantor,
  grantee,screator,stname,tcreator,ttname,
  selectauth,insertauth,deleteauth,
  updateauth,updatecols,alterauth,referenceauth )
```

```
as select u1.user_name,u2.user_name,u3.user_name,tab1.table_name,
  u4.user_name,tab2.table_name,tp.selectauth,tp.insertauth,
  tp.deleteauth,tp.updateauth,tp.updatecols,tp.alterauth,
  tp.referenceauth
  from SYS.ISYSTABLEPERM as tp
    join SYS.ISYSUSER as u1 on u1.user_id = tp.grantor
    join SYS.ISYSUSER as u2 on u2.user_id = tp.grantee
    join SYS.ISYSTAB as tab1 on tab1.table_id = tp.stable_id
    join SYS.ISYSUSER as u3 on u3.user_id = tab1.creator
    join SYS.ISYSTAB as tab2 on tab2.table_id = tp.stable_id
    join SYS.ISYSUSER as u4 on u4.user_id = tab2.creator
```

另请参见

- “SYSTABLEPERM 系统视图”一节第 977 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSTAB 系统视图”一节第 973 页

SYSTRIGGERS 统一视图

SYSTRIGGERS 视图的每一行都描述数据库中的一个触发器。该视图还包含自动为具有参照触发操作（例如 ON DELETE CASCADE）的外键定义创建的触发器。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSTRIGGERS"( owner,
  trigname,tname,event,trigtime,trigdefn )
  as select u.user_name,trig.trigger_name,tab.table_name,
    if trig.event = 'I' then 'INSERT'
    else if trig.event = 'U' then 'UPDATE'
      else if trig.event = 'C' then 'UPDATE'
        else if trig.event = 'D' then 'DELETE'
          else if trig.event = 'A' then 'INSERT,DELETE'
            else if trig.event = 'B' then 'INSERT,UPDATE'
              else if trig.event = 'E' then 'DELETE,UPDATE'
                else 'INSERT,DELETE,UPDATE'
                endif
              endif
            endif
          endif
        endif
      endif
    endif,if trig.trigger_time = 'B' or trig.trigger_time = 'P' then 'BEFORE'
    else if trig.trigger_time = 'A' or trig.trigger_time = 'S' then 'AFTER'
      else if trig.trigger_time = 'R' then 'RESOLVE'
        else 'INSTEAD OF'
        endif
      endif
    endif,trig.trigger_defn
    from SYS.ISYSTRIGGER as trig
      join SYS.ISYSTAB as tab on(tab.table_id = trig.table_id)
      join SYS.ISYSUSER as u on u.user_id = tab.creator where
    trig.foreign_table_id is null
```


另请参见

- “SYSTRIGGER 系统视图”一节第 981 页
- “SYSTAB 系统视图”一节第 973 页
- “SYSUSER 系统视图”一节第 983 页

SYSUSEROPTIONS 统一视图

SYSUSEROPTIONS 视图包含了对每个用户都有效的选项设置。如果用户没有某个选项的设置，该视图将显示该选项的公共设置。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSUSEROPTIONS"( user_name,
  "option",setting )
  as select u.user_name,
    o."option",
    isnull((select s.setting
      from SYS.ISYSOPTION as s
      where s.user_id = u.user_id
      and s."option" = o."option"),
    o.setting)
    from SYS.SYSOPTIONS as o,SYS.ISYSUSER as u
    where o.user_name = 'PUBLIC'
```

另请参见

- “SYSOPTIONS 统一视图”一节第 994 页
- “SYSUSER 系统视图”一节第 983 页

SYSVIEWS 统一视图

SYSVIEWS 视图的每一行都描述一个视图，包括其视图定义。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSVIEWS"( vcreator,
  viewname,viewtext )
  as select u.user_name,t.table_name,v.view_def
    from SYS.ISYSTAB as t
      join SYS.ISYSVIEW as v on(t.object_id = v.view_object_id)
      join SYS.ISYSUSER as u on(u.user_id = t.creator)
```

另请参见

- “SYSTAB 系统视图”一节第 973 页
- “SYSVIEW 系统视图”一节第 986 页
- “SYSUSER 系统视图”一节第 983 页

兼容性视图

兼容性视图是为了与 SQL Anywhere 10.0.0 之前的版本兼容而提供的视图。建议您尽可能使用系统视图和统一视图，因为在将来的版本中可能会减小对某些兼容性视图的支持。

SYSCOLLATION 兼容性视图（不建议使用）

SYSCOLLATION 兼容性视图包含数据库的归类序列信息。它可通过内置函数来获得且不是保存在目录中。以下是该视图的定义：

```
ALTER VIEW "SYS"."SYSCOLLATION"
  as select 1 as collation_id,
    DB_PROPERTY('Collation') as collation_label,
    DB_EXTENDED_PROPERTY('Collation','Description') as collation_name,
    cast(DB_EXTENDED_PROPERTY('Collation','LegacyData') as binary(1280)) as
collation_order;
```

另请参见

- “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “DB_PROPERTY 函数 [System]”一节第 177 页
- “DB_EXTENDED_PROPERTY 函数 [System]”一节第 173 页

SYSCOLLATIONMAPPINGS 兼容性视图（不建议使用）

SYSCOLLATIONMAPPINGS 兼容性视图只有一行，其中包含数据库归类映射。它可通过内置函数来获得且不是保存在目录中。以下是该视图的定义：

```
ALTER VIEW "SYS"."SYSCOLLATIONMAPPINGS"
  as select DB_PROPERTY('Collation') as collation_label,
    DB_EXTENDED_PROPERTY('Collation','Description') as collation_name,
    DB_PROPERTY('Charset') as cs_label,
    DB_EXTENDED_PROPERTY('Collation','ASESensitiveSortOrder') as
so_case_label,
    DB_EXTENDED_PROPERTY('Collation','ASEInsensitiveSortOrder') as
so_caseless_label,
    DB_EXTENDED_PROPERTY('Charset','java') as jdk_label;
```

另请参见

- “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “DB_PROPERTY 函数 [System]”一节第 177 页
- “DB_EXTENDED_PROPERTY 函数 [System]”一节第 173 页

SYSCOLUMN 兼容性视图（不建议使用）

提供 SYSCOLUMN 视图的目的是为了与提供 SYSCOLUMN 系统表的旧版本 SQL Anywhere 兼容。但以前的 SYSCOLUMN 表已被 ISYSTABCOL 系统表所取代，并且您应当使用与该系统表对应的“SYSTABCOL 系统视图”一节第 976 页。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSCOLUMN"
  as select b.table_id,
    b.column_id,
    if c.sequence is null then 'N' else 'Y' endif as pkey,
    b.domain_id,
    b.nulls,
    b.width,
    b.scale,
    b.object_id,
    b.max_identity,
    b.column_name,
    r.remarks,
    b."default",
    b.user_type,
    b.column_type
    from SYS.ISYSTABCOL as b
      left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
      left outer join SYS.ISYSIDXCOL as c on(b.table_id = c.table_id
      and b.column_id = c.column_id and c.index_id = 0);
```

另请参见

- “SYSTABCOL 系统视图”一节第 976 页
- “SYSREMARK 系统视图”一节第 965 页
- “SYSIDXCOL 系统视图”一节第 953 页

SYSFILE 兼容性视图（不建议使用）

SYSFILE 系统视图中的每一行都描述数据库的一个 dbspace。每个数据库都由一个或多个 dbspace 组成；每个 dbspace 都对应一个操作系统文件。

SQL Anywhere 为主数据库文件、临时文件、事务日志文件和事务日志镜像文件自动创建 dbspace。有关事务日志和事务日志镜像 dbspace 的信息未出现在 SYSFILE 系统视图中。请参见“预定义 dbspace”一节 《SQL Anywhere 服务器 - 数据库管理》。

