SYBASE®
An SAP® Company

系统管理指南第二卷

Sybase IQ 15.3

文档 ID： DC01144-01-1530-02
最后修订日期： 2011 年 10 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568。

目录

读者

本指南的目标读者是访问 Sybase® IQ 数据库中数据的应用程序的开发人员。

他们熟悉关系数据库系统并具有 Sybase IQ 产品的入门级用户经验。请将本指南与文档集内的其它手册结合使用。

使用过程和批处理

创建与 Sybase IQ 一起使用的过程和批处理。

过程在数据库中存储过程 SQL 语句，以供所有应用程序使用。它们增强了数据库的安全性，提高了效率并促进了标准化。用户定义的函数是一种类型的过程，它们将值返回到调用环境中，以供查询和其它 SQL 语句使用。

出于多种目的，服务器端 JDBC 提供比 SQL 存储过程更灵活的方法来将逻辑内置于数据库中。请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 编程”>“SQL Anywhere 数据访问 API”>“SQL Anywhere JDBC 驱动程序”>“JDBC 简介”。

批处理是作为一组提交到数据库服务器的 SQL 语句的集合。在过程中提供的许多功能（例如控制语句）在批处理中也提供。

过程概述

过程在数据库中存储过程 SQL 语句，以供所有应用程序使用。它们包括允许 SQL 语句的重复执行（**LOOP** 语句）和条件执行（**IF** 语句和 **CASE** 语句）的控制语句。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器概述”。

注意： Sybase IQ 不支持触发器。可以忽略 SQL Anywhere 文档中有关触发器的信息。

过程的优点

过程的定义在数据库中提供，与任何一个数据库应用程序相分离。这一分离具有许多优点。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器的优点”。

过程简介

本节讨论可用的过程及其功能。

在使用存储过程时，有两个系统存储过程非常有用：**sp_iqprocedure** 和 **sp_iqprocparm**。**sp_iqprocedure** 存储过程显示数据库中与系统过程和用户定义过程有关的信息。**sp_iqprocparm** 存储过程显示有关存储过程参数的信息，包括以下列：

- proc_name

- proc_owner
- parm_name
- parm_type
- parm_mode
- domain_name
- width, scale
- default

另请参见

- 过程结果 （第 11 页）

创建过程

过程是用 **CREATE PROCEDURE** 语句创建的。您必须具有资源权限才能创建过程。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程简介”>“创建过程”。

Sybase IQ 示例

注意： 例如，使用 Sybase IQ 演示数据库 `iqdemo.db`。

```
CREATE PROCEDURE new_dept(IN id INT,
             IN name CHAR(35),
                IN head_id INT)
BEGIN
        INSERT
                INTO GROUPO.departments(DepartmentID,
                DepartmentName,
                DepartmentHeadID)
                values (id, name, head_id);
END
```

注意： 要在 IQ 中创建远程过程，必须使用 **CREATE PROCEDURE** 的 `AT` *`location-string`* SQL 语法来创建代理存储过程。此功能目前仅在 Windows 和 Sun Solaris 上得到认证。Sybase Central 中的“创建远程过程向导”(Create Remote Procedure Wizard) 仅适用于远程服务器。

更改过程

您可以使用 Sybase Central 或 Interactive SQL 修改现有过程。您必须具有 DBA 权限或是过程的所有者。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程简介”>“更改过程”。

有关更改数据库对象属性的信息，请参见《Sybase IQ 简介》>“管理数据库”>“管理过程”。

有关授予或撤消过程权限的信息，请参见《系统管理指南第一卷》>“管理用户 ID 和权限”>“管理单个用户 ID 和权限”>“在 Interactive SQL 中授予针对过程的权限”

以及《系统管理指南第一卷》>“管理用户 ID 和权限”>“管理单个用户 ID 和权限”>“在 Interactive SQL 中撤消用户权限”。

您还可以使用 **ALTER PROCEDURE** 语句来修改过程。

调用过程

CALL 语句可调用过程。过程可由应用程序调用，也可以由其它过程调用。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程简介”>“调用过程”。

另请参见

- 执行过程的权限 （第 5 页）

在 Sybase Central 中复制过程

您可以将一个数据库中的过程代码复制到另一个连接的数据库。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程简介”>“在 Sybase Central 中复制过程”。

删除过程

在创建了过程后，该过程将一直保留在数据库中，直到它被显式删除。只有该过程的所有者或具有 DBA 权限的用户才可以从数据库中删除该过程。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程简介”>“删除过程”。

执行过程的权限

过程由创建它的用户拥有，即使没有权限，该用户也可以执行它。

可以使用 **GRANT EXECUTE** 命令为其他用户授予执行过程的权限。例如，**new_dept** 过程的所有者可以通过以下语句允许 **another_user** 执行 **new_dept**：

```
GRANT EXECUTE ON new_dept TO another_user
```

以下语句撤消该过程的执行权限：

```
REVOKE EXECUTE ON new_dept FROM another_user
```

请参见《系统管理指南第一卷》>“管理用户 ID 和权限”>“管理单个用户 ID 和权限”>“在 Interactive SQL 中授予针对过程的权限”。

另请参见

- 调用过程 （第 5 页）

在参数中返回过程结果

过程将结果返回到调用环境中。

过程通过以下方式之一返回结果：

- 单独的值作为 OUT 或 INOUT 参数返回。
- 可返回结果集。
- 可以使用 **RETURN** 语句返回单个结果。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程简介”>“在参数中返回过程结果”。

Sybase IQ 示例

注意： 例如，使用 Sybase IQ 演示数据库 `iqdemo.db`。

```
CREATE PROCEDURE SalaryList (IN department_id INT)
RESULT ( "Employee ID" INT, "Salary" NUMERIC(20,3) )
BEGIN
            SELECT EmployeeID, Salary
            FROM Employees
            WHERE Employees.DepartmentID = department_id;
END
```

在结果集中返回过程结果

除了在单独参数中将结果返回到调用环境中之外，过程可以在结果集内返回信息。结果集通常是查询的结果。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程简介”>“在结果集中返回过程结果”。

创建临时表并从中进行选择

如果某个过程在存储过程中动态创建临时表并随后选择该表，您必须使用 **EXECUTE IMMEDIATE WITH RESULT SET ON** 语法来避免出现“未找到列”错误。

例如：

```
CREATE PROCEDURE p1 (IN @t varchar(30))    BEGIN       EXECUTE
IMMEDIATE       'SELECT * INTO #resultSet FROM ' || @t;      EXECUTE
IMMEDIATE WITH RESULT SET ON        'SELECT * FROM
#resultSet';   END
```

用户定义的函数简介

用户定义的函数是一种过程，此类函数将单个值返回到调用环境中。本节介绍如何创建、使用和删除用户定义的函数。

创建用户定义的函数

您可以使用 **CREATE FUNCTION** 语句来创建用户定义的函数。但是，您必须具有 RESOURCE 授权。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“用户定义的函数简介”>“创建用户定义的函数”。

有关 **CREATE FUNCTION** 语法的完整说明（包括性能注意事项以及 SQL Anywhere 和 IQ 之间的区别），请参见《参考：语句和选项》>“SQL 语句”>“CREATE FUNCTION 语句”。

调用用户定义的函数

在您要使用内置的非集合函数的任何位置，都可以根据权限使用用户定义的函数。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“用户定义的函数简介”>“调用用户定义的函数”。

Sybase IQ 示例

注意： 例如，使用 Sybase IQ 演示数据库 `iqdemo.db`。

```
SELECT fullname (GivenName, SurName)FROM Employees;
```

fullname (GivenName, SurName)

Fran Whitney Matthew Cobb Philip Chin……

删除用户定义的函数

用户定义的函数一经创建，便会保留在数据库中，直到它被显式删除。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“用户定义的函数简介”>“删除用户定义的函数”。

执行用户定义的函数的权限

用户定义的函数由创建它的用户拥有，即使没有权限，该用户也可以执行它。

用户定义的函数的所有者可以使用 **GRANT EXECUTE** 命令向其他用户授予权限。

例如，fullname 函数的创建者可以通过以下语句允许 another_user 使用 fullname：

```
GRANT EXECUTE ON fullname TO another_user
```

以下语句撤消使用该函数的权限：

```
REVOKE EXECUTE ON fullname FROM another_user
```

请参见《系统管理指南第一卷》>“管理用户 ID 和权限”>“管理单个用户 ID 和权限”>“在 Interactive SQL 中授予针对过程的权限”。

批处理简介

简单的批处理由一组用分号隔开的 SQL 语句组成。

例如，下面的一组语句构成一个批处理，该批处理创建 Eastern Sales 部门并将所有销售代表从 Massachusetts (MA) 转移到该部门。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“批处理简介”。

Sybase IQ 示例

注意： 例如，使用 Sybase IQ 演示数据库 iqdemo.db。

```
INSERT
INTO Departments ( DepartmentID, DepartmentName )
VALUES ( 220, 'Eastern Sales' ) ;
UPDATE Employees

SET DepartmentID = 220
WHERE DepartmentID = 200
AND state = 'GA' ;

COMMIT ;
```

控制语句

在过程的主体或在批处理中，有多种用于逻辑流和决策的控制语句。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“控制语句”。

有关每个控制语句的完整说明，请参见《参考：语句和选项》>“SQL 语句”中的条目。

使用复合语句

复合语句可被嵌套，并且可以与其它控制语句合并在一起，以定义过程中或批处理中的执行流。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“控制语句”>“使用复合语句”。

另请参见

- 过程中允许的 SQL 语句 （第 10 页）
- 过程的结构 （第 9 页）
- 过程中的事务和保存点 （第 14 页）

复合语句中的声明

复合语句中的局部声明紧随在 **BEGIN** 关键字之后。这些局部声明只存在于复合语句内。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“控制语句”>“复合语句中的声明”。

原子复合语句

*原子*语句是完全执行或根本不执行的语句。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“控制语句”>“原子复合语句”。

过程的结构

过程的主体由复合语句组成。

复合语句由 **BEGIN** 和 **END** 组成，它们将一组 SQL 语句括起来，用分号分隔每一语句。

另请参见

- 过程中允许的 SQL 语句 （第 10 页）
- 过程中的事务和保存点 （第 14 页）
- 使用复合语句 （第 9 页）

过程中允许的 SQL 语句

您可以在过程内使用几乎所有的 SQL 语句，包括以下各种语句：

- **SELECT**、**UPDATE**、**DELETE**、**INSERT** 和 **SET VARIABLE**
- 用来执行其它过程的 **CALL** 语句
- 控制语句
- 游标语句
- 异常处理语句
- **EXECUTE IMMEDIATE** 语句

下面是一些不能在过程中使用的 SQL 语句：

- **CONNECT** 语句
- **DISCONNECT** 语句

可以在过程中使用 **COMMIT**、**ROLLBACK** 和 **SAVEPOINT** 语句，但有一些限制。

请参见《参考：语句和选项》>“SQL 语句”中每个语句的“用法”一节。

另请参见

- 过程的结构 （第 9 页）
- 过程中的事务和保存点 （第 14 页）
- 使用复合语句 （第 9 页）

为过程声明参数

过程参数在 **CREATE PROCEDURE** 语句中以列表形式出现。

参数名必须符合其它数据库标识符（如列名）的规则。它们必须具有有效的数据类型，而且必须用关键字 `IN`、`OUT` 或 `INOUT` 之一作为前缀。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器的结构”>“为过程声明参数”。

将参数传递给过程

您可以通过 **CALL** 语句的两种形式之一，利用存储过程参数的缺省值。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器的结构”>“将参数传递给过程”。

将参数传递给函数

UDF 不通过 **CALL** 语句进行调用，而是采用与内置函数同样的方式使用。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器的结构”>“将参数传递给函数”。

过程结果

过程能够以单行数据或多行数据的形式返回结果。

由单行数据组成的结果可作为参数传递回过程。由多行数据组成的结果作为结果集传递回过程。过程也可以返回在 **RETURN** 语句中给定的单个值。

有关如何从过程返回结果的简单示例，请参见“过程简介”。有关详细信息，请参见以下各节。

另请参见

- 过程简介 （第 3 页）

使用 RETURN 语句返回值

RETURN 语句将单个整数值返回到调用环境中，并且导致立即从过程退出。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“从过程返回结果”>“使用 RETURN 语句返回值”。

作为过程参数返回结果

过程可以将结果作为该过程的参数返回到调用环境中。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“从过程返回结果”>“作为过程参数返回结果”。

从过程返回结果集

结果集允许过程将多行结果返回到调用环境中。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“从过程返回结果”>“从过程返回结果集”。

从过程返回多个结果集

过程可以将多个结果集返回到调用环境中。

dbisql 和 **dbisqlc** 用来返回多个结果集的方法不同。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“从过程返回结果”>“从过程返回多个结果集”。

从过程返回可变结果集

RESULT 子句在过程中是可选的。通过省略 RESULT 子句，您可以编写返回不同结果集的过程，结果集内具有不同数目或类型的列，具体情况取决于执行它们的方式。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“从过程返回结果”>“从过程返回可变结果集”。

过程中的游标

游标在其结果集内从具有多行的查询或存储过程中一次检索一行。

游标是查询或过程的句柄或标识符，也是结果集内的当前位置的句柄或标识符。

游标管理概述

管理游标类似于通过编程语言管理文件。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“在过程和触发器中使用游标”>“游标管理概述”。

sp_iqcursorinfo 存储过程可显示有关在服务器上当前打开的游标的信息。有关详细信息，请参见《参考：构件块、表和过程》>“系统过程”>“sp_iqcursorinfo 过程”。

游标定位

游标定位极其灵活。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 编程”>“使用 SQL Anywhere 编程简介”>“在应用程序中使用 SQL”>“使用游标”>“游标定位”。

注意： Sybase IQ 从结果集的开头处理 FIRST、LAST 和 ABSOLUTE 选项。它将行计数为负值的 RELATIVE 视为从当前位置开始。

过程中的游标和 SELECT 语句

`TopCustomerValue` 过程在 **SELECT** 语句上使用游标并基于 `ListCustomerValue` 过程中使用的同一查询。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“在过程和触发器中使用游标”>“在过程中在 SELECT 语句上使用游标”。

过程中的错误和警告

在应用程序执行 SQL 语句后，它可以检查*返回码*（或状态码）以了解是否有错误。

返回码指示所执行的语句是成功还是失败，并给出失败的原因。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器中的错误和警告”。

注意： Sybase IQ 不支持触发器。可以忽略 SQL Anywhere 文档中有关触发器的信息。

过程中缺省的错误处理

Sybase IQ 会处理过程执行期间发生的错误。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器中的错误和警告”>“过程和触发器中缺省的错误处理”。

注意： Sybase IQ 不支持触发器。可以忽略 SQL Anywhere 文档中有关触发器的信息。

使用 ON EXCEPTION RESUME 处理错误

ON EXCEPTION RESUME 子句包括在 **CREATE PROCEDURE** 语句中。

发生错误时，过程会检查语句。如果该语句处理错误，则在发生错误时该过程不返回对调用环境的控制。相反它继续执行，在导致错误的语句处继续执行。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器中的错误和警告”>“使用 ON EXCEPTION RESUME 处理错误”。

过程中错误和警告的缺省处理

在过程中，系统对错误和警告的处理方式不同。

对错误的缺省操作是为 *SQLSTATE* 和 *SQLCODE* 变量设置值，并且在遇到错误时将控制返回到调用环境；对警告的缺省操作是设置 *SQLSTATE* 和 *SQLCODE* 值，并且继续执行过程。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器中的错误和警告”>“过程和触发器中警告的缺省错误处理”。

注意： Sybase IQ 不支持触发器。您可以忽略 SQL Anywhere 文档中有关触发器的信息。

在过程中使用异常处理程序

您可以在过程内截取并处理某些类型的错误，而不是将错误传递回调用环境。上述操作是通过使用*异常处理程序*执行的。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器中的错误和警告”>“在过程和触发器中使用异常处理程序”。

注意： Sybase IQ 不支持触发器。可以忽略 SQL Anywhere 文档中有关触发器的信息。

嵌套的复合语句和异常处理程序

嵌套的复合语句可用于增强用户相关控制，以控制哪些语句在出现错误之后仍执行，哪些语句不执行。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器中的错误和警告”>“嵌套的复合语句和异常处理程序”。

在过程中使用 EXECUTE IMMEDIATE 语句

EXECUTE IMMEDIATE 语句允许使用文字字符串（在引号中）和变量的组合，在过程内编译语句。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“在过程中使用 EXECUTE IMMEDIATE 语句”。

过程中的事务和保存点

过程或触发器中的 SQL 语句是当前事务的一部分。

您可以在一个事务内调用若干过程，或在一个过程中具有若干事务。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程和触发器中的事务和保存点”。

注意： Sybase IQ 不支持触发器。可以忽略 SQL Anywhere 文档中有关触发器的信息。

有关详细信息，请参见《系统管理指南第一卷》>“事务和版本控制”>“事务内的保存点”。

另请参见

- 过程中允许的 SQL 语句 （第 10 页）
- 过程的结构 （第 9 页）
- 使用复合语句 （第 9 页）

隐藏过程、函数和视图的内容

在某些情况下，当您分发应用程序和数据库时，可能不希望公开过程、函数、触发器和视图中包含的逻辑。

作为附加的安全手段，您可以使用 **ALTER PROCEDURE**、**ALTER FUNCTION** 和 **ALTER VIEW** 语句的 SET HIDDEN 子句来隐藏这些对象的内容。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“隐藏过程、函数、触发器和视图的内容”。

注意： Sybase IQ 不支持触发器。可以忽略 SQL Anywhere 文档中有关触发器的信息。

有关详细信息，请参见《参考：语句和选项》中的 `ALTER FUNCTION` 语句、`ALTER PROCEDURE` 语句和 `ALTER VIEW` 语句。

批处理中允许的语句

批处理中可以接受大多数 SQL 语句，但也有一些例外。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程、触发器、事件和批处理中允许的语句”。

注意： Sybase IQ 不支持触发器。可以忽略 SQL Anywhere 文档中有关触发器的信息。

在批处理中使用 SELECT 语句

可以在批处理中包括一个或多个 **SELECT** 语句。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“使用过程、触发器和批处理”>“过程、触发器、事件和批处理中允许的语句”>“在批处理中使用 SELECT 语句”。

Sybase IQ 示例

注意： 例如，使用 Sybase IQ 演示数据库 `iqdemo.db`。

```
IF EXISTS(
            SELECT * FROM SYSTAB
                        WHERE table_name='Employees' )
THEN
            SELECT Surname AS LastName,
                        GivenName AS FirstName
            FROM Employees;
            SELECT Surname, GivenName
            FROM Customers;
            SELECT Surname, GivenName
            FROM Contacts;
ELSE
            MESSAGE 'The Employees table does not exist'
            TO CLIENT;
END IF
```

使用 IQ UTILITIES 创建自己的存储过程

Sybase IQ 中提供的系统存储过程可以在 SQL 中，借助于本章其余部分介绍的方法来实现。

您必须按照与系统存储过程完全相同的方式使用本地临时表和 **IQ UTILITIES** 语句：

警告！ 违反这些规则可能会导致 IQ 服务器或数据库出现严重问题。

过程的所有 SQL 代码均会被加密并编译到共享库 `libiqscripts15_r.so` 文件（在 Unix 上）和 `iqscripts15.dll` 文件（在 Windows 上）中。

您可以使用 Sybase Central 或在 Interactive SQL 中输入 **sp_helptext 'owner.procname'** 来查看存储过程代码。

IQ UTILITIES 的语法为：

```
IQ UTILITIES  MAIN INTO local-temp-table-name arguments
```

《参考：语句和选项》中仅记录了针对 IQ 监视器的 **IQ UTILITIES** 命令，这是由于该命令的使用要求非常严格，如果使用不当，会使系统运行面临风险。

您可能希望为其中某些过程创建自己的变体。下面是一些具体操作方法：

1. 创建一个调用系统存储过程的过程。
2. 创建一个与系统存储过程无关但是执行类似功能的过程。
3. 创建一个与系统存储过程使用相同结构但是提供其它功能的过程。例如，您可能希望在前端工具或浏览器中以图形（而不是文本）方式显示过程结果。
4. 如果您选择第二个或第三个选项，则需要了解 **IQ UTILITIES** 语句以及使用该语句而需要满足的严格要求。

IQ 如何使用 IQ UTILITIES 命令

IQ UTILITIES 是基础语句，它会在您运行大多数 IQ 系统过程时执行。在大多数情况下，用户并不知道 **IQ UTILITIES** 正在执行。用户仅在运行 IQ 缓冲区高速缓存监视器时才会直接发出 **IQ UTILITIES**。

IQ UTILITIES 提供一种系统方法来收集和报告 IQ 系统表中所维护的信息。没有通用的用户界面，您只能按照与现有系统过程相同的方法来使用 **IQ UTILITIES**。

系统过程声明用来存储信息的本地临时表。它们执行 **IQ UTILITIES** 来从系统表获得信息并将这些信息存储在本地临时表中。系统过程可以只是报告本地临时表中的信息，也可以执行其它处理。

在某些系统过程中，**IQ UTILITIES** 语句包括一个预定义的数字作为它的某个参数。这个数字执行特定的函数，例如，从系统表中的信息派生值。有关可用作 **IQ UTILITIES** 参数的数字列表，请参见“表 1”。

选择要调用的过程

您可以安全地使用 **IQ UTILITIES** 来基于所说明的系统过程创建自己的版本，并用其报告数据库中的信息。

例如，**sp_iqspaceused** 显示有关 IQ 主存储和 IQ 临时存储中已用空间和可用空间的信息。检查您基于系统存储过程所创建过程的所有者，确保您所创建版本的过程具有正确的所有者。

请勿基于系统过程创建用来控制 IQ 操作的过程。修改用来控制 IQ 操作的过程可能会导致严重问题。

IQ UTILITIES 使用的数字

下表列出了在 **IQ UTILITIES** 命令中用作参数的数字以及使用每个数字的系统过程。

有关这些过程的功能的信息，请参见《参考：构件块、表和过程》>“系统过程”。

表 1. 系统过程中使用的 IQ UTILITIES 值

数字	过程	注释
10000	sp_iqtransaction	
20000	sp_iqconnection 和 sp_iqmpxcountdbremote	
30000	sp_iqspaceused	
40000	sp_iqspaceinfo	
50000	sp_iqlocks	
60000	sp_iqmpxversionfetch	不使用

数字	过程	注释
70000	sp_iqmpxdumptlvlog	
80000	sp_iqcontext	
100000	sp_iqindexfragmentation	
110000	sp_iqrowdensity	

过程测试

请始终首先在开发环境中测试您的过程。在生产环境中运行过程之前首先对过程进行测试有助于维护 IQ 服务器和数据库的稳定性。

使用 OLAP

OLAP（联机分析处理）是对存储在关系数据库中的信息执行数据分析的有效方法。

使用 OLAP，可以基于不同的维度对数据进行分析，获取含小计行的结果集，以及将数据组织成多维 CUBE，所有这些操作均可通过一个 SQL 查询完成。您也可以使用过滤器深入分析数据，从而快速返回结果集。本章介绍了 Sybase IQ 支持的 SQL/OLAP 功能。

注意： OLAP 示例中显示的表位于 `iqdemo` 数据库中。

关于 OLAP

这些分析函数（使用它们，可以通过一个 SQL 语句执行复杂数据分析）通过一种名为“联机分析处理”(OLAP) 的软件技术来执行。以下列表中显示了此类函数：

- **GROUP BY** 子句扩展 - **CUBE** 和 **ROLLUP**
- 分析函数：
  - 简单集合 - **AVG**、**COUNT**、**MAX**、**MIN**、**SUM**、**STDDEV** 和 **VARIANCE**

    注意： 您可以将除 **Grouping()** 以外的简单集合函数与 OLAP 窗口函数结合使用。
  - 窗口函数：
    - 窗口化集合 - **AVG**、**COUNT**、**MAX**、**MIN** 和 **SUM**
    - 排名函数 - **RANK**、**DENSE_RANK**、**PERCENT_RANK** 和 **NTILE**
    - 统计函数 - **STDDEV**、**STDDEV_SAMP**、**STDDEV_POP**、**VARIANCE**、**VAR_POP**、**VAR_SAMP**、**REGR_AVGX**、**REGR_AVGY**、**REGR_COUNT**、**REGR_INTERCEPT**、**REGR_R2**、**REGR_SLOPE**、**REGR_SXX**、**REGR_SXY**、**REGR_SYY**、**CORR**、**COVAR_POP**、**COVAR_SAMP**、**CUME_DIST**、**EXP_WEIGHTED_AVG** 和 **WEIGHTED_AVG** 。
    - 分布函数 - **PERCENTILE_CONT** 和 **PERCENTILE_DISC**
  - 数值函数 - **WIDTH_BUCKET**、**CEIL**、**LN**、**EXP**、**POWER**、**SQRT** 和 **FLOOR**