```
ALTER VIEW "SYS"."SYSFILE"
  as select b.dbfile_id as file_id,
    if b.dbspace_id = 0 and b.dbfile_id = 0 then
      db_property('File')
    else
      if b.dbspace_id = 15 and b.dbfile_id = 15 then
        db_property('TempFileName')
      else
        b.file_name
      endif
    endif as file_name,
    a.dbspace_name,
    a.store_type,
    b.lob_map,
    b.dbspace_id
    from SYS.ISYSDBSPACE as a
      join SYS.ISYSDBFILE as b on(a.dbspace_id = b.dbspace_id);
```

SYSFKCOL 兼容性视图（不建议使用）

SYSFKCOL 的每一行都描述关系中外表的外列与主表的主列之间的关联。不建议使用该视图；请改用 SYSIDX 和 SYSIDXCOL 系统视图。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSFKCOL"
  as select a.table_id as foreign_table_id,
    a.index_id as foreign_key_id,
    a.column_id as foreign_column_id,
    a.primary_column_id
    from SYS.ISYSIDXCOL as a
      ,SYS.ISYSIDX as b
    where a.table_id = b.table_id
    and a.index_id = b.index_id
    and b.index_category = 2;
```

另请参见

- “SYSIDX 系统视图”一节第 952 页
- “SYSIDXCOL 系统视图”一节第 953 页

SYSFOREIGNKEY 兼容性视图（不建议使用）

提供 SYSFOREIGNKEY 视图的目的是为了与提供 SYSFOREIGNKEY 系统表的旧版本 SQL Anywhere 兼容。但以前的 SYSFOREIGNKEY 系统表已被 ISYSFKEY 系统表所取代，并且您应当使用与该系统表对应的“SYSFKEY 系统视图”一节第 949 页。

外键是两个表（外表和主表）之间的关系。每个外键都由 SYSFOREIGNKEY 中的一行和 SYSFKCOL 的一行或多行定义。SYSFOREIGNKEY 包含有关外键的一般信息，而 SYSFKCOL 标识外键中的列并将外键中每个列与主表的主键中的一列相关联。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSFOREIGNKEY"
  as select b.foreign_table_id,
    b.foreign_index_id as foreign_key_id,
    a.object_id,
    b.primary_table_id,
    p.root,
    b.check_on_commit,
    b.nulls,
    a.index_name as role,
    r.remarks,
    b.primary_index_id,
    a.not_enforced as fk_not_enforced,
    10 as hash_limit
    from(SYS.ISYSIDX as a left outer join SYS.ISYSPHYSIDX as p on(a.table_id
= p.table_id and a.phys_index_id = p.phys_index_id))
      left outer join SYS.ISYSREMARK as r on(a.object_id = r.object_id)
      ,SYS.ISYSFKEY as b
    where a.table_id = b.foreign_table_id
    and a.index_id = b.foreign_index_id;
```


另请参见

- “SYSIDX 系统视图”一节第 952 页
- “SYSPHYSIDX 系统视图”一节第 960 页
- “SYSREMARK 系统视图”一节第 965 页
- “SYSFKEY 系统视图”一节第 949 页

SYSINDEX 兼容性视图（不建议使用）

提供 SYSINDEX 视图的目的是为了与提供 SYSINDEX 系统表的旧版本 SQL Anywhere 兼容。但以前的 SYSINDEX 系统表已被 ISYSIDX 系统表所取代，并且您应当使用与该系统表对应的“SYSIDX 系统视图”一节第 952 页。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSINDEX"
  as select b.table_id,
    b.index_id,
    b.object_id,
    p.root,
    b.dbspace_id,
    case b."unique"
    when 1 then 'Y'
    when 2 then 'U'
    when 3 then 'M'
    when 4 then 'N'
    else 'I'
    end as "unique",
    t.creator,
    b.index_name,
    r.remarks,
    10 as hash_limit,
    b.dbspace_id as file_id
    from(SYS.ISYSIDX as b left outer join SYS.ISYSPHYSIDX as p on(b.table_id
= p.table_id and b.phys_index_id = p.phys_index_id))
      left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
      ,SYS.ISYSTAB as t
    where t.table_id = b.table_id
    and b.index_category = 3;
```

另请参见

- “SYSIDX 系统视图”一节第 952 页
- “SYSPHYSIDX 系统视图”一节第 960 页
- “SYSTABLE 兼容性视图（不建议使用）”一节第 1008 页
- “SYSREMARK 系统视图”一节第 965 页

SYSINFO 兼容性视图（不建议使用）

SYSINFO 视图指出了创建数据库时定义的数据库特性。它始终只有一行。该视图可通过内置函数来获得且不是保存在目录中。以下是 SYSINFO 视图的定义：

```
ALTER VIEW "SYS"."SYSINFO"( page_size,
  encryption,
  blank_padding,
  case_sensitivity,
  default_collation,
  database_version )
  as select db_property('PageSize'),
    if db_property('Encryption') <> 'None' then 'Y' else 'N' endif,
    if db_property('BlankPadding') = 'On' then 'Y' else 'N' endif,
    if db_property('CaseSensitive') = 'On' then 'Y' else 'N' endif,
    db_property('Collation'),
    null;
```

另请参见

- “数据库属性”一节 《SQL Anywhere 服务器 - 数据库管理》
- “DB_PROPERTY 函数 [System]”一节第 177 页
- “DB_EXTENDED_PROPERTY 函数 [System]”一节第 173 页

SYSIXCOL 兼容性视图（不建议使用）

提供 SYSIXCOL 视图的目的是为了与提供 SYSIXCOL 系统表的旧版本 SQL Anywhere 兼容。但 SYSIXCOL 系统表已被 ISYSIDXCOL 系统表所取代，与后者对应的是 SYSIDXCOL 系统视图。您应改用“SYSIDXCOL 系统视图”一节第 953 页。

SYSIXCOL 的每一行都描述索引中的一列。在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSIXCOL"
  as select a.table_id,
    a.index_id,
    a.sequence,
    a.column_id,
    a."order"
    from SYS.ISYSIDXCOL as a
      ,SYS.ISYSIDX as b
    where a.table_id = b.table_id
    and a.index_id = b.index_id
    and b.index_category = 3;
```

另请参见

- “SYSIDX 系统视图”一节第 952 页
- “SYSIDXCOL 系统视图”一节第 953 页

SYSTABLE 兼容性视图（不建议使用）

提供 SYSTABLE 视图的目的是为了与提供 SYSTABLE 系统表的旧版本 SQL Anywhere 兼容。但以前的 SYSTABLE 系统表已被 ISYSTAB 系统表所取代，并且您应当使用与该系统表对应的“SYSTAB 系统视图”一节第 973 页。

SYSTABLE 视图的每一行都描述数据库中的一个表。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSTABLE"
  as select b.table_id,
    b.file_id,
    b.count,
    0 as first_page,
    b.commit_action as last_page,
    COALESCE(ph.root,0) as primary_root,
    b.creator,
    0 as first_ext_page,
    0 as last_ext_page,
    b.table_page_count,
    b.ext_page_count,
    b.object_id,
    b.table_name,
    b.table_type_str as table_type,
    v.view_def,
    r.remarks,
    b.replicate,
    p.existing_obj,
    p.remote_location,
    'T' as remote_objtype,
    p.srvid,
    case b.server_type
    when 1 then 'SA'
    when 2 then 'IQ'
    when 3 then 'OMNI'
    else 'INVALID'
    end as server_type,
    10 as primary_hash_limit,
    0 as page_map_start,
    s.source,
    b."encrypted"
    from SYS.SYSTAB as b
      left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id)
      left outer join SYS.ISYSSOURCE as s on(b.object_id = s.object_id)
      left outer join SYS.ISYSVIEW as v on(b.object_id = v.view_object_id)
      left outer join SYS.ISYSPROXYTAB as p on(b.object_id =
p.table_object_id)
      left outer join(SYS.ISYSIDX as i left outer join SYS.ISYSPHYSIDX as ph
on(i.table_id = ph.table_id
        and i.phys_index_id = ph.phys_index_id)) on(b.table_id = i.table_id
        and i.index_category = 1 and i.index_id = 0);
```