已经对 1999 SQL 标准进行了修订，即引入了包括复杂数据分析的 ANSI SQL 标准扩展。Sybase IQ 中加入了这些 SQL 增强中的部分内容，可针对扩展提供其它全面支持。

有些数据库产品会提供一个单独的 OLAP 模块，您在分析数据之前需要先将数据从数据库移至该 OLAP 模块中。与此不同，Sybase IQ 将 OLAP 功能内置在数据库本身当中，从而使您可以轻松而无缝地进行部署以及与其它数据库功能（如存储过程）集成。

OLAP 优点

OLAP 函数与 **GROUPING**、**CUBE** 和 **ROLLUP** 扩展组合使用时，具有以下两大优点。

首先，利用它们，可以执行多维数据分析、数据采集、时序分析、趋势分析、成本分配、目标寻求、即席多维结构更改、非过程建模，以及异常警告，执行这些操作时通常是利用一个 SQL 语句。其次，窗口集合函数和报告聚合函数使用一个名为 *window* 的关系运算符，与使用自连接或相关子查询的同语义查询相比，该运算符可以更有效地执行。使用 OLAP 获得的结果集可以包含小计行，并且可以组织成多维 CUBE。请参见“窗口化”。

可计算各种区间的移动平均值和移动总和，在所选列值改变时重置集合和秩，以及使用简单的术语来表示复杂的比值。在一个查询表达式的范围内，您可以定义多个不同的 OLAP 函数，每个函数都有它自己的分区规则。

另请参见

- 分布函数 （第 53 页）
- OLAP 计算 （第 20 页）
- 排名函数 （第 43 页）
- 统计集合函数 （第 49 页）
- 窗口化 （第 33 页）
- 窗口化集合函数 （第 47 页）
- OLAP 函数的 BNF 语法 （第 67 页）

OLAP 计算

OLAP 计算可概念化为多个查询执行阶段，这些阶段共同配合来生成最终结果。

您可以通过查询中的相关子句来标识执行的 OLAP 阶段。例如，如果 SQL 查询规范包含窗口函数，则会首先处理 **WHERE**、**JOIN**、**GROUP BY** 和 **HAVING** 子句。分区是在 GROUP BY 子句中定义组之后、对查询的 **ORDER BY** 子句中的最终 **SELECT** 列表进行计算之前创建的。

为了进行分组，将所有空值视为属于同一组，即使空值彼此不相等也是如此。

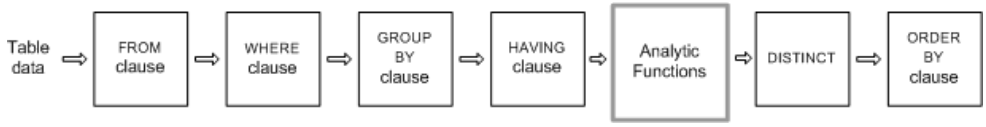
HAVING 子句充当过滤器（与 **WHERE** 子句非常相似），用于过滤 GROUP BY 子句的结果。

请考虑涉及 ANSI SQL 标准中的 SQL 语句和子句（**SELECT**、**FROM**、**WHERE**、**GROUP BY** 和 **HAVING**）的简单查询规范的语义：

1. 查询生成一组满足 **FROM** 子句中存在的表表达式的行。
2. **WHERE** 子句中的谓词应用到表中的行。不满足 **WHERE** 子句条件的行（不等于 True）将被拒绝。
3. 除集合函数外，系统还会为其余每一行计算 **SELECT** 列表和 GROUP BY 子句中的表达式。

4. 结果行根据 GROUP BY 子句中表达式的不同的值组合在一起，并将空值视为每个域中的特殊值。如果存在 **PARTITION BY** 子句，则 GROUP BY 子句中的表达式充当分区键。
5. 对于每个分区，系统将计算 **SELECT** 列表或 **HAVING** 子句中存在的集合函数。单个表行一旦集合在一起，便无法再显示在中间结果集中。新结果集包含 **GROUP BY** 表达式以及为每个分区计算的集合函数的值。
6. **HAVING** 子句中的条件应用到结果组。不满足 **HAVING** 子句的组将被删除。
7. 结果根据 **PARTITION BY** 子句中定义的边界进行分区。系统将为结果窗口计算 OLAP 窗口函数（秩和集合）。

图 1: 对 OLAP 的 SQL 处理



请参见“语法规则 2”。另请参见“OLAP 函数的 BNF 语法”。

另请参见

- 分布函数 （第 53 页）
- OLAP 优点 （第 20 页）
- 排名函数 （第 43 页）
- 统计集合函数 （第 49 页）
- 窗口化 （第 33 页）
- 窗口化集合函数 （第 47 页）
- OLAP 函数的 BNF 语法 （第 67 页）

GROUP BY 子句扩展

使用 **GROUP BY** 子句的扩展，应用程序开发人员可以编写用于执行以下操作的复杂 SQL 语句：

- 将输入行以多维形式分区，并将结果组的多个子集合并。
- 创建“数据 CUBE”，从而提供稀疏的多维结果集以进行数据采集分析。
- 创建包含原始组（并且也可包含小计和总计行）的结果集。

OLAP Grouping() 操作（如 **ROLLUP** 和 **CUBE**）可概念化为前缀和小计行。

前缀

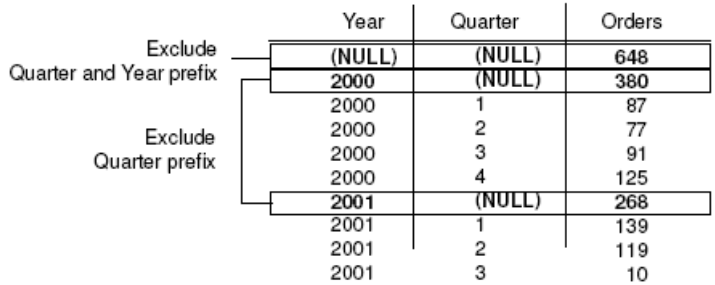
*前缀列*表是为任何包含 **GROUP BY** 子句的查询而构造的。前缀是 **GROUP BY** 子句中项的子集，并且是通过从查询的 **GROUP BY** 子句中的项中排除最右侧的一个或多个项构造而成的。排除后所剩余的列称为*前缀列*。

ROLLUP 示例 1 - 在以下 **ROLLUP** 示例查询中，**GROUP BY** 列表包括 *Year* 和 *Quarter* 这两个变量：

```
SELECT year (OrderDate) AS Year, quarter(OrderDate)
  AS Quarter, COUNT(*) Orders
FROM SalesOrders
GROUP BY ROLLUP(Year, Quarter)
ORDER BY Year, Quarter
```

查询的两个前缀是：

- Exclude Quarter - 前缀列的集包含一个列 `Year`。
- Exclude both Quarter and Year – 不存在前缀列。



Year	Quarter	Orders
(NULL)	(NULL)	648
2000	(NULL)	380
2000	1	87
2000	2	77
2000	3	91
2000	4	125
2001	(NULL)	268
2001	1	139
2001	2	119
2001	3	10

注意： **GROUP BY** 列表包含与项数量相同的前缀。

Group by ROLLUP 和 Group by CUBE

ROLLUP 和 **CUBE** 是指定常见分组前缀的语法快捷方式。

Group by ROLLUP

ROLLUP 运算符要求以参数的方式提供分组表达式的有序列表。

ROLLUP 语法。

```
            SELECT ⋯ [ GROUPING (column-name) ⋯ ] ⋯
GROUP BY [ expression [, ⋯]
| ROLLUP ( expression [, ⋯] ) ]
```

GROUPING 采用列名作为参数，并返回下表中所列的布尔值：

表 2. 使用 ROLLUP 运算符时 GROUPING 返回的值

如果结果值是	GROUPING 将返回
由 **ROLLUP** 运算创建的空值	1 (TRUE)
指示该行是小计所在行的空值	1 (TRUE)
并非由 **ROLLUP** 运算创建	0 (FALSE)

如果结果值是	GROUPING 将返回
存储的空值	0 (FALSE)

ROLLUP 首先计算 **GROUP BY** 子句中指定的标准集合值。然后，**ROLLUP** 在整个分组列的列表中从右侧移到左侧，并以累积方式创建更高级别的小计。在结尾处创建总计。如果 *n* 代表分组列数，则 **ROLLUP** 会创建 *n*+1 个级别的小计。

此 SQL 语法...	定义以下集...
`GROUP BY ROLLUP (A, B, C);`	(A, B, C) (A, B) (A) ( )

ROLLUP 和小计行

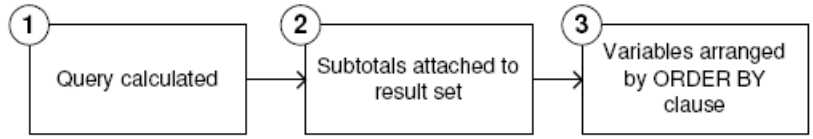
ROLLUP 等同于一组 **GROUP BY** 查询的 **UNION**。以下查询的结果集是相同的。**GROUP BY** (A, B) 的结果集包括所有那些 A 和 B 保持不变的行的小计。要实现联合，可为列 C 分配空值。

此 ROLLUP 查询...	等同于未进行 ROLLUP 的此查询...
`select year(orderdate) as year, quarter(orderdate) as Quarter, count(*) Ordersfrom SalesOrdersgroup by Rollup (year, quarter)order by year, quarter`	`Select null,null, count(*) Orders from SalesOrdersunion allSELECT year(orderdate) AS YEAR, NULL, count(*) Orders from SalesOrdersGROUP BY year(orderdate) union allSELECT year(orderdate) as YEAR, quarter(orderdate) as QUATER, count(*) Orders from SalesOrdersGROUP BY year(orderdate), quarter(orderdate)`

小计行可以帮助您分析数据，尤其是在以下情况下：有大量数据、数据有不同的维数、数据包含在不同的表甚至完全不同的数据库中。例如，一名销售经理可能会发现按销售代表、区域和季度进行分类的有关销售数据的报告对于了解销售模式十分有用。数据小计可让销售经理从不同的角度了解总体销售情况。如果根据销售经理要比较的条件提供了摘要信息，分析此数据就会更加容易。

使用 OLAP，用户看不到分析和计算行和列小计的过程。

图 2： 小计



1. 此步骤生成尚未考虑 **ROLLUP** 的中间结果集。
2. 计算小计并将其附加到结果集中。
3. 行根据查询中的 **ORDER BY** 子句排列。

空值和小计行

当 **GROUP BY** 运算的输入行包含空值时，可能会将由 **ROLLUP** 或 **CUBE** 运算添加的小计行与包含的空值属于原始输入数据的行混淆。

Grouping() 函数通过以下方法将小计行与其它行区分开：采用 **GROUP BY** 列表中的列作为它的参数，并在该列为空值时（因为行是小计行）返回 1，在该列不为空值时返回 0。

下面的示例在结果集中包括 Grouping() 列。包含空值作为输入数据结果的行突出显示，小计行则不会。Grouping() 列会突出显示。查询是 `Employees` 表和 `SalesOrders` 表之间的外连接。查询选择居住在德克萨斯、纽约或加利福尼亚的女雇员。空值会显示在与不是销售代表（因而没有销售额）的那些女雇员相对应的列中。

注意： 例如，使用 Sybase IQ 演示数据库 `iqdemo.db`。

```
SELECT Employees.EmployeeID as EMP, year(OrderDate) as
  YEAR, count(*) as ORDERS, grouping(EMP) as
  GE, grouping(YEAR) as GY
  FROM Employees LEFT OUTER JOIN SalesOrders on
  Employees.EmployeeID = SalesOrders.SalesRepresentative
  WHERE Employees.Sex IN ('F') AND Employees.State
  IN ('TX', 'CA', 'NY')
GROUP BY ROLLUP (YEAR, EMP)
ORDER BY YEAR, EMP
```

前面的查询返回：

```
EMP         YEAR      ORDERS     GE     GY
------      ----      ------     --     --
NULL        NULL           5      1      0
NULL        NULL         169      1      1
 102        NULL           1      0      0
 309        NULL           1      0      0
1062        NULL           1      0      0
1090        NULL           1      0      0
1507        NULL           1      0      0
NULL        2000          98      1      0
 667        2000          34      0      0
 949        2000          31      0      0
```

```
1142        2000          33      0      0
NULL        2001          66      1      0
 667        2001          20      0      0
949         2001          22      0      0
1142        2001          24      0      0
```

对于每个前缀，*小计行*是对应于所有那些前缀列具有相同值的行而构建的。

为了说明 **ROLLUP** 结果，请再看一下示例查询：

```
SELECT year (OrderDate) AS Year, quarter
  (OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
  GROUP BY ROLLUP (Year, Quarter)
  ORDER BY Year, Quarter
```

在此查询中，包含 `Year` 列的前缀位于 Year=2000 的汇总行和 Year=2001 的汇总行前面。前缀的单一汇总行不包括列，该行是中间结果集中所有行的小计。

小计行中每一列的值如下：

- 前缀中包括的列 - 列的值。例如，在上一查询中 Year=2000 的行的小计中，`Year` 列的值为 2000。
- 从前缀中排除的列 – 空值。例如，对于由包含 Year 列的前缀生成的小计行，`Quarter` 列中的值为空值。
- 集合函数 – 所排除的列的值的集合。
  小计值是针对基础数据中的行而不是集合行而计算的。在许多情况下，例如，使用 **SUM** 或 **COUNT** 时，结果都是相同的，但是，在使用 **AVG**、**STDDEV** 和 **VARIANCE** 这样的统计函数时，结果是不同的，因而这时差别就很重要。

对 **ROLLUP** 运算符的限制如下：

- **ROLLUP** 运算符支持除 **COUNT DISTINCT** 和 **SUM DISTINCT** 以外所有可用于 **GROUP BY** 子句的集合函数。
- **ROLLUP** 只能用在 **SELECT** 语句中；子查询中无法使用 **ROLLUP**。
- 当前不支持将多个 **ROLLUP**、**CUBE** 和 **GROUP BY** 列组合在同一个 **GROUP BY** 子句中的分组规范。
- 不支持以常量表达式作为 **GROUP BY** 键。

有关表达式的一般格式，请参见《参考：构件块、表和过程》>“表达式”以及《参考：构件块、表和过程》>“SQL 语言元素”。

ROLLUP 示例 2 - 以下示例说明了 **ROLLUP** 和 **GROUPING** 的用法并显示了由 **GROUPING** 创建的一组掩码列。显示在列 S、N 和 C 中的数字 0 和 1 是由 **GROUPING** 返回的值，表示 **ROLLUP** 结果的值。程序可通过使用掩码“011”标识小计行，使用“111”标识总计行来分析此查询的结果。

```
SELECT size, name, color, SUM(quantity),
  GROUPING(size) AS S,
  GROUPING(name) AS N,
  GROUPING(color) AS C
FROM Products
```

```
GROUP BY ROLLUP(size, name, color) HAVING (S=1 or N=1 or C=1)
ORDER BY size, name, color;
```

前面的查询返回：

```
size               name           color     SUM  S  N  C
----               -----          ------     ---  -  -  -
(NULL)             (NULL)          (NULL)    496  1  1  1
Large              (NULL)          (NULL)     71  0  1  1
Large              Sweatshirt      (NULL)     71  0  0  1
Medium             (NULL)          (NULL)    134  0  1  1
Medium             Shorts          (NULL)     80  0  0  1
Medium             Tee Shirt       (NULL)     54  0  0  1
One size fits all  (NULL)          (NULL)    263  0  1  1
One size fits all  Baseball Cap    (NULL)    124  0  0  1
One size fits all  Tee Shirt       (NULL)     75  0  0  1
One size fits all  Visor           (NULL)     64  0  0  1
Small              (NULL)          (NULL)     28  0  1  1
Small              Tee Shirt       (NULL)     28  0  1  1
```

注意： 在 Rollup 示例 2 结果中，SUM 列显示为 *SUM(products.quantity)*。

ROLLUP 示例 3 - 以下示例说明了如何使用 **GROUPING** 来区分存储空值和 **ROLLUP** 运算创建的“空”值。之后，存储空值在列 `prod_id` 中显示为 [空]，而 **ROLLUP** 创建的“空”值在 `PROD_IDS` 列中由 ALL 替换，如查询中所指定。

```
SELECT year(ShipDate) AS Year,
                ProductID, SUM(quantity)AS OSum,
CASE
        WHEN GROUPING(Year) = 1
        THEN 'ALL'
        ELSE
        CAST(Year AS char(8))
END,
CASE
        WHEN GROUPING(ProductID) = 1
        THEN 'ALL'
        ELSE
        CAST(ProductIDas char(8))
END
FROM SalesOrderItems
GROUP BY ROLLUP(Year, ProductID) HAVING OSum > 36
ORDER BY Year, ProductID;
```

前面的查询返回：

```
Year  ProductID  OSum  ...(Year)...  ...(ProductID)...
---------     -------   ---   ----------   --------
NULL           NULL   28359   ALL          ALL
2000           NULL   17642   2000         ALL
2000           300     1476   2000         300
2000           301     1440   2000         301
2000           302     1152   2000         302
2000           400     1946   2000         400
2000           401     1596   2000         401
2000           500     1704   2000         500
```

```
2000            501     1572   2000          501
2000            600     2124   2000          600
2000            601     1932   2000          601
2000            700     2700   2000          700
2001            NULL   10717   2001          ALL
2001            300      888   2001          300
2001            301      948   2001          301
2001            302      996   2001          302
2001            400     1332   2001          400
2001            401     1105   2001          401
2001            500      948   2001          500
2001            501      936   2001          501
2001            600      936   2001          600
2001            601      792   2001          601
2001            700     1836   2001          700
```

ROLLUP 示例 4 - 下面的示例查询返回按年和季度汇总销售订单数的数据。

```
SELECT year (OrderDate) AS Year,
quarter(OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
GROUP BY ROLLUP (Year, Quarter)
ORDER BY Year, Quarter
```

下图说明了查询结果，并突出显示了结果集中的小计行。每个小计行在计算小计时所针对的列中的值为空值。

	Year	Quarter	Orders
1	**(NULL)**	**(NULL)**	**648**
2	**2000**	**(NULL)**	**380**
3	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	235
2	**2001**	**(NULL)**	**268**
3	2001	1	139
	2001	2	119
	2001	3	10

行 [1] 表示两年（2000、2001）和所有季度的订单总数。该行在 `Year` 列和 `Quarter` 列中的值均为空值，并且是所有列均未包括在前缀中的行。

注意： 每个 **ROLLUP** 运算返回的结果集均含有这样一行：除集合列外，每个列中均显示空值。该行表示针对其执行集合函数的每列的汇总。例如，如果 SUM 是正在讨论的集合函数，则该行将表示所有值的总和。

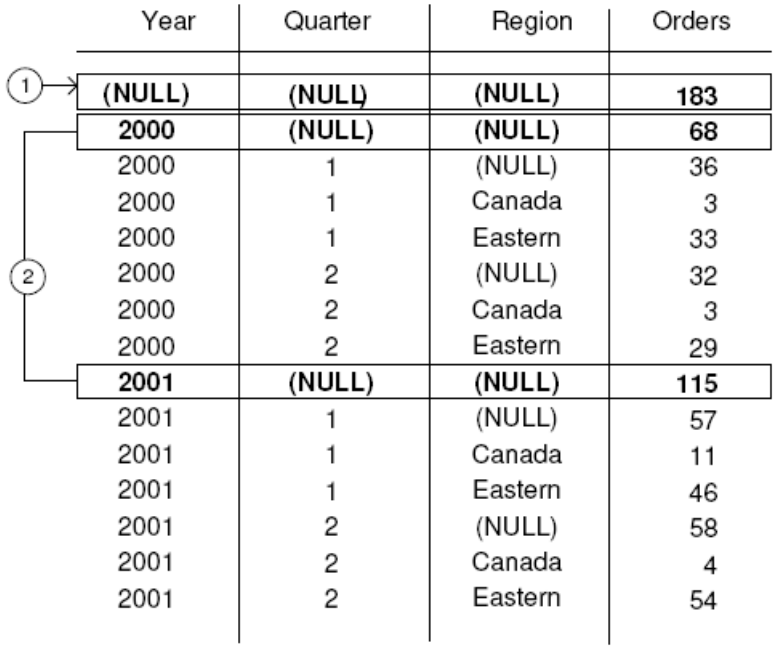
行 [2] 分别表示 2000 年和 2001 年的订单总数。这两个行的 `Quarter` 列中均为空值，原因是：该列中的值会累计，从而为 `Year` 提供小计。结果集中这种行的数目取决于 **ROLLUP** 查询中出现的变量数。

标有 [3] 的其余行通过提供两年内每个季度的订单总数而提供摘要信息。

ROLLUP 示例 5 - 此 **ROLLUP** 运算示例返回一个略加复杂的结果集，该结果集按年份、季度和区域汇总销售订单数。在此示例中，只检查第一季度和第二季度以及两个选定区域（加拿大和东部地区）。

```
SELECT year(OrderDate) AS Year, quarter(OrderDate)AS Quarter,
region, COUNT(*) AS OrdersFROM SalesOrders WHERE region IN
('Canada','Eastern') AND quarter IN (1, 2)GROUP BY ROLLUP (Year,
Quarter, Region)ORDER BY Year, Quarter, Region
```

下图说明了从上述查询得到的结果集。每个小计行在计算小计时所针对的列中的值为空值。



	Year	Quarter	Region	Orders
1	**(NULL)**	**(NULL)**	**(NULL)**	**183**
	2000	**(NULL)**	**(NULL)**	**68**
	2000	1	(NULL)	36
	2000	1	Canada	3
	2000	1	Eastern	33
2	2000	2	(NULL)	32
	2000	2	Canada	3
	2000	2	Eastern	29
	2001	**(NULL)**	**(NULL)**	**115**
	2001	1	(NULL)	57
	2001	1	Canada	11
	2001	1	Eastern	46
	2001	2	(NULL)	58
	2001	2	Canada	4
	2001	2	Eastern	54

行 [1] 是所有行的集合，在 `Year`、`Quarter` 和 `Region` 列中包含空值。此行的 `Orders` 列中的值表示加拿大和东部地区在 2000 年和 2001 年第 1 季度和第 2 季度的订单总数。

标有 [2] 的行表示加拿大和东部地区每年（2000 年和 2001 年）第 1 季度和第 2 季度的销售订单总数。这些行 [2] 的值等于行 [1] 中呈现的总数。

标有 [3] 的行按区域提供有关给定年份和季度的订单总数的数据。

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	**1**	**(NULL)**	**36**
2000	1	Canada	3
2000	1	Eastern	33
2000	**2**	**(NULL)**	**32**
2000	2	Canada	3
2000	2	Eastern	29
2001	(NULL)	(NULL)	115
2001	**1**	**(NULL)**	**57**
2001	1	Canada	11
2001	1	Eastern	46
2001	**2**	**(NULL)**	**58**
2001	2	Canada	4
2001	2	Eastern	54

标有 [4] 的行在结果集中提供了有关每年、每季度和每个区域的订单总数的数据。

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	1	(NULL)	36
2000	1	Canada	3
2000	1	Eastern	33
2000	2	(NULL)	32
2000	2	Canada	3
2000	2	Eastern	29
2001	(NULL)	(NULL)	115
2001	1	(NULL)	57
2001	1	Canada	11
2001	1	Eastern	46
2001	2	(NULL)	58
2001	2	Canada	4
2001	2	Eastern	54

Group by CUBE

GROUP BY 子句中的 **CUBE** 运算符通过以多维形式将数据分组（分组表达式）来分析数据。

CUBE 需要一个有序维度列表作为参数，它让 **SELECT** 语句计算您在查询中指定的维度组的所有可能组合的小计，并生成一个结果集来显示选定列中值的所有组合的集合。

CUBE 语法：

```
SELECT ··· [ GROUPING (column-name) ··· ] ··· GROUP BY
[ expression [,···] | CUBE ( expression [,···] ) ]
```

GROUPING 采用列名作为参数，并返回下表中所列的布尔值：

表 3. 使用 CUBE 运算符时 GROUPING 返回的值

如果结果值是	GROUPING 将返回
由 **CUBE** 运算创建的空值	1 (TRUE)
指示该行是小计所在行的空值	1 (TRUE)
并非由 **CUBE** 运算创建	0 (FALSE)
存储的空值	0 (FALSE)