另请参见

- “SYSTAB 系统视图”一节第 973 页
- “SYSREMARK 系统视图”一节第 965 页
- “SYSSOURCE 系统视图”一节第 970 页
- “SYSVIEW 系统视图”一节第 986 页
- “SYSPROXYTAB 系统视图”一节第 964 页
- “SYSIDX 系统视图”一节第 952 页
- “SYSPHYSIDX 系统视图”一节第 960 页

SYSUSERAUTH 兼容性视图（不建议使用）

提供 SYSUSERAUTH 视图的目的是为了与旧版本的 SQL Anywhere 兼容。改用 SYSUSERAUTHORITY 系统视图。请参见“SYSUSERAUTHORITY 系统视图”一节第 984 页。

SYSUSERAUTH 视图的每一行描述一个用户，但不公开其 user_id。而是用用户名来标识每个用户。由于该视图显示口令，因此该视图没有 PUBLIC 选择权限。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSUSERAUTH"( name,
  password,resourceauth,dbaauth,scheduleauth,user_group )
  as select SYSUSERPERM.user_name,
            SYSUSERPERM.password,
            SYSUSERPERM.resourceauth,
            SYSUSERPERM.dbaauth,
            SYSUSERPERM.scheduleauth,
            SYSUSERPERM.user_group
    from SYS.SYSUSERPERM;
```

另请参见

- “SYSUSERPERM 兼容性视图（不建议使用）”一节第 1011 页

SYSUSERLIST 兼容性视图（不建议使用）

提供 SYSUSERAUTH 视图的目的是为了与旧版本的 SQL Anywhere 兼容。

SYSUSERLIST 视图的每一行都描述一个用户，但不公开其 user_id 和口令。每个用户都由其用户名来标识。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSUSERLIST"(
      name,
      resourceauth,
      dbaauth,
      scheduleauth,user_group )
  as select SYSUSERPERM.user_name,
            SYSUSERPERM.resourceauth,
            SYSUSERPERM.dbaauth,
            SYSUSERPERM.scheduleauth,
            SYSUSERPERM.user_group
    from SYS.SYSUSERPERM;
```

另请参见

- “SYSUSERPERM 兼容性视图（不建议使用）”一节第 1011 页

SYSUSERPERM 兼容性视图（不建议使用）

不建议使用该视图，因为它只显示以前版本中可用的特权和权限。应更改您的应用程序改用 SYSUSERAUTHORITY 系统视图。

SYSUSERPERM 视图的每一行都描述一个用户 ID。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSUSERPERM"
  as select b.user_id,
    b.object_id,
    b.user_name,
    b.password,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth
= 'RESOURCE') then
      'Y' else 'N' endif as resourceauth,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth
= 'DBA') then
      'Y' else 'N' endif as dbaauth,
    'N' as scheduleauth,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth
= 'PUBLISH') then
      'Y' else 'N' endif as publishauth,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth
= 'REMOTE DBA') then
      'Y' else 'N' endif as remotedbaauth,
    if exists(select * from SYS.ISYSUSERAUTHORITY
      where ISYSUSERAUTHORITY.user_id = b.user_id and ISYSUSERAUTHORITY.auth
= 'GROUP') then
      'Y' else 'N' endif as user_group,
    r.remarks
    from SYS.ISYSUSER as b
      left outer join SYS.ISYSREMARK as r on(b.object_id = r.object_id);
```

另请参见

- “SYSUSERAUTHORITY 系统视图”一节第 984 页
- “SYSUSER 系统视图”一节第 983 页
- “SYSREMARK 系统视图”一节第 965 页

SYSUSERPERMS 兼容性视图（不建议使用）

不建议使用该视图，因为它只显示以前版本中可用的特权和权限。应更改您的应用程序改用 SYSUSERAUTHORITY 系统视图。

与 SYSUSERPERM 视图相似，SYSUSERPERMS 视图的每一行都描述一个用户 ID。但不包括口令信息。所有用户都被允许从该视图中读取。

在以下的 SQL 语句中提供了组成该视图的表和列。要了解有关特定表或列的详细信息，请使用该视图定义下所提供的链接。