当维度不是同一层次的一部分时，**CUBE** 特别有用。

此 SQL 语法...	定义以下集...
`GROUP BY CUBE (A, B, C);`	(A, B, C) (A, B) (A, C) (A) (B, C) (B) (C) ( )

对 **CUBE** 运算符的限制如下：

- **CUBE** 运算符支持所有可用于 **GROUP BY** 子句的集合函数，但 **COUNT DISTINCT** 或 **SUM DISTINCT** 当前不支持 **CUBE**。
- 逆分布分析函数 **PERCENTILE_CONT** 和 **PERCENTILE_DISC** 当前不支持 **CUBE**。

- **CUBE** 只能用在 **SELECT** 语句中；**SELECT** 子查询中无法使用 **CUBE**。
- 当前不支持将 **ROLLUP**、**CUBE** 和 **GROUP BY** 列组合在同一个 GROUP BY 子句中的 **GROUPING** 规范。
- 不支持以常量表达式作为 **GROUP BY** 键。

注意： 如果 CUBE 的大小超过临时高速缓存的大小，**CUBE** 的性能将会下降。

GROUPING 可与 **CUBE** 运算符配合使用来区分存储空值和 **CUBE** 创建的查询结果中的空值。

请参见 **ROLLUP** 运算符说明中的示例，以了解如何使用 **GROUPING** 函数解释结果。

所有 **CUBE** 运算返回的结果集均至少有一行在除集合列以外的每个列中显示空值。该行表示针对其执行集合函数的每列的汇总。

CUBE 示例 1 - 下列查询使用人口统计中的数据，包括州/省（地理位置）、性别、受教育水平，以及人员收入。第一个查询包含一个 **GROUP BY** 子句，该子句根据表 census 中列 state、gender 和 education 的值将查询结果组织成行组，并计算每一组的平均收入和总计。此查询只使用 **GROUP BY** 子句（不带 **CUBE** 运算符）来将行分组。

```
SELECT State, Sex as gender, DepartmentID,
COUNT(*),CAST(ROUND(AVG(Salary),2) AS NUMERIC(18,2))AS AVERAGEFROM
employees WHERE state IN ('MA' , 'CA')GROUP BY State, Sex,
DepartmentIDORDER BY 1,2;
```

以上查询的结果：

```
state  gender  DepartmentID  COUNT()     AVERAGE
-----  ------  -------   --------    -------- CA      F       200
 2         58650.00 CA       M       200           1         39300.00
```

如果要根据州/省、性别和受教育情况计算整个人口统计中的平均收入，并计算列 state、gender 和 education 的所有可能组合的平均收入，同时只遍历一次人口统计数据，请使用 **GROUP BY** 子句的 **CUBE** 扩展。例如，如果要计算所有州/省的所有女性的平均收入，或者要根据所受教育和地理位置计算人口统计中所有人的平均收入，请使用 **CUBE** 运算符。

当 **CUBE** 计算组时，系统会为计算其组的列生成空值。必须使用 **GROUPING** 函数确定某个空值是存储在数据库中的空值，还是从 **CUBE** 生成的空值。如果已将指定列合并到更高级别的组中，则 **GROUPING** 函数返回 1。

CUBE 示例 2 - 以下查询说明了如何将 **GROUPING** 函数与 **GROUP BY CUBE** 结合使用。

```
SELECT case grouping(State) WHEN 1 THEN 'ALL' ELSE StateEND AS
c_state, case grouping(sex) WHEN 1 THEN 'ALL'ELSE Sex end AS
c_gender, case grouping(DepartmentID)WHEN 1 THEN 'ALL' ELSE
cast(DepartmentID as char(4)) endAS c_dept, COUNT(*),
CAST(ROUND(AVG(salary),2) ASNUMERIC(18,2))AS AVERAGEFROM employees
WHERE state IN ('MA' , 'CA')GROUP BY CUBE(state, sex,
DepartmentID)ORDER BY 1,2,3;
```

下面显示了此查询的结果。系统将按照查询中的指定，在小计行中将 **CUBE** 生成的指示小计行的空值替换为 ALL。

```
c_state   c_gender  c_dept    COUNT()   AVERAGE
-------   --------  -------    -----     --------
ALL       ALL        200         3       52200.00
ALL       ALL        ALL         3       52200.00
ALL       F          200         2       58650.00
ALL       F          ALL         2       58650.00
ALL       M         200          1       39300.00
ALL       M         ALL          1       39300.00
CA        ALL       200          3       52200.00
CA        ALL       ALL          3       52200.00
CA        F         200          2       58650.00
CA        F         ALL          2       58650.00
CA        M         200          1       39300.00
CA        M         ALL          1       39300.00
```

CUBE 示例 3 - 在此示例中，查询返回一个汇总订单总数的结果集，然后按年和季度计算订单数小计。

注意： 随着待比较变量数量的增加，CUBE 的计算成本呈指数增长。

```
SELECT year (OrderDate) AS Year, quarter(OrderDate) AS Quarter, COUNT
(*) OrdersFROM SalesOrdersGROUP BY CUBE (Year, Quarter)ORDER BY Year,
Quarter
```

下图显示的是从查询得到的结果集。在结果集中突出显示小计行。每个小计行在计算小计所针对的列中的值为空值。

	Year	Quarter	Orders
1	**(NULL)**	**(NULL)**	**648**
2	**(NULL)**	**1**	**226**
	(NULL)	**2**	**196**
	(NULL)	**3**	**101**
	(NULL)	**4**	**125**
3	**2000**	**(NULL)**	**380**
	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
3	**2001**	**(NULL)**	**268**
	2001	1	139
	2001	2	119
	2001	3	10

第一个突出显示的行 [1] 表示两年和所有季度内的订单总数。`Orders` 列中的值是每个标有 [3] 的行中值的总和。它也是标有 [2] 的行中的四个值的总和。

下一组突出显示的行 [2] 按季度显示两年内的订单总数。标有 [3] 的两个行分别显示 2000 年和 2001 年的所有季度的订单总数。

分析函数

Sybase IQ 既提供简单集合函数，又提供窗口集合函数，利用这些函数，可以通过一个 SQL 语句执行复杂数据分析。

可以使用这些函数计算查询（如“What is the quarterly moving average of the Dow Jones Industrial average”或“List all employees and their cumulative salaries for each department”）的结果。可计算各个区间的移动平均值和累计总和，并且可对集合和秩进行分区以便集合计算在分区值改变时重置。在单个查询表达式的范围内，您可以定义多个不同的 OLAP 函数，每个函数都有它自己的任意分区规则。分析函数可分为两种类别：

- 简单集合函数（如 **AVG**、**COUNT**、**MAX**、**MIN** 和 **SUM**）可对数据库的一组行中的数据进行汇总。这些组是使用 **SELECT** 语句的 **GROUP BY** 子句构成的。
- 采用一个参数的一元统计集合函数包括 **STDDEV**、**STDDEV_SAMP**、**STDDEV_POP**、**VARIANCE**、**VAR_SAMP** 和 **VAR_POP**。

简单集合类别和一元集合类别汇总数据库中某一组行的数据，并且可在处理结果集时与窗口规范配合使用来计算结果集的移动窗口。

注意： 集合函数 **AVG**、**SUM**、**STDDEV**、**STDDEV_POP**、**STDDEV_SAMP**、**VAR_POP**、**VAR_SAMP** 和 **VARIANCE** 不支持二进制数据类型 BINARY 和 VARBINARY。

简单集合函数

简单集合函数（如 **AVG**、**COUNT**、**MAX**、**MIN** 和 **SUM**）可对数据库的一组行中的数据进行汇总。

这些组是使用 **SELECT** 语句的 **GROUP BY** 子句构成的。只能在选择列表中以及 **SELECT** 语句的 **HAVING** 和 **ORDER BY** 子句中使用这些集合。

注意： 除 Grouping() 函数以外，简单集合和一元集合都可以在将 <window clause> 纳入 SQL 查询规范（一个*窗口*，可在处理结果集时在概念上为该结果集创建移动窗口）的窗口函数中使用。

请参见《参考：构件块、表和过程》>“SQL 函数”>“集合函数”。

窗口化

OLAP 的 ANSI SQL 扩展的一个主要特征是：它具有一个名为*窗口*的结构。利用此窗口化扩展，用户可以将查询（或查询的逻辑分区）的结果集分成名为“分区”的行组，并确定要与当前行集合的行的子集。

您可以对窗口使用三类窗口函数：排名函数、行计算函数和窗口集合函数。

```
<WINDOWED TABLE FUNCTION TYPE> ::=
  <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
```

```
  | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
  | <WINDOW AGGREGATE FUNCTION>
```

窗口化扩展通过窗口名称或规范指定窗口函数的类型，并应用于单个查询表达式范围内的分区结果集。窗口分区是由查询返回的行的子集，由一个特殊的 **OVER** 子句中的一个或多个列定义：

```
olap_function() OVER (PARTITION BY col1, col2...)
```

使用窗口化操作，可以建立信息，诸如在行的分区中排列每行、在某个分区内的行中分布值，以及类似操作。利用窗口化操作，还可对数据计算移动平均值和总数，从而增强对数据及其对操作的影响进行评估的能力。

OLAP 窗口的三个基本部分

OLAP 窗口包含三个基本方面：窗口分区、窗口排序和窗口构架。在任何时刻，每一方面都会对窗口中显示的特定数据行产生重大影响。同时，OLAP **OVER** 子句可利用三项独特的功能将 OLAP 函数与其它分析函数或报告函数区分开：

- 定义窗口分区（**PARTITION BY** 子句）。
- 对分区中的行进行排序（**ORDER BY** 子句）。
- 定义窗口构架（ROWS/RANGE 规范）。

若要指定多个窗口函数，并避免出现冗余窗口定义，可以指定 OLAP 窗口规范的名称。在这种用法中，关键字 **WINDOW** 后面至少跟有一个窗口定义，窗口定义之间用逗号分隔开。窗口定义包含窗口在查询中使用的名称以及窗口规范中的详细信息，利用它，可以定义窗口分区、窗口排序和窗口构架：

```
<WINDOW CLAUSE> ::= <WINDOW DEFINITION LIST>
```

```
<WINDOW DEFINITION LIST> ::=
  <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
  } . . . ]
```

```
<WINDOW DEFINITION> ::=
  <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

```
<WINDOW SPECIFICATION DETAILS> ::=
  [ <EXISTING WINDOW NAME> ]
  [ <WINDOW PARTITION CLAUSE> ]
  [ <WINDOW ORDER CLAUSE> ]
  [ <WINDOW FRAME CLAUSE> ]
```

对于窗口分区中的每一行，用户可以定义窗口构架，这可能会改变用于对分区的当前行执行任何计算的特定行范围。当前行提供了可用于确定窗口构架的起点和终点的参照点。

窗口规范可以基于一定数量的物理行（使用定义 ROWS 的窗口构架单元的窗口规范）或数值的逻辑区间（使用定义 RANGE 的窗口构架单元的窗口规范）。

在 OLAP 窗口化操作中，可以使用以下函数类别：

- 排名函数

- 窗口化集合函数
- 统计集合函数
- 分布函数

另请参见
- 分布函数 （第 53 页）
- OLAP 优点 （第 20 页）
- OLAP 计算 （第 20 页）
- 排名函数 （第 43 页）
- 统计集合函数 （第 49 页）
- 窗口化集合函数 （第 47 页）
- OLAP 函数的 BNF 语法 （第 67 页）

窗口分区

窗口分区是指使用 **PARTITION BY** 子句拆分用户指定的结果集（输入行）。

分区由一个或多个用逗号分隔的值表达式定义。分区数据也是隐式排序的，缺省排序顺序是升序 (ASC)。

```
<WINDOW PARTITION CLAUSE> ::=
  PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

如果未指定窗口分区子句，则会将输入作为单个分区处理。

注意： 术语*分区*像在分析函数中使用时一样，仅仅是指使用 **PARTITION BY** 子句将结果行集拆分开。

可根据任意表达式定义窗口分区。此外，因为窗口分区在 GROUPING 之后发生（如果指定了 **GROUP BY** 子句），所以，任何集合函数（如 **SUM**、**AVG** 和 **VARIANCE**）的结果都可以在分区表达式中使用。因此，*除了* **GROUP BY** 和 **ORDER BY** 子句外，利用分区，也可以执行分组和排序操作；例如，您可以构造通过集合函数来计算集合函数（如特定数量的最大 **SUM**）的查询。

即使没有 **GROUP BY** 子句，也可以指定 **PARTITION BY** 子句。

另请参见
- 窗口构架 （第 36 页）
- 窗口排序 （第 35 页）

窗口排序

窗口排序是指使用 window order 子句（包含一个或多个用逗号分隔的值表达式）排列每个窗口分区中的结果（行）。

如果未指定 window order 子句，则可以按任意顺序处理输入行。

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

OLAP window order 子句不同于 **ORDER BY** 子句，后者可以附加到非窗口查询表达式中。

例如，OLAP 函数中的 **ORDER BY** 子句通常定义用于对窗口分区中的行进行排序的表达式；但是，您可以使用不带 **PARTITION BY** 子句的 **ORDER BY** 子句，在这种情况下，排序规范可确保使用 OLAP 函数来对中间结果集进行有意义的（并且是预期的）排序。

排序规范是 OLAP 函数的排列系列的前提条件；用于标识排列值的测量标准的不是函数参数本身，而是 **ORDER BY** 子句。如果是 OLAP 集合，则一般不需要 **ORDER BY** 子句，但该子句是定义窗口构架的前提条件。这是因为，必须先对分区行进行排序，然后才可以为每个构架计算相应的集合值。

ORDER BY 子句包括用于定义升序排序和降序排序的语义，以及空值的处理规则。缺省情况下，OLAP 函数采用升序，而最小测量值的排名为 1。

虽然此行为与 **ORDER BY** 子句（位于 **SELECT** 语句的末尾）的缺省行为一致，但对于大多数有序计算，它都是与直觉相反的。OLAP 计算通常需要降序，而最大测量值的排名为 1；必须使用带 DESC 关键字的 **ORDER BY** 子句显式声明此要求。

注意： 排名函数需要 <window order clause>，因为它们是仅针对排序输入定义的。就像使用 <query specification> 中的 <order by clause> 一样，缺省排序顺序是升序。

使用 RANGE 的 <window frame unit> 时也要求存在 <window order clause>。如果是 RANGE，则 <window order clause> 只能包括一个表达式。

另请参见

- 窗口构架 （第 36 页）
- 窗口分区 （第 35 页）

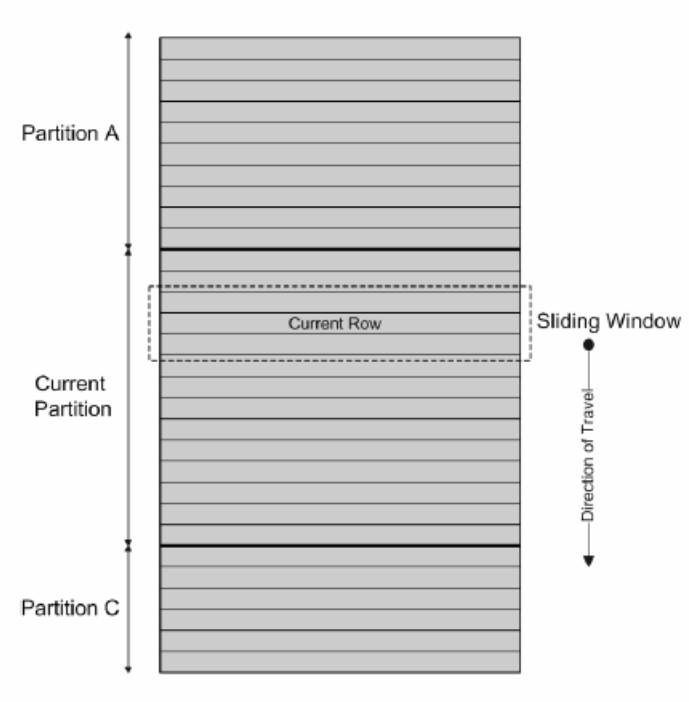
窗口构架

对于非秩集合 OLAP 函数，可以使用 window frame 子句定义一个窗口构架，该子句指定窗口的开始位置和结束位置（相对于当前行）。

```
<WINDOW FRAME CLAUSE> ::=
  <WINDOW FRAME UNIT>
  <WINDOW FRAME EXTENT>
```

计算此 OLAP 函数时针对的是移动构架的内容而不是整个分区的固定内容。根据定义，分区具有开始行和结束行，而窗口构架从分区的起点滑至终点。

图 3： 含分区输入的三行移动窗口



UNBOUNDED PRECEDING 和 FOLLOWING
窗口构架可由未受限制的集合组（扩展回到分区的开始位置 (UNBOUNDED PRECEDING) 或扩展到分区的结束位置 (UNBOUNDED FOLLOWING)）定义。

UNBOUNDED PRECEDING 包括分区中当前行之前的所有行，对于这些行，可使用 ROWS 或 RANGE 指定。UNBOUNDED FOLLOWING 包括分区中当前行之后的所有行，对于这些行，可使用 ROWS 或 RANGE 指定。

值 FOLLOWING 指定当前行后面的行的范围或数量。如果指定了 ROWS，则该值是一个表示行数的正整数。如果指定了 RANGE，则窗口包括所有小于当前行加指定数值的行。如果是 RANGE，则窗口值的数据类型必须与 **ORDER BY** 子句的排序键表达式的类型相当。只能有一个排序键表达式，并且排序键表达式的数据类型必须允许相加。

值 PRECEDING 指定当前行前面的行的范围或数量。如果指定了 ROWS，则该值是一个表示行数的正整数。如果指定了 RANGE，则窗口包括所有小于当前行减去指定数值的行。如果是 RANGE，则窗口值的数据类型必须与 **ORDER BY** 子句的排序键表达式的类型相当。只能有一个排序键表达式，并且排序键表达式的数据类型必须允许相减。如果第一个绑定组为 CURRENT ROW 或值 FOLLOWING，则不能在第二个绑定组中指定此子句。

组合 BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING 提供整个分区的集合，且不需要构造到分组查询的连接。整个分区的集合也称为报告聚合。

CURRENT ROW 概念

在物理集合组中，根据行相对于当前行的位置来包括或排除行（通过统计相邻行的数目）。当前行仅仅是查询的中间结果中下一行的参照。如果当前行前进，则会根据窗口中的新行集重新计算窗口。对于窗口中包括的当前行，没有任何要求。

如果未指定 window frame 子句，则缺省窗口构架取决于是否指定了 window order 子句：

- 如果窗口规范包含 window order 子句，且窗口的起点是 **UNBOUNDED PRECEDING**，终点是 **CURRENT ROW**，则定义适合用于计算累计值的大小可变的窗口。
- 如果窗口规范不包含 window order 子句，且窗口的起点是 **UNBOUNDED PRECEDING**，终点是 **UNBOUNDED FOLLOWING**，则定义大小固定的窗口（不考虑当前行）。

注意： window frame 子句不能与排名函数一起使用。

您还可以通过指定基于行（行规范）或基于值（范围规范）的窗口构架单元来定义窗口。

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

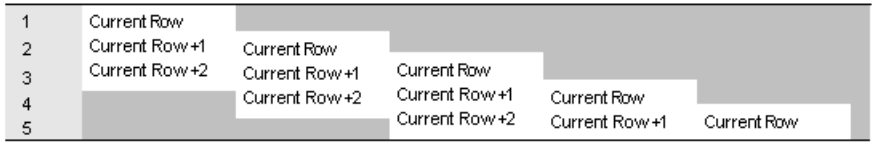
```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME
BETWEEN>
```

当窗口构架范围指定 **BETWEEN** 时，它会显式提供窗口构架的开始位置和结束位置。

如果窗口构架范围仅指定这两个值中的一个，则另一个值缺省为 **CURRENT ROW**。

基于行的窗口构架 - 在示例中，行 [1] 至 [5] 表示分区；随着 OLAP 窗口构架的向前滑动，每个行都可成为当前行。构架是指“Between Current Row And 2 Following”，因此每个构架都最多包括三行，最少包括一行。当构架到达分区的末尾时，只包括当前行。带阴影的区域表示每一步从构架中排除的行。

图 4： 基于行的窗口构架

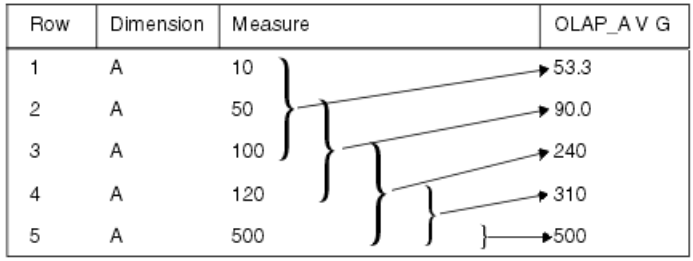


窗口构架实施下列规则：

- 当行 [1] 为当前行时，系统将排除行 [4] 和行 [5]。
- 当行 [2] 为当前行时，将排除行 [5] 和行 [1]。

- 当行 [3] 为当前行时，系统将排除行 [1] 和行 [2]。
- 当行 [4] 为当前行时，将排除行 [1]、[2] 和 [3]。
- 当行 [5] 为当前行时，系统将排除行 [1]、[2]、[3] 和 [4]。

下图将这些规则应用到一组特定值，并显示将为每一行计算的 OLAP **AVG** 函数。滑动计算生成区间为三行（或更少）的移动平均值，具体取决于哪一行是当前行：



下面的示例说明了一个滑动窗口：

```
SELECT dimension, measure,
  AVG(measure) OVER(partition BY dimension
    ORDER BY measure
    ROWS BETWEEN CURRENT ROW and 2 FOLLOWING)
    AS olap_avg
FROM ...
```

平均值按以下方式计算：

- 行 [1] = (10 + 50 + 100)/3
- 行 [2] = (50+ 100 + 120)/3
- 行 [3] = (100 + 120 + 500)/3
- 行 [4] = (120 + 500 + NULL)/3
- 行 [5] =（500 + 空值 + 空值）/3

将为结果集中的所有后续分区执行相似计算（如 B、C 等）。

如果当前窗口中没有行，则结果为空值，但对于 **COUNT** 则例外。

另请参见

- 窗口排序 （第 35 页）
- 窗口分区 （第 35 页）

ROWS

窗口构架单元 ROWS 通过指定当前行前面或后面的行的数量来定义窗口，当前行充当确定窗口开始位置和结束位置的参照点。

每个分析计算都基于分区中的当前行。要为以行表示的窗口生成确定性结果，排序表达式应该是唯一的。

所有窗口构架的参照点都是当前行。使用 SQL/OLAP 语法，可以将基于行的窗口构架定义为当前行前面和/或后面的任意数量的行。

以下列表说明了常见的窗口构架单元示例：

- Rows Between Unbounded Preceding and Current Row - 指定以每个分区的开始位置作为起点并以当前行作为终点的窗口，经常用于构造计算累计结果（如累计总和）的窗口。
- Rows between unbounded preceding and unbounded following - 指定跨整个分区的固定窗口（不考虑当前行）。因此，窗口集合函数的值在分区的每一行中都是相同的。
- Rows between 1 preceding and 1 following - 指定跨三个相邻行（当前行前后各一行）的大小固定的移动窗口。您可以使用此窗口构架单元执行计算，例如，计算 3 天或 3 个月的移动平均值。

  请注意，由于窗口值之间存在间距，因此在使用 ROWS 时可能会生成没有意义的结果。如果值集是不连续的，请考虑使用 RANGE 而不是 ROWS，因为基于 RANGE 的窗口定义会自动处理含重复值的相邻行，并且，如果范围中存在间距，则不会包括任何其它行。

  注意： 如果是移动窗口，则假设在输入中，包含空值的行位于第一行之前、最后一行之后。这意味着，在一个包含 3 行的移动窗口中，输入中最后一行（当前行）的计算将包括紧挨在它前面的一行和空值。
- Rows between current row and current row - 将窗口限制为仅含当前行。
- Rows Between 1 Preceding and 1 Preceding - 指定仅含上一行（相对于当前行而言）的单行窗口。此构造与仅基于当前行计算值的另一窗口函数组合使用，使得您可以轻松地计算相邻行间的增量（即值的差额）。

另请参见

- RANGE （第 40 页）

RANGE

基于范围的窗口构架 - SQL/OLAP 语法支持另一种窗口构架，这种构架的限制是根据一组基于值（或基于范围）的行定义的，而不是根据一系列特定行定义的。

基于值的窗口构架定义窗口分区中包含特定范围的数值的行。OLAP 函数的 **ORDER BY** 子句定义应用范围规范的数值列（相对于当前行的该列中的值）。范围规范与行规范使用的语法相同，但是语法的解释方式不同。