```
ALTER VIEW "SYS"."SYSUSERPERMS"
  as select SYSUSERPERM.user_id,
   SYSUSERPERM.user_name,
   SYSUSERPERM.resourceauth,
   SYSUSERPERM.dbaauth,
   SYSUSERPERM.scheduleauth,
   SYSUSERPERM.user_group,
   SYSUSERPERM.publishauth,
   SYSUSERPERM.remotedbaauth,
   SYSUSERPERM.remarks
    from SYS.SYSUSERPERM;
```

另请参见

- “SYSUSERPERM 兼容性视图（不建议使用）”一节第 1011 页
- “SYSUSERAUTHORITY 系统视图”一节第 984 页

Transact-SQL 兼容性视图

Adaptive Server Enterprise 和 SQL Anywhere 的系统目录是有区别的。Adaptive Server Enterprise 系统表和视图由用户 dbo 所拥有，而且一部分存在于 master 数据库中，一部分存在于 sybsecurity 数据库中，还有一部分存在于各个单独的数据库中。SQL Anywhere 系统表和视图由特殊用户 SYS 所拥有，并且分别存在于每个数据库中。

为帮助准备兼容的应用程序，SQL Anywhere 提供了由特殊用户 dbo 所拥有以下一组视图，这些视图对应 Adaptive Server Enterprise 的相应视图。当结构上的不同使具体某个 Adaptive Server Enterprise 表或视图的内容在 SQL Anywhere 环境中没有意义时，该视图为空，仅包含列名称和数据类型。

视图名称	说明
syscolumns	表或视图中的每一列都占一行，过程中的每个参数都占一行
syscomments	每个视图、规则、缺省值、触发器和过程都占一行或多行，以给出 SQL 定义语句
sysindexes	每个聚簇索引或非聚簇索引都占一行，每个没有索引的表都占一行，每个包含文本或图像数据的表另外占一行。
sysobjects	每个表、视图、过程、规则、触发器、缺省值、日志或临时对象（仅在 tempdb 中）都占一行
systypes	每个系统提供或用户定义的数据类型都占一行
sysusers	数据库允许的每个用户都占一行
syslogins	每个有效用户帐户都占一行

术语表

术语表

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。在版本 10.0.0 中，Adaptive Server Anywhere 更名为 SQL Anywhere 服务器，SQL Anywhere Studio 更名为 SQL Anywhere。

另请参见： “SQL Anywhere”一节第 1032 页

包

Java 中相关类的集合。

被引用对象

一种对象（如表），该对象在另一个对象（如视图）的定义中被直接引用。

另请参见： “主键”一节第 1042 页

编码

也称作字符编码，编码是一种方法，通过该方法可以将字符集中的每个字符映射到一个或多个字节的信息，这些信息通常以十六进制数字表示。编码的一个例子是 UTF-8。

另请参见：

- “字符集”一节第 1042 页
- “代码页”一节第 1017 页
- “归类”一节第 1021 页

标识符

用于引用数据库对象（如表或列）的字符串。标识符可以包含 A 到 Z、a 到 z、0 到 9、下划线 (_)、at 符号 (@)、数字符号 (#) 或美元符号 ($) 中的任何字符。

并发

同时执行两个或更多个独立并且可能存在竞争关系的进程。SQL Anywhere 会自动使用锁定来隔离事务，并确保每个并发应用程序看到的数据集均一致。

另请参见：

- “事务”一节第 1029 页
- “隔离级别”一节第 1020 页

参考数据库

MobiLink 中一种用于 UltraLite 客户端开发的 SQL Anywhere 数据库。在开发过程中，可以将一个 SQL Anywhere 数据库同时作为参考数据库和统一数据库使用。通过其它产品建立的数据库无法用作参考数据库。

参照完整性

遵守数据一致性控制规则（具体而言，不同表中主键值与外键值之间的关系）。若要实现参照完整性，每个外键中的值必须与被引用表中行的主键值相符。

另请参见：

- “主键”一节第 1042 页
- “外键”一节第 1034 页

策略

QAnywhere 中指定应在何时进行消息传输的方式。

插件模块

Sybase Central 中一种用于访问和管理产品的方法。当您安装相应的产品时，插件通常会自动安装并注册 Sybase Central。通常，插件在 Sybase Central 主窗口中作为顶级容器出现，并且使用产品本身的名称，如 SQL Anywhere。

另请参见： “Sybase Central”一节第 1033 页

查询

一条或一组 SQL 语句，用于访问和/或操作数据库中的数据。

另请参见： “SQL”一节第 1032 页

冲突解决

在 MobiLink 中，冲突解决是指一种逻辑，它指定当两个用户修改不同远程数据库上同一行时的处理方法。

重定向器

一种 Web 服务器插件，用于为客户端与 MobiLink 服务器之间的请求和响应选择发送路径。此插件还实现了负荷平衡和故障转移机制。

抽取

SQL Remote 复制中从统一数据库卸载相应结构和数据的行为。此信息用于初始化远程数据库。

另请参见： “复制”一节第 1019 页

触发器

一种特殊形式的存储过程，用户运行修改数据的查询时会自动执行该存储过程。

另请参见：

- “行级触发器”一节第 1021 页
- “语句级触发器”一节第 1039 页
- “完整性”一节第 1035 页

传输规则

QAnywhere 中用于确定何时进行消息传输、传输哪些消息以及应在何时删除消息的逻辑。

窗口

作为分析功能执行对象的行组。一个窗口可以包含一行、多行或所有行的数据，这些数据已根据窗口定义中提供的分组规格进行了分区。窗口会进行移动，以包括为输入中的当前行执行计算所需的行数或行范围。窗口结构的主要优点是，不需要执行附加查询就可以有机会对结果进行分组和分析。

创建者 ID

UltraLite Palm OS 应用程序中一种在创建应用程序时指派的 ID。

存储过程

存储过程是数据库中存储的一组 SQL 指令，用于在数据库服务器上执行一组操作或查询。

代理表

一种本地表，它所包含的元数据可以像访问本地表一样访问远程数据库服务器上的表。

另请参见： “元数据”一节第 1040 页

代理 ID

另请参见： “客户端消息存储库 ID”一节第 1025 页

代码页

代码页是一种将字符集的字符映射到数字表示的编码，数字表示通常是 0 到 255 之间的一个整数。例如，Windows 代码页 1252 就是一个代码页。就本文档而言，代码页和编码这两个术语可以互换。

另请参见：

- “字符集”一节第 1042 页
- “编码”一节第 1015 页
- “归类”一节第 1021 页

DBA 权限

使用户能够在数据库中执行管理活动的权限级别。DBA 用户在缺省情况下具有 DBA 权限。

另请参见： “数据库管理员 (DBA)”一节第 1031 页

dbspace

用于创建更多数据存储空间的附加数据库文件。一个数据库可以包含在最多 13 个独立的文件（一个初始文件和 12 个 dbspace）中。每个表及其索引必须包含在单个数据库文件中。SQL 命令 CREATE DBSPACE 可将新文件添加到数据库中。

另请参见： “数据库文件”一节第 1032 页

动态 SQL

执行前由程序以编程方式生成的 SQL。UltraLite 动态 SQL 是一种专用于小型设备的 SQL 变体。

对象树

Sybase Central 中数据库对象的层次。对象树的顶层显示您的 Sybase Central 版本所支持的全部产品。每种产品展开后会显示其自己的对象子树。

另请参见： “Sybase Central”一节第 1033 页

EBF

快速错误修正软件。快速错误修正软件是含有一个或多个错误修正软件的软件子集。错误修正软件列在更新程序的发行说明中。错误修正软件更新可能只适用于具有相同版本号的已安装软件。已对该软件执行了一些测试，但该软件尚未进行完全测试。除非您自己已验证了软件的适用性，否则不要随应用程序分发这些文件。

发布

MobiLink 或 SQL Remote 中一种用于标识将要同步的数据的数据库对象。在 MobiLink 中，发布仅存在于客户端。一个发布包括多个项目。SQL Remote 用户可以通过预订发布来接收发布。MobiLink 用户可以通过创建发布的同步预订来同步发布。

另请参见：

- “复制”一节第 1019 页
- “项目”一节第 1037 页
- “发布更新”一节第 1018 页

发布更新

SQL Remote 复制中对一个数据库中的一个或多个发布所做更改的列表。发布更新将作为复制消息的一部分定期发送到远程数据库。

另请参见：

- “复制”一节第 1019 页
- “发布”一节第 1018 页

发布者

SQL Remote 复制中数据库内可以与其它复制数据库交换复制消息的单个用户。

另请参见： “复制”一节第 1019 页。

FILE

SQL Remote 复制中一种使用共享文件来交换复制消息的消息系统。它对测试以及在没有显式消息传送系统的情况下进行的安装很有用。

另请参见“复制”一节第 1019 页。

分析树

查询的代数表示。

服务

在 Windows 操作系统上，服务是在运行应用程序的用户 ID 未登录时的应用程序运行方式。

服务器管理请求

一种 QAnywhere 消息，其格式设置为 XML 并发送到 QAnywhere 系统队列，作为一种管理服务器消息存储库或监控 QAnywhere 应用程序的方法。

服务器启动的同步

一种从 MobiLink 服务器启动 MobiLink 同步的方式。

服务器消息存储库

QAnywhere 中在消息传输到客户端消息存储库或 JMS 系统之前服务器上用于临时存储消息的关系数据库。消息通过服务器消息存储库在各客户端之间进行交换。

复制

在物理上不相同的数据库之间共享数据。Sybase 有三种复制技术：MobiLink、SQL Remote 和复制服务器。

复制代理

请参见： “LTM”一节第 1026 页

复制服务器

Sybase 的一种基于连接的复制技术，用于与 SQL Anywhere 和 Adaptive Server Enterprise 一起使用。它专用于在一些数据库之间进行接近实时的复制。

另请参见： “LTM”一节第 1026 页

复制频率

SQL Remote 复制中一项针对每个远程用户的设置，它决定发布者的消息代理向该远程用户发送复制消息的频率应为多少。

另请参见： “复制”一节第 1019 页。

复制消息

SQL Remote 或复制服务器中一种在发布数据库与预订数据库之间发送的通信。消息包含复制系统所需的数据、直通语句及信息。

另请参见：

- “复制”一节第 1019 页
- “发布更新”一节第 1018 页

隔离级别

一个事务中的操作对其它并发事务中的操作的可见程度。隔离级别有四级，编号依次为 0 至 3。第 3 级提供最高级别的隔离。级别 0 为缺省设置。SQL Anywhere 还支持以下三个快照隔离级别：快照、语句快照和只读语句快照。

另请参见： “快照隔离”一节第 1025 页

个人服务器

与客户端应用程序在同一台计算机上运行的数据库服务器。个人数据库服务器通常由单个用户在一台计算机上使用，但它可以支持来自该用户的几个并发连接。

工作表

一种内部存储区域，用于在查询优化过程中存储中间结果。

故障切换

在活动服务器、系统或网络出现故障或意外终止时切换到冗余或备用的服务器、系统或网络。故障转移会自动进行。

关系数据库管理系统 (RDBMS)

一种以相关表的形式存储数据的数据库管理系统。

另请参见： “数据库管理系统 (DBMS)”一节第 1031 页

规范化

对数据库模式的改进，目的在于按照基于关系数据库理论的规则消除冗余并改善组织。

归类

定义数据库中文本属性的字符集与排序顺序的组合。对于 SQL Anywhere 数据库，缺省归类取决于运行服务器时所使用的操作系统和语言；例如，英语 Windows 系统上的缺省归类为 1252LATIN1。归类（也称作归类序列）用于对字符串进行比较和排序。

另请参见：

- “字符集”一节第 1042 页
- “代码页”一节第 1017 页
- “编码”一节第 1015 页

行级触发器

每更改一行即执行一次的触发器。

另请参见：

- “触发器”一节第 1017 页
- “语句级触发器”一节第 1039 页

回退日志

对在每个未提交的事务执行过程中所做更改的记录。当收到 ROLLBACK 请求或者系统出现故障时，未提交的事务会从数据库中回退，将数据库返回其原先的状态。每个事务都有一个单独的回退日志，事务完成时日志会被删除。

另请参见： “事务”一节第 1029 页

iAnywhere JDBC 驱动程序

iAnywhere JDBC 驱动程序提供了一个 JDBC 驱动程序，与纯 Java jConnect JDBC 驱动程序相比，该驱动程序拥有一些性能优势和功能优点，但它不是纯 Java 解决方案。建议在大多数情况下使用 iAnywhere JDBC 驱动程序。

另请参见：

- “JDBC”一节第 1022 页
- “jConnect”一节第 1022 页

InfoMaker

一种报告和数据维护工具，它用于创建复杂的表格、报告、图形、交叉表和表，并创建将这些报告用作构件块的应用程序。

Interactive SQL

一种 SQL Anywhere 应用程序，用于查询和更改数据库中的数据以及修改数据库的结构。Interactive SQL 不但提供了一个用于输入 SQL 语句的窗格，还提供了一些用于返回有关查询处理过程的信息和结果集的窗格。

JAR 文件

Java 档案文件。一种压缩的文件格式，由一个或多个用于 Java 应用程序的包的集合组成。它将安装和运行 Java 程序所需的全部资源都放在一个压缩文件中。

Java 类

Java 中的主要代码结构单元。它是组合在一起的过程和变量的集合，将过程和变量组合在一起的原因是它们都与某个特定的可识别类别有关。

jConnect

JavaSoft JDBC 标准的 Java 实现。它为 Java 开发人员提供多层和异类环境中的本地数据库访问。但在大多数情况下，iAnywhere JDBC 驱动程序是首选的 JDBC 驱动程序。

另请参见：

- “JDBC”一节第 1022 页
- “iAnywhere JDBC 驱动程序”一节第 1021 页

JDBC

Java 数据库连接。一种 SQL 语言编程接口，它允许 Java 应用程序访问关系数据。首选的 JDBC 驱动程序是 iAnywhere JDBC 驱动程序。

另请参见：

- “jConnect”一节第 1022 页
- “iAnywhere JDBC 驱动程序”一节第 1021 页

基表

永久性的数据表。有时为区别于临时表和视图，会将这种表称作**基表**。

另请参见：

- “临时表”一节第 1025 页
- “视图”一节第 1029 页

基于会话的同步

一种同步类型，这种同步会使数据表示在统一数据库和远程数据库都一致。MobiLink 基于会话。

基于脚本的上载

MobiLink 中一种将上载过程自定义为使用日志文件的替代方法的方式。

基于 SQL 的同步

MobiLink 中一种使用 MobiLink 事件将表数据与支持 MobiLink 的统一数据库进行同步的方式。对于基于 SQL 的同步，可以直接使用 SQL，也可以使用面向 Java 和 .NET 平台的 MobiLink 服务器 API 返回 SQL。

基于文件的下载

在 MobiLink 中同步数据的一种方式，其中下载以文件的方式进行分发，从而支持脱机分发同步更改。

集成登录

一种登录功能，它允许将同一个用户 ID 和口令用于操作系统登录、网络登录和数据库连接。

监听器

一个程序 (dblsn)，用于 MobiLink 服务器启动的同步。监听器安装在远程设备上，它们被配置为在接收到来自通告程序的信息时启动针对设备的操作。

另请参见： “服务器启动的同步”一节第 1019 页

检查点

将对数据库的所有更改都保存到数据库文件中的时间点。在其它时间，所提交的更改仅保存到事务日志中。

检查约束

对列或列集强制实施指定条件的一种限制。

另请参见：

- “约束”一节第 1041 页
- “外键约束”一节第 1035 页
- “主键约束”一节第 1042 页
- “唯一约束”一节第 1036 页

脚本

MobiLink 中为处理 MobiLink 事件而编写的代码。脚本通过编程方式控制数据交换，以满足业务需要。

另请参见： “事件模型”一节第 1029 页

脚本版本

MobiLink 中为创建同步而一起应用的一组同步脚本。

校验

测试数据库、表或索引是否受到特定类型的文件损坏。

校验和

随数据库页本身一起记录的计算出的数据库页位数。校验和能够确保数据库页写入磁盘时位数相符，因此数据库管理系统可以通过它来验证数据库页的完整性。如果计数相符，即认为数据库页已成功写入。

镜像日志

另请参见： “事务日志镜像”一节第 1030 页

角色

概念性数据库建模中从一个角度描述某种关系的动词或短语。您可以用两个角色来描述每种关系。例如，"包含" 和 "隶属于" 便是角色。

角色名

外键的名称。由于它命名外表和主表之间的关系，因此称作角色名。缺省情况下，角色名就是表名，除非其它外键已经使用该名称（在这种情况下，缺省的角色名是表名后接一个三位的唯一数字）。也可以自己创建角色名。

另请参见： “外键”一节第 1034 页

局部临时表

一种临时表，仅在复合语句执行期间或连接结束之前存在。当您只需要将数据集装载一次时，局部临时表非常有用。缺省情况下，行会在提交时被删除。

另请参见：

- “临时表”一节第 1025 页
- “全局临时表”一节第 1028 页

客户端/服务器

一种软件体系结构，在这种体系结构中，一个应用程序（客户端）从另一个应用程序（服务器）获取信息并向该应用程序发送信息。这两个应用程序常位于通过网络连接的不同计算机上。

客户端消息存储库

QAnywhere 中一种用于在远程设备上存储消息的 SQL Anywhere 数据库。