窗口构架单元 RANGE 可定义一个窗口构架，通过查找排序列的值在指定值范围内的行（相对于当前行），可以确定该构架的内容。这称为窗口构架的逻辑偏移，您可以使用常量（如“3 preceding”）或任何计算结果为数值常量的表达式指定该偏移。如果所使用的窗口是用 RANGE 定义的，则 **ORDER BY** 子句中只能有一个数值表达式。

注意： **ORDER BY** 键必须是 RANGE 窗口构架中的数值数据。

例如，可将构架定义为一组含 *year* 值（当前行的年份之前或之后的数年）的行：

```
ORDER BY year ASC range BETWEEN CURRENT ROW and 1 PRECEDING
```

在上面的示例查询中，“1 preceding”表示当前行的 *year* 值减去 1。

这种范围规范是包含性的。如果当前行的 *year* 值为 2000，则窗口分区中所有 year 值为 2000 和 1999 的行都符合成为构架内容的条件，无论这些行在分区中处于哪个物理位置都是如此。用于包括和排除基于值的行的规则与应用于基于行的构架的规则非常不同，后者完全依赖于行的物理顺序。

如果是在 OLAP **AVG()** 计算环境中，则下面的部分结果集可进一步说明基于值的窗口构架的概念。重申一遍，构架包括符合以下条件的行：

- 与当前行具有相同的 year 值
- year 值等于当前行减去 1

Row	Dimension	Year	Measure	Olap_avg
1	A	1999	10000	10000
2	A	2001	5000	3000
3	A	2001	1000	3000
4	A	2002	12000	5250
5	A	2002	3000	5250

以下查询说明了基于范围的窗口定义：

```
SELECT dimension, year, measure,
  AVG(measure) OVER(PARTITION BY dimension
    ORDER BY year ASC
    range BETWEEN CURRENT ROW and 1 PRECEDING)
    as olap_avg
FROM ...
```

平均值按以下方式计算：

- 行 [1] = 1999；排除行 [2] 至 [5]；AVG = 10,000/1
- 行 [2] = 2001；排除行 [1]、[4] 和 [5]；AVG = 6,000/2
- 行 [3] = 2001；排除行 [1]、[4] 和 [5]；AVG = 6,000/2
- 行 [4] = 2002；排除行 [1]；AVG = 21,000/4
- 行 [5] = 2002；排除行 [1]；AVG = 21,000/4

基于值的构架的升序和降序 - 用于基于值的窗口构架的 OLAP 函数的 **ORDER BY** 子句不仅标识范围规范所基于的数值列，而且还声明 **ORDER BY** 值的排序顺序。以下规范受它之前的排序顺序（ASC 或 DESC）约束：

```
RANGE BETWEEN CURRENT ROW AND n FOLLOWING
```

规范 *n* FOLLOWING 表示：

- 如果分区按缺省升序 (ASC) 排序，则加上 *n*
- 如果分区按降序 (DESC) 排序，则减去 *n*

例如，假设 `year` 列包含四个不同的值（从 1999 到 2002）。下表在左侧显示这此值的缺省升序，在右侧显示这些值的降序：

ORDER BY year ASC	ORDER BY year DESC
1999	2002
2000	2001
2001	2000
2002	1999

如果当前行是 1999 并且按照如下所示指定构架，则包含值 1999 和 1998 的行（在表中不存在）会包括在构架中：

```
ORDER BY year DESC range BETWEEN CURRENT ROW and 1 FOLLOWING
```

注意： **ORDER BY** 值的排序顺序是测试哪些行可以包括在基于值的构架中的关键部分；仅仅依赖于数值，无法确定是排除还是包括行。

使用未受限制的窗口 - 以下查询会生成一个结果集，结果集中包含所有产品以及所有产品的总量：

```
SELECT id, description, quantity,
  SUM(quantity) OVER () AS total
FROM products;
```

计算相邻行之间的增量 - 如果使用两个窗口（一个在当前行上，另一个在上一行上），则可以直接计算相邻行之间的增量（即变化）。

```
SELECT EmployeeID, Surname, SUM(salary)
OVER(ORDER BY BirthDate rows between current row and current row)
AS curr, SUM(Salary)
OVER(ORDER BY BirthDate rows between 1 preceding and 1 preceding)
AS prev, (curr-prev) as delta
FROM Employees
WHERE State IN ('MA', 'AZ', 'CA', 'CO') AND DepartmentID>10
ORDER BY EmployeeID, Surname;
```

以上查询的结果：

```
EmployeeID  Surname    curr       prev        delta
------              ---------    --------  ---------  ----------
148            Jordan                  51432.000191
Bertrand                  29800.000            39300.000
-9500.000278          Melkisetian            48500.000
42300.000             6200.000299            Overbey
39300.000             41700.750              -2400.750318
Crow                  41700.750              45000.000
-3299.250586            Coleman                42300.000
46200.000             -3900.000690           Poitras
46200.000             29800.000              16400.000703
Martinez                 55500.800              51432.000
4068.800949           Savarino                 72300.000
55500.800             16799.2001101            Preston
37803.000             48500.000              -10697.0001142
Clark                 45000.000              72300.000
-27300.000
```

虽然使用了窗口函数 **SUM()**，但总和只包含当前行或上一行的薪水值，这是由于窗口的指定方式而导致的。此外，结果中第一行的 prev 值为空值（因为它没有前项）；因此，delta 也为空值。

在上面的每个示例中，用于 **OVER()** 子句的函数是 **SUM()** 集合函数。

另请参见

- ROWS （第 39 页）

显式窗口子句和行内窗口子句

SQL OLAP 提供了两种方式来在查询中指定窗口：

- 使用 **HAVING** 子句后面的显式窗口子句，可以定义窗口。您可以在调用 OLAP 函数时通过指定名称来引用这些窗口子句定义的窗口，如：

  ```
  SUM ( ...) OVER w2
  ```

- 使用行内窗口规格，可以通过查询表达式的 **SELECT** 列表定义窗口。使用此功能，可以在跟在 **HAVING** 子句后面的窗口子句中定义窗口，然后利用窗口函数调用按名称引用它们，或将它们与函数调用一起定义。

  注意： 如果使用行内窗口规格，则无法命名窗口。一个 **SELECT** 列表中的两个或更多个使用相同窗口的窗口函数调用都必须引用在窗口子句中定义的命名窗口，或者必须以冗余方式定义它们的行内窗口。

窗口函数示例 - 下面的示例显示了一个窗口函数。查询返回一个结果集，该结果集按部门将数据分区，然后提供雇员薪水的累计汇总（从在公司工作时间最长的雇员开始）。结果集仅包括住在马萨诸塞州的雇员。列 Sum_Salary 提供雇员薪水的累计总计。

```
SELECT DepartmentID, Surname, StartDate, Salary,SUM(Salary) OVER
(PARTITION BY DepartmentID ORDER BYstartdate rows between unbounded
preceding andcurrent row) AS sum_salaryFROM EmployeesWHERE State IN
('CA') AND DepartmentID IN (100, 200)ORDER BY DepartmentID;
```

以下结果集是按部门分区的。

```
DepartmentID  Surname   start_date      salary    sum_salary
------------               -----------  ------------
-----------     -----------
200                         Overbey                 1987-02-19
39300.000       39300.000
200                       Savarino                1989-11-07
72300.000      111600.000
200                       Clark                   1990-07-21
45000.000      156600.000
```

排名函数

使用排名函数，可以按排列顺序编译数据集中值的列表，以及编写执行请求的单语句 SQL 查询（如“Name the top 10 products shipped this year by total sales”或“Give the top 5% of salespersons who sold orders to at least 15 different companies”）。

SQL/OLAP 定义五个属于排名函数类别的函数：

```
<RANK FUNCTION TYPE> ::=
  RANK | DENSE_RANK | PERCENT_RANK | ROW_NUMBER | NTILE
```

使用排名函数，可以根据在查询中指定的顺序为结果集中的每一行计算秩值。例如，一名销售经理可能需要确定公司中的最领先或最落后的销售人员，业绩最好或最差的销售区域，或者销售情况最好或最差的产品。排名函数可以提供此信息。

另请参见

- 分布函数 （第 53 页）
- OLAP 优点 （第 20 页）
- OLAP 计算 （第 20 页）
- 统计集合函数 （第 49 页）
- 窗口化 （第 33 页）
- 窗口化集合函数 （第 47 页）
- OLAP 函数的 BNF 语法 （第 67 页）

RANK() 函数

RANK 函数返回一个数字，该数字表示当前行在行分区中各行间的秩（由 **ORDER BY** 子句定义）。

分区中第一行的秩为 1，包含 25 个行的分区的最后一个秩为 25。**RANK** 被指定为语法转换，这表示实施可以选择将 **RANK** 实际转换为它的等同项，否则，它仅能返回与转换返回的结果等同的结果。

在下面的示例中，ws1 表示定义名为 w1 的窗口的窗口规范。

```
RANK() OVER ws
```

等效于：

```
( COUNT (*) OVER ( ws RANGE UNBOUNDED PRECEDING )
- COUNT (*) OVER ( ws RANGE CURRENT ROW ) + 1 )
```

RANK 函数的转换使用逻辑集合 (RANGE)。因此，两个或更多个绑定的（即在排序列中具有相等值）的记录将具有相同的秩。分区中下一个具有不同值的组的秩将比绑定行的秩至少大 1。例如，如果有一些行的排序列值为 10、20、20、20、30，则第一行的秩为 1，第二行的秩为 2，第三行和第四行的秩也为 2，但第五行的秩为 5。没有秩为 3 或 4 的行。此算法有时也称为稀疏排名。

另请参见《参考：构件块、表和过程》>“SQL 函数”>“RANK 函数 [分析]”。

DENSE_RANK() 函数

DENSE_RANK 会返回没有间距的秩值。

绑定行的值仍旧相等，但行的秩表示在排序列中具有相等值的行的集群位置，而不是各个行的位置。正如在 **RANK** 示例中，行排序列值是 10、20、20、20、30，第一行

的秩仍旧是 1，而第二行的秩仍旧是 2，第三行和第四行的秩也是相同的。但是，最后一行的秩是 3 而不是 5。

DENSE_RANK 也是通过语法转换计算的。

```
DENSE_RANK() OVER ws
```

等效于：

```
COUNT ( DISTINCT ROW ( expr_1, . . ., expr_n ) )
 OVER ( ws RANGE UNBOUNDED PRECEDING )
```

在上面的示例中，*expr_1* 至 *expr_n* 代表窗口 w1 的排序规范列表中值表达式的列表。

另请参见《参考：构件块、表和过程》>“SQL 函数”>“DENSE_RANK 函数 [分析]”。

PERCENT_RANK() 函数

PERCENT_RANK 函数计算秩的百分比，而不是分数，并返回一个介于 0 和 1 之间的小数值。

PERCENT_RANK 返回行的相对秩，这个数字指出当前行在它所在的窗口分区中的相对位置。例如，在包含 10 个在排序列中具有不同值的行的分区中，将为第三行提供 **PERCENT_RANK** 值 0.222……，原因是您涵盖了分区的第一行后面的 2/9（22.222……%）个行。行的 **PERCENT_RANK** 是指行的 **RANK** 减去一，再除以分区中的行数减去一，如以下示例中所示（其中“ANT”代表近似数值类型，如 REAL 或 DOUBLE PRECISION）。

```
PERCENT_RANK() OVER ws
```

等效于：

```
CASE
  WHEN COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
  PRECEDING AND UNBOUNDED FOLLOWING ) = 1
  THEN CAST (0 AS ANT)
  ELSE
   ( CAST ( RANK () OVER ( ws ) AS ANT ) -1 /
   ( COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
   PRECEDING AND UNBOUNDED FOLLOWING ) - 1 )
END
```

另请参见《参考：构件块、表和过程》>“SQL 函数”>“PERCENT_RANK 函数 [分析]”。

ROW_NUMBER() 函数

ROW_NUMBER 函数为每一行返回一个唯一行号。

如果您定义窗口分区，**ROW_NUMBER** 就会在每个分区中从 1 开始对行进行编号，每行递增 1。如果您不指定窗口分区，**ROW_NUMBER** 就会对整个结果集进行编号，从 1 到表的总基数。