客户端消息存储库 ID

QAnywhere 中一种对客户端消息存储库进行唯一标识的 MobiLink 远程 ID。

快照隔离

一种为发出读请求的事务返回数据的已提交版本的隔离级别。SQL Anywhere 提供了以下三种快照隔离级别：快照、语句快照和只读语句快照。使用快照隔离时，读操作不会阻塞写操作。

另请参见： “隔离级别”一节第 1020 页

连接

关系系统中的一种基本操作，它通过比较指定列中的值将两个或更多个表中的行链接在一起。

连接 ID

用于标识客户端应用程序与数据库之间给定连接的唯一编号。可以使用以下 SQL 语句来确定当前连接 ID：

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

连接类型

SQL Anywhere 提供了四种类型的连接：交叉连接、键连接、自然连接和使用 ON 子句的连接。

另请参见： “连接”一节第 1025 页

连接配置

连接到数据库所需的一组参数，如用户名、口令和服务器名称，它们在存储后即可方便地使用。

连接启动的同步

一种 MobiLink 服务器启动的同步，在这种同步下，连接发生变化时会启动同步。

另请参见： “服务器启动的同步”一节第 1019 页

连接条件

一种影响连接结果的限制。您可以通过紧跟在连接语句的后面插入 ON 子句或 WHERE 子句来指定连接条件。对于自然连接和关键连接，SQL Anywhere 会生成连接条件。

另请参见：

- “连接”一节第 1025 页
- “生成的连接条件”一节第 1030 页

临时表

为临时存储数据而创建的表。有两种类型：全局临时表和局部临时表。

另请参见：

- “局部临时表”一节第 1024 页
- “全局临时表”一节第 1028 页

LTM

日志传送管理器（Log Transfer Manager，简称 LTM）也称作复制代理。LTM 是一个与 Replication Server 一起使用的程序，它读取数据库事务日志并将提交的更改发送到 Sybase 复制服务器。

请参见： “复制服务器”一节第 1020 页

轮询

在 MobiLink 服务器启动的同步中，轻量级轮询器（例如 MobiLink 监听器）从通告程序请求推式通知的方式。

另请参见： “服务器启动的同步”一节第 1019 页

逻辑索引

指向物理索引的引用（指针）。磁盘上不存储逻辑索引的索引结构。

命令文件

包含 SQL 语句的文本文件。命令文件可以手工建立，也可以通过数据库实用程序自动建立。例如，dbunload 实用程序会创建一个命令文件，其中包含重新创建给定数据库所需的 SQL 语句。

MobiLink

一种基于会话的同步技术，其设计用途是将 UltraLite 和 SQL Anywhere 远程数据库与统一数据库同步。

另请参见：

- “统一数据库”一节第 1033 页
- “同步”一节第 1033 页
- “UltraLite”一节第 1034 页

MobiLink 服务器

运行 MobiLink 同步的计算机程序，即 mlsrv11。

MobiLink 监控器

一种用于监控 MobiLink 同步的图形化工具。

MobiLink 客户端

有两种 MobiLink 客户端。对于 SQL Anywhere 远程数据库，MobiLink 客户端是 dbmlsync 命令行实用程序。对于 UltraLite 远程数据库，MobiLink 客户端内置于 UltraLite 运行时库中。

MobiLink 系统表

MobiLink 同步所需的系统表。它们由 MobiLink 安装程序脚本安装到 MobiLink 统一数据库中。

MobiLink 用户

MobiLink 用户用于与 MobiLink 服务器进行连接。在远程数据库上创建 MobiLink 用户，然后在统一数据库中注册该用户。MobiLink 用户名完全独立于数据库用户名。

模式

数据库的结构，其中包括表、列和索引以及它们之间的关系。

内连接

一种连接，在这种连接中，仅当两个表都满足连接条件时行才会出现在结果集中。内连接是缺省设置。

另请参见：

- “连接”一节第 1025 页
- “外连接”一节第 1035 页

ODBC

开放式数据库连接。一种用于与数据库管理系统连接的标准 Windows 接口。ODBC 是 SQL Anywhere 所支持的几种接口之一。

ODBC 管理器

一种随 Windows 操作系统提供的 Microsoft 程序，用于设置 ODBC 数据源。

ODBC 数据源

用户要通过 ODBC 访问的数据的规范以及获取该数据时所需的信息。

PDB

Palm 数据库文件。

PowerDesigner

一种数据库建模应用程序。PowerDesigner 为设计数据库或数据仓库提供了结构化的方法。SQL Anywhere 包括 PowerDesigner 的 Physical Data Model 组件。

PowerJ

一种 Sybase 产品，用于开发 Java 应用程序。

QAnywhere

应用程序到应用程序的消息传递（包括移动设备到移动设备和移动设备与企业之间的消息传递），它使在移动或无线设备上运行的自定义程序能够与处在中央位置的服务器应用程序进行通信。

QAnywhere 代理

QAnywhere 中一种运行在客户端设备上的进程，用于监控客户端消息存储库和确定应在何时传输消息。

嵌入式 SQL

一种 C 语言程序编程接口。SQL Anywhere 嵌入式 SQL 是 ANSI 和 IBM 标准的实现。

轻量级轮询器

在 MobiLink 服务器启动的同步中，轮询来自 MobiLink 服务器的推式通知的设备应用程序。

另请参见： “服务器启动的同步”一节第 1019 页

全局临时表

一种临时表，在被显式地删除之前，其数据定义对所有用户都可见。全局临时表允许用户各自打开一个表的相同实例。缺省情况下，行在提交时被删除，并且始终是在连接结束时被删除。

另请参见：

- “临时表”一节第 1025 页
- “局部临时表”一节第 1024 页

日志文件

SQL Anywhere 所维护的事务日志。该日志文件用于确保在出现系统或介质故障时可以恢复数据库、提高数据库性能以及使用 SQL Remote 实现数据复制。

另请参见：

- “事务日志”一节第 1029 页
- “事务日志镜像”一节第 1030 页
- “完全备份”一节第 1035 页

散列

散列是一种将索引条目转化为键的索引优化。索引散列旨在通过将足够的行实际数据与其行 ID 包括在一起，以避免进行先查找行、后装载行然后再将行解出才能得出索引值的高开销操作。

上载

同步过程的一个阶段，在此阶段数据从远程数据库传送到统一数据库。

设备跟踪

在 MobiLink 服务器启动的同步中，允许使用标识设备的 MobiLink 用户名来对消息进行寻址的功能。

另请参见： “服务器启动的同步”一节第 1019 页

实例化视图

实例化视图是指已计算并已存储在磁盘上的视图。实例化视图同时具有视图的特征（使用查询说明进行定义）和表的特征（可以对其执行大多数表操作）。

另请参见：

- “基表”一节第 1022 页
- “视图”一节第 1029 页

世代号

MobiLink 中的一种机制，用于强制远程数据库先上载数据，然后再应用任何其它下载文件。

另请参见： “基于文件的下载”一节第 1023 页

事件模型

MobiLink 中组成同步的事件（如 begin_synchronization 和 download_cursor）序列。如果为事件创建了脚本，则会调用事件。

视图

一种作为对象存储在数据库中的 SELECT 语句。它使用户能够看到一个或多个表中的行子集或列子集。每当用户使用特定表或表组合的视图时，都将利用存储在这些表中的信息重新计算视图。视图对确保安全以及定制数据库信息的外观来使数据访问简单明了有帮助。

事务

组成一个逻辑工作单元的 SQL 语句序列。事务要么全部得到处理，要么根本不做处理。SQL Anywhere 支持事务处理，并内置了锁定功能，使并发事务能够访问数据库而又不损坏数据。事务要么以 COMMIT 语句结束，该语句使对数据的更改成为永久性更改；要么以 ROLLBACK 语句结束，该语句撤消在事务执行过程中所做的全部更改。

事务日志

一种按进行更改的顺序存储对数据库所做全部更改的文件。它会提高性能并支持在数据库文件损坏时恢复数据。

事务日志镜像

同时维护的事务日志文件的完全相同副本（可选）。每当数据库更改写入事务日志文件时，也会同时写入事务日志镜像文件。

镜像文件应该与事务日志保留在不同的设备上，这样在任意设备出现故障时，日志的其它副本会确保数据可以安全地恢复。

另请参见： “事务日志”一节第 1029 页

事务完整性

MobiLink 中对整个同步系统事务的有保证维护。要么同步整个事务，要么不对事务的任何部分进行同步。

生成的连接条件

一种自动生成的对连接结果的限制。有两种类型：关键和自然。指定 KEY JOIN 或指定关键字 JOIN 但不使用关键字 CROSS、NATURAL 或 ON 时，会生成关键连接。对于关键连接，所生成的连接条件取决于表之间的外键关系。指定 NATURAL JOIN 时会生成自然连接；所生成的连接条件基于两个表中的公用列名。

另请参见：

- “连接”一节第 1025 页
- “连接条件”一节第 1025 页

受保护的功能

数据库服务器启动时由 -sf 选项指定的功能，该数据库服务器上运行的任何数据库都无法使用该功能。

授权选项

一种权限级别，它允许用户向其他用户授予权限。

数据操作语言 (DML)

用于操作数据库中数据的 SQL 语句子集。DML 语句可以检索、插入、更新和删除数据库中的数据。

数据定义语言 (DDL)

用于定义数据库中数据结构的 SQL 语句子集。DDL 语句可以创建、修改和删除数据库对象（如表和用户）。

数据类型

数据的格式，如 CHAR 或 NUMERIC。在 ANSI SQL 标准中，数据类型也可以包括对大小、字符集和归类的限制。

另请参见： “域”一节第 1039 页

数据立方体

一种多维结果集，每一维都以不同的方式对相同的结果进行分组和排序。数据立方体提供了有关数据的综合性信息，如果不使用数据立方体，要获得同样的信息就必须进行自连接查询和相关子查询。数据立方体是 OLAP 功能的一部分。

数据库

通过主键和外键关联的表的集合。表包含数据库中的信息。表和键一起定义数据库的结构。数据库管理系统会访问此信息。

另请参见：

- “外键”一节第 1034 页
- “主键”一节第 1042 页
- “数据库管理系统 (DBMS)”一节第 1031 页
- “关系数据库管理系统 (RDBMS)”一节第 1020 页

数据库对象

包含或接收信息的数据库组件。表、索引、视图、过程和触发器便是数据库对象。

数据库服务器

对所有针对数据库信息的访问进行管理的计算机程序。SQL Anywhere 提供了两种类型的服务器：网络服务器和个人服务器。

数据库管理系统 (DBMS)

用于创建和使用数据库的程序的集合。

另请参见： “关系数据库管理系统 (RDBMS)”一节第 1020 页

数据库管理员 (DBA)

具有维护数据库所需权限的用户。DBA 通常负责对数据库模式的所有更改以及管理用户和组。数据库管理员角色自动内置于数据库中，其用户 ID 为 DBA，口令是 sql。

数据库连接

客户端应用程序与数据库之间的通信渠道。必须具有有效的用户 ID 和口令才能建立连接。为用户 ID 授予的特权决定了在连接过程中可以执行的操作。

数据库名称

服务器装载数据库时为数据库指定的名称。缺省数据库名是初始数据库文件的文件名（不含扩展名）。

另请参见： “数据库文件”一节第 1032 页

数据库所有者 (dbo)

一种特殊的用户，他拥有不归 SYS 所有的系统对象。

另请参见：

- “数据库管理员 (DBA)”一节第 1031 页
- “SYS”一节第 1033 页

数据库文件

数据库保存在一个或多个数据库文件中。其中一个是初始文件，后面的文件称作 dbspace。每个表（包括其索引）都必须包含在单个数据库文件中。

另请参见： “dbspace”一节第 1018 页

死锁

一组事务会进入的一种特殊状态，在该状态下这些事务都不能继续执行。

SQL

用于与关系数据库进行通信的语言。ANSI 定义了 SQL 的标准，其最新标准是 SQL-2003。SQL 的非官方全称是结构化查询语言。

SQL Anywhere

SQL Anywhere 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。SQL Anywhere 也是包含 SQL Anywhere RDBMS、UltraLite RDBMS、MobiLink 同步软件和其它组件的软件包的名称。

SQL Remote

一种基于消息的数据复制技术，用于在统一数据库与远程数据库之间进行双向复制。统一数据库和远程数据库必须是 SQL Anywhere。

SQL 语句

包含用于将指令传递给 DBMS 的 SQL 关键字的字符串。

另请参见：

- “模式”一节第 1027 页
- “SQL”一节第 1032 页
- “数据库管理系统 (DBMS)”一节第 1031 页

锁定

一种在同时执行多个事务的过程中保护数据完整性的并发控制机制。SQL Anywhere 会自动应用锁以防止两个连接同时更改同一数据，并防止其它连接读取正接受更改的数据。

您可以通过设置隔离级别来控制锁定。

另请参见：

- “隔离级别”一节第 1020 页
- “并发”一节第 1015 页
- “完整性”一节第 1035 页

索引

一组已排序的、与基表中的一个或多个列关联的键和指针。在表中一个或多个列上设置索引可以提高性能。

Sybase Central

一种数据库管理工具，通过图形用户界面提供 SQL Anywhere 数据库设置、属性和实用程序。Sybase Central 也可用于管理其它 Sybase 产品，其中包括 MobiLink。

SYS

一种拥有大多数系统对象的特殊用户。无法以 SYS 身份登录。

统一数据库

在分布式数据库环境中，是指用于存储数据主副本的数据库。出现冲突或差异时，将把统一数据库视为具有数据的主副本。

另请参见：

- “同步”一节第 1033 页
- “复制”一节第 1019 页

通信流

MobiLink 中 MobiLink 客户端与 MobiLink 服务器之间进行通信时所使用的网络协议。

通告程序

一种由 MobiLink 服务器启动的同步使用的程序。通告程序集成在 MobiLink 服务器中。它们会检查统一数据库是否有推式请求，并发送推式通知。

另请参见：

- “服务器启动的同步”一节第 1019 页
- “监听器”一节第 1023 页

同步

利用 MobiLink 技术在数据库之间复制数据的过程。

在 SQL Remote 中，同步专指以初始数据集初始化远程数据库的过程。

另请参见：

- “MobiLink”一节第 1026 页
- “SQL Remote”一节第 1032 页

推式请求

在 MobiLink 服务器启动的同步中，通告程序通过检查它来确定推式通知是否需要发送到设备的结果集中的一行值。

另请参见： “服务器启动的同步”一节第 1019 页

推式通知

QAnywhere 中一种从服务器传送到 QAnywhere 客户端的特殊消息，用于提示客户端启动消息传输。在 MobiLink 服务器启动的同步中，从通告程序传送到包含推式请求数据和内部信息的设备的特殊消息。

另请参见：

- “QAnywhere”一节第 1028 页
- “服务器启动的同步”一节第 1019 页

UltraLite

一种针对小型设备、移动设备和嵌入式设备进行了优化的数据库。所面向的平台包括手机、传呼机和个人记事本。

UltraLite 运行时

一种过程中关系数据库管理系统，其中包括一个内置 MobiLink 同步客户端。每个 UltraLite 编程接口使用的库以及 UltraLite 引擎中都包括 UltraLite 运行时。

外表

包含外键的表。

另请参见： “外键”一节第 1034 页

外部登录

与远程服务器通信时使用的替代登录名和口令。缺省情况下，SQL Anywhere 每次代表其客户端连接到远程服务器时都会使用这些客户端的名称和口令。但是，您可以通过创建外部登录来替换这一缺省设置。外部登录是指与远程服务器通信时使用的替代登录名和口令。

外键

一个表中复制另一个表中主键值的一个或多个列。外键建立表间的关系。

另请参见：

- “主键”一节第 1042 页
- “外表”一节第 1034 页

外键约束

对单个列或一组列的限制，指定表中的数据与某个其它表中数据的关系。对列集施加外键约束可使这些列成为外键。

另请参见：

- “约束”一节第 1041 页
- “检查约束”一节第 1023 页
- “主键约束”一节第 1042 页
- “唯一约束”一节第 1036 页

外连接

一种保留表中所有行的连接。SQL Anywhere 支持左、右和完全外连接。左外连接保留表中位于连接运算符左侧的行，当右表中的行不满足连接条件时，它将返回空值。完全外连接保留两个表中的所有行。

另请参见：

- “连接”一节第 1025 页
- “内连接”一节第 1027 页

完全备份

对整个数据库和事务日志（可选）的备份。完全备份包含数据库中的所有信息，因此可以在系统或介质出现故障时提供保护。

另请参见： “增量备份”一节第 1041 页

完整性

遵守完整性规则的情况，完整性规则确保数据正确并准确，而且数据库的关系结构保持不变。

另请参见： “参照完整性”一节第 1016 页

网关

一种 MobiLink 对象，存储在 MobiLink 系统表或通告程序属性文件中，包含有关如何发送用于服务器启动同步的消息的信息。

另请参见： “服务器启动的同步”一节第 1019 页

网络服务器

从共享公共网络的计算机接受连接的数据库服务器。

另请参见： “个人服务器”一节第 1020 页

网络协议

通信类型，如 TCP/IP 或 HTTP。

维护版本

维护版本是一套完整的软件，它升级已安装的具有相同主版本号的较早版本的软件（版本号格式是 *major.minor.patch.build*）。升级程序的发行说明中列出了错误修正软件和其它更改。

唯一约束

对某个列或一组列的限制，它要求所有非空值都各不相同。一个表可以有多个唯一约束。

另请参见：

- “外键约束”一节第 1035 页
- “主键约束”一节第 1042 页
- “约束”一节第 1041 页

谓语

一种条件表达式，可以选择性地将其与逻辑运算符 AND 和 OR 组合在一起，以组成 WHERE 或 HAVING 子句中的条件集。在 SQL 中，求值结果为 UNKNOWN 的谓语将解释为 FALSE。

位数组

位数组是一种用于有效率地存储位序列的数组数据结构。位数组与字符串类似，不同的是其各个部分由 0（零）和 1（一）而不是字符组成。位数组通常用于保存一串布尔值。

Windows

Microsoft Windows 操作系统系列，如 Windows Vista、Windows XP 和 Windows 200x。

Windows CE

请参见“Windows Mobile”一节第 1036 页。

Windows Mobile

Microsoft 为移动设备制造的操作系统系列。

文件定义数据库

MobiLink 中一种用于创建下载文件的 SQL Anywhere 数据库。

另请参见： “基于文件的下载”一节第 1023 页

物理索引

索引存储在磁盘上的实际索引结构。

系统表

一种表，由 SYS 或 dbo 拥有，用于保存元数据。系统表也称作数据字典表，由数据库服务器创建并维护。

系统对象

由 SYS 或 dbo 拥有的数据库对象。

系统视图

存在于每一个数据库中的一种视图，它以易于理解的格式表示系统表中包含的信息。

下载

同步过程的一个阶段，在此阶段数据从统一数据库传送到远程数据库。

相关名

查询的 FROM 子句中使用的表或视图的名称—要么是表或视图的原始名称，要么是在 FROM 子句中定义的替代名称。

项目

在 MobiLink 或 SQL Remote 中，项目是表示整个表或表中行和列子集的数据库对象。项目在发布中组合在一起。

另请参见：

- “复制”一节第 1019 页
- “发布”一节第 1018 页

消息存储库

QAnywhere 中客户端和服务器设备上存储消息的数据库。

另请参见：

- “客户端消息存储库”一节第 1024 页
- “服务器消息存储库”一节第 1019 页

消息类型

SQL Remote 复制中指定远程用户与统一数据库发布者通信方式的数据库对象。一个统一数据库可能定义了几种消息类型，这样一来，不同的远程用户就可以使用不同的消息系统与统一数据库进行通信。

另请参见：

- “复制”一节第 1019 页
- “统一数据库”一节第 1033 页

消息日志

可存储来自数据库服务器或 MobiLink 服务器等应用程序的消息的日志。此类信息还可以出现在消息窗口中或记录到文件中。消息日志包括信息性消息、错误、警告以及来自 MESSAGE 语句的消息。

消息系统

SQL Remote 复制中用于在统一数据库与远程数据库之间交换消息的协议。SQL Anywhere 包括对以下消息系统的支持：FILE、FTP 和 SMTP。

另请参见：

- “复制”一节第 1019 页
- “FILE”一节第 1019 页

卸载

卸载数据库时会将数据库的结构和/或数据导出到文本文件（如果是结构，则导出到 SQL 命令文件中；如果是数据，则导出到 ASCII 逗号分隔文件中）。使用卸载实用程序来卸载数据库。

此外，您也可以使用 UNLOAD 语句卸载数据的选定部分。

性能统计

反映数据库系统性能的值。例如，CURRREAD 统计表示数据库服务器已发出但尚未完成的文件读取次数。

业务规则

基于实际要求的准则。通常，业务规则通过检查约束、用户定义数据类型以及事务的正确使用来实现。

另请参见：

- “约束”一节第 1041 页
- “用户定义数据类型”一节第 1039 页

引用对象

一种对象（如视图），其定义直接引用数据库中的另一个对象（如表）。

另请参见： “外键”一节第 1034 页

用户定义数据类型

请参见“域”一节第 1039 页。

游标

指向结果集的已命名链接，用于通过编程接口访问和更新行。在 SQL Anywhere 中，游标支持在查询结果中进行向前和向后移动。游标由两部分组成：游标结果集（通常由 SELECT 语句定义）和游标位置。

另请参见：

- “游标结果集”一节第 1039 页
- “游标位置”一节第 1039 页

游标结果集

与游标关联的查询所得到的行集。

另请参见：

- “游标”一节第 1039 页
- “游标位置”一节第 1039 页

游标位置

指向游标结果集中一个行的指针。

另请参见：

- “游标”一节第 1039 页
- “游标结果集”一节第 1039 页

语句级触发器

在整个触发语句完成后执行的触发器。

另请参见：

- “触发器”一节第 1017 页
- “行级触发器”一节第 1021 页

域

内置数据类型的别名，其中包括适用的精度值和小数位值，还可以选择是否包括 DEFAULT 值和 CHECK 条件。SQL Anywhere 中预定义了一些域，如货币数据类型。也称作用户定义数据类型。

另请参见：“数据类型”一节第 1030 页

预订

MobiLink 同步中发布与 MobiLink 用户之间的客户端数据库中的一个链接，它使发布所描述的数据能够得到同步。

SQL Remote 复制中发布与远程用户之间的一种链接，它使用户能够与统一数据库交换该发布上的更新。

另请参见：

- “发布”一节第 1018 页
- “MobiLink 用户”一节第 1027 页

元数据

数据的数据。元数据描述其它数据的性质和内容。

另请参见： “模式”一节第 1027 页

原子事务

保证成功完成或保证根本不予完成的事务。如果错误使原子事务的一部分无法完成，则将回退事务以防止数据库处于不一致的状态。

REMOTE DBA 特权

在 SQL Remote 中，消息代理 (dbremote) 所需的权限级别。MobiLink 中 SQL Anywhere 同步客户端 (dbmlsync) 所需的权限级别。当消息代理或同步客户端作为具有该权限的用户建立连接时，它将具有完全的 DBA 访问权。如果不是通过消息代理或同步客户端进行连接，则该用户 ID 将不具有附加权限。

另请参见： “DBA 权限”一节第 1018 页

远程 ID

SQL Anywhere 和 UltraLite 数据库中一种由 MobiLink 使用的唯一标识符。远程 ID 初始情况下设置为 NULL，在数据库第一次同步期间将设置为 GUID。

远程数据库

MobiLink 或 SQL Remote 中一种与统一数据库交换数据的数据库。远程数据库可以共享统一数据库中的全部或部分数据。

另请参见：

- “同步”一节第 1033 页
- “统一数据库”一节第 1033 页

约束

对特定数据库对象（如表或列）中所包含值的限制。例如，列可以具有唯一性约束，该约束要求该列中的所有值互不相同。表可以具有外键约束，该约束指定该表中的信息与某个其它表中数据的关系。

另请参见：

- “检查约束”一节第 1023 页
- “外键约束”一节第 1035 页
- “主键约束”一节第 1042 页
- “唯一约束”一节第 1036 页

运营公司

一种 MobiLink 对象，存储在 MobiLink 系统表或通告程序属性文件中，包含有关供服务器启动的同步使用的公共运营公司的信息。

另请参见： “服务器启动的同步”一节第 1019 页

增量备份

仅包含事务日志的备份，通常在两次完全备份之间使用。

另请参见： “事务日志”一节第 1029 页

争用

为获取资源而竞争的行为。例如，就数据库而言，如果有两个或更多个用户试图编辑数据库的同一行，就会为获得编辑该行的权利而发生争用。

正则表达式

正则表达式是字符、通配符和运算符的序列，用于定义某种模式以在字符串内进行搜索。

直方图

直方图是列统计信息最重要的组成部分，是一种表示数据分布的方式。SQL Anywhere 维护直方图以为优化程序提供有关列值分布情况的统计信息。

直接行处理

MobiLink 中一种用于将表数据同步到 MobiLink 支持的统一数据库以外的数据源的方法。使用直接行处理时，上载和下载都可以实现。

另请参见：

- “统一数据库”一节第 1033 页
- “基于 SQL 的同步”一节第 1023 页

主表

包含外键关系中的主键的表。

主键

其值唯一标识表中各行的一个列或多个列。

另请参见： “外键”一节第 1034 页

主键约束

一种对主键列的唯一性约束。一个表只能有一个主键约束。

另请参见：

- “约束”一节第 1041 页
- “检查约束”一节第 1023 页
- “外键约束”一节第 1035 页
- “唯一约束”一节第 1036 页
- “完整性”一节第 1035 页

子查询

嵌套在 SELECT、INSERT、UPDATE 或 DELETE 语句或者其它子查询中的 SELECT 语句。

有两种类型的子查询：相关子查询和嵌套子查询。

字符串

字符串是以单引号围起的字符序列。

字符集

字符集是一组符号，包括字母、数字、空格和其它符号。字符集的一个例子是 ISO-8859-1，又称作 Latin1。

另请参见：

- “代码页”一节第 1017 页
- “编码”一节第 1015 页
- “归类”一节第 1021 页

索引

其它

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