ROW_NUMBER 函数语法为：

```
ROW_NUMBER( ) OVER ( [PARTITION BY
window partition] ORDER BY
window ordering)
```

ROW_NUMBER 不需要参数，但您必须指定括号。

PARTITION BY 子句是可选的。**OVER** (**ORDER_BY**) 子句不能包含窗口构架 **ROWS**/**RANGE** 规范。

排名示例

下面是一些排名函数示例：

排名示例 1 - 下面的 SQL 查询查找加利福尼亚州的男女雇员并按薪水以降序排列他（她）们。

```
SELECT Surname, Sex, Salary, RANK() OVER (
ORDER BY Salary DESC) as RANK FROM Employees
WHERE State IN ('CA') AND DepartmentID =200
ORDER BY Salary DESC;
```

以上查询的结果：

```
Surname      Sex        Salary    RANK
-------      ---        ------    ----
Savarino      F          72300.000  1
Clark         F          45000.000  2
Overbey       M          39300.000  3
```

排名示例 2 - 使用上一示例中的查询，您可以按性别划分数据，从而更改数据。下面的示例按薪水以降序排列雇员并按性别将雇员分区：

```
SELECT Surname, Sex, Salary, RANK() OVER (PARTITION BY Sex
ORDER BY Salary DESC) AS RANK FROM Employees
WHERE State IN ('CA', 'AZ') AND DepartmentID IN (200, 300)
ORDER BY Sex, Salary DESC;
```

以上查询的结果：

```
Surname           Sex      Salary     RANK
-------           ---      ---------  ----
Savarino           F        72300.000  1
Jordan             F        51432.000  2
Clark              F        45000.000  3
Coleman            M        42300.000  1
Overbey            M        39300.000  2
```

排名示例 3 - 此示例查找加利福尼亚州和德克萨斯州的一些女雇员并根据薪水以降序排列她们。**PERCENT_RANK** 函数按降序提供累计总计。

```
SELECT Surname, Salary, Sex, CAST(PERCENT_RANK() OVER
(ORDER BY Salary DESC) AS numeric (4, 2)) AS RANK
FROM Employees WHERE State IN ('CA', 'TX') AND Sex ='F'
ORDER BY Salary DESC;
```

以上查询的结果：

```
Surname          salary    sex            RANK
---------       ---------    ---    ----------
Savarino        72300.000    F            0.00
Smith           51411.000    F            0.33
Clark           45000.000    F            0.66
Garcia          39800.000    F            1.00
```

排名示例 4 - 您可以使用 **PERCENT_RANK** 函数在数据集中查找最高或最低的百分点。此查询会返回其薪水在数据集的最高的五个百分点之内的男雇员。

```
SELECT * FROM (SELECT Surname, Salary, Sex,
CAST(PERCENT_RANK() OVER (ORDER BY salary DESC) as
numeric (4, 2)) AS percent
FROM Employees WHERE State IN ('CA') AND sex ='F' ) AS
DT where percent > 0.5
ORDER BY Salary DESC;
```

以上查询的结果：

```
Surname           salary     sex      percent
---------       ----------     ---    ---------
Clark                            45000.000     F      1.00
```

排名示例 5 - 此示例使用 **ROW_NUMBER** 函数为所有窗口分区中的每一行返回行号。查询按照部门 ID 对 `Employees` 表进行分区，并按照开始日期对每个分区中的行进行排序。

```
SELECT DepartmentID dID, StartDate, Salary ,
ROW_NUMBER()OVER(PARTITION BY dID ORDER BY StartDate)
FROM  Employees ORDER BY 1,2;
```

以上查询的结果为：

```
dID         StartDate      Salary       Row_number()
========    ===========    ==========   =============
  100       1984-08-28     47500.000        1
  100       1985-01-01     62000.500        2
  100       1985-06-17     57490.000        3
  100       1986-06-07     72995.000        4
  100       1986-07-01     48023.690        5
  ...       ...            ...              ...
  200       1985-02-03     38500.000        1
  200       1985-12-06     54800.000        2
  200       1987-02-19     39300.000        3
  200       1987-07-10     49500.000        4
  ...       ...            ...              ...
  500       1994-02-27      24903.000       9
```

窗口化集合函数

使用窗口化集合函数，可以在同一查询中处理多个集合级别。

例如，可以列出花销小于平均值的所有季度。您可以使用集合函数（包括简单集合函数 **AVG**、**COUNT**、**MAX**、**MIN** 和 **SUM**）将可能在语句中的不同级别计算出的结果放置在

同一行。通过这样放置，可以将集合值与组中的明细行进行比较，且不需要连接或相关子查询。

使用这些函数，还可以将非集合值与集合值进行比较。例如，有些客户所订购产品的数量超过了某产品在指定年份内的平均数量，销售人员可能需要编辑所有这些客户的列表，或者，经理可能需要将雇员的薪水与部门的平均薪水进行比较。

如果查询在 **SELECT** 语句中指定 **DISTINCT**，则在窗口运算符之后应用 **DISTINCT** 运算。窗口运算符的计算是在处理 **GROUP BY** 子句之后、计算 **SELECT** 列表项和查询的 **ORDER BY** 子句之前进行的。

窗口化集合示例 1 - 此查询返回一个结果集（按年分区），该结果集显示了销售量高于平均销售量的产品的列表。

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID ASDept,
CAST(Salary AS numeric(10,2) ) AS Sal,CAST(AVG(Sal) OVER(PARTITION
BY DepartmentID) ASnumeric(10, 2)) AS Average,
CAST(STDDEV_POP(Sal)OVER(PARTITION BY DepartmentID) AS
numeric(10,2)) ASSTD_DEVFROM EmployeesGROUP BY Dept, E_name, Sal) AS
derived_table WHERESal> (Average+STD_DEV )ORDER BY Dept, Sal,
E_name;
```

以上查询的结果：

```
 E_name    Dept                            Sal    Average   STD_DEV
--------  -----          --------  -------          --------
Lull              100        87900.00            58736.28
16829.59Sheffield          100            87900.00
58736.28            16829.59Scott               100
96300.00           58736.28           16829.59Sterling
200        64900.00             48390.94
13869.59Savarino               200        72300.00
48390.94            13869.59Kelly               200
87500.00            48390.94            13869.59Shea
300        138948.00            59500.00
30752.39Blaikie               400        54900.00
43640.67            11194.02Morris               400
61300.00            43640.67            11194.02Evans
400        68940.00             43640.67
11194.02Martinez              500        55500.80
33752.20             9084.49
```

对于 2000 年，平均订单数是 1,787。四种产品（700、601、600 和 400）的销售数量高于该数量。在 2001 年，平均订单数是 1,048，有三种产品超过该数量。

窗口化集合示例 2 - 此查询返回一个结果集，该结果集显示了其薪水比其部门的平均薪水高一个标准偏差的雇员。标准偏差是数据与平均值的差异的测量单位。

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
  Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
  CAST(AVG(Sal) OVER(PARTITION BY dept) AS
  numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
  OVER(PARTITION BY dept) AS numeric(10,2)) AS
  STD_DEV
FROM Employees
```

```
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
  Sal> (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

每个部门至少有一名雇员的薪水远远超过平均值，如下列结果所示：

```
E_name        Dept         Sal     Average       STD_DEV
--------      ----    --------    --------      --------
Lull           100    87900.00    58736.28      16829.59
Sheffield      100    87900.00    58736.28      16829.59
Scott          100    96300.00    58736.28      16829.59
Sterling       200    64900.00    48390.94      13869.59
Savarino       200    72300.00    48390.94      13869.59
Kelly          200    87500.00    48390.94      13869.59
Shea           300   138948.00    59500.00      30752.39
Blaikie        400    54900.00    43640.67      11194.02
Morris         400    61300.00    43640.67      11194.02
Evans          400    68940.00    43640.67      11194.02
Martinez       500    55500.80    33752.20       9084.49
```

雇员 Scott 的薪水为 $96,300.00，而部门 100 的平均薪水为 $58,736.28。该部门的标准偏差是 16,829.00，这表示低于 $75,565.88 (58736.28 + 16829.60 = 75565.88) 的薪水会在平均值的一个标准偏差范围内。

另请参见

- 分布函数 （第 53 页）
- OLAP 优点 （第 20 页）
- OLAP 计算 （第 20 页）
- 排名函数 （第 43 页）
- 统计集合函数 （第 49 页）
- 窗口化 （第 33 页）
- OLAP 函数的 BNF 语法 （第 67 页）

统计集合函数

ANSI SQL/OLAP 扩展提供了许多其它集合函数，这些函数允许对数值数据进行统计分析。此支持包括用于计算方差、标准偏差、相关和线性回归的函数。

标准偏差和方差

采用一个参数的 SQL/OLAP 常规集函数包括以下语法语句中以粗体显示的那些函数：

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
  <BASIC AGGREGATE FUNCTION TYPE>
  | STDDEV | STDDEV_POP | STDDEV_SAMP
  | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

- **STDDEV_POP** - 计算所提供的值表达式（为组或分区的每个行计算，如果指定了 **DISTINCT**，则为删除重复值后仍旧保留的每个行计算）的总体标准偏差，总体标准偏差是指总体方差的平方根。

- **STDDEV_SAMP** - 计算所提供的值表达式（为组或分区的每个行计算，如果指定了 **DISTINCT**，则为删除重复值后仍旧保留的每个行计算）的总体标准偏差，总体标准偏差是指样本方差的平方根。
- **VAR_POP** - 计算值表达式（为组或分区的每个行计算，如果指定了 **DISTINCT**，则为删除重复值后仍旧保留的每个行计算）的总体方差，总体方差是指值表达式与值表达式平均值的差值的平方和除以组或分区中的保留行数。
- **VAR_SAMP** - 计算值表达式（为组或分区的每个行计算，如果指定了 **DISTINCT**，则为删除重复值后仍旧保留的每个行计算）的样本方差，样本方差是指值表达式与值表达式平均值的差值的平方和除以组或分区中的保留行数减一。

这些函数（包括 **STDDEV** 和 **VARIANCE**）是真正的集合函数，原因在于，它们可以计算由查询的 **ORDER BY** 子句确定的行分区的值。就像使用其它基本集合函数（如 **MAX** 或 **MIN**）一样，它们的计算忽略输入中的空值。此外，无论所分析的表达式的域是什么，所有方差和标准偏差计算都使用 IEEE 双精度浮点值。如果对任何方差或标准偏差函数的输入为空集，则每个函数都将返回空值作为其结果。如果为单个行计算 **VAR_SAMP**，则它返回空值，而 **VAR_POP** 返回值 0。

相关
用于计算相关系数的 SQL/OLAP 函数是：

- **CORR** - 返回一组数字对的相关系数。

您可以使用 CORR 函数作为窗口化集合函数（在该函数中指定窗口名称或规范使用的窗口函数类型）或作为不带 OVER 子句的简单集合函数。

协方差
用于计算协方差的 SQL/OLAP 函数包括：

- **COVAR_POP** - 返回一组数字对的总体协方差。
- **COVAR_SAMP** - 返回一组数字对的样本协方差。

协方差函数删除了 expression1 或 expression2 的值为空值的所有对。

您可以使用协方差函数作为窗口化集合函数（在该函数中指定窗口名称或规范使用的窗口函数类型）或作为不带 OVER 子句的简单集合函数。

累积分布
用于计算一个值在行组中的相对位置的 SQL/OLAP 函数是 **CUME_DIST**。

窗口规范必须包含 ORDER_BY 子句。

不得在 CUME_DIST 函数中使用组合排序键。

回归分析
回归分析函数使用线性回归公式计算独立变量和相关变量之间的关系。SQL/OLAP 线性回归函数包括：

- **REGR_AVGX** - 计算回归线的独立变量的平均值。
- **REGR_AVGY** - 计算回归线的相关变量的平均值。
- **REGR_COUNT** - 返回一个整数，该整数表示用于拟合回归线的非空数字对的数量。
- **REGR_INTERCEPT** - 计算可以最好地拟合相关和独立变量的回归线的 y 截距。
- **REGR_R2** - 计算回归线的决定系数（拟合优度统计）。
- **REGR_SLOPE** - 计算与非空对拟合的线性回归线的斜率。
- **REGR_SXX** - 返回线性回归模型中使用的独立表达式的平方和。使用此函数可以计算回归模型的统计有效性。
- **REGR_SXY** - 返回相关和独立变量的乘积之和。使用此函数可以计算回归模型的统计有效性。
- **REGR_SYY** - 返回可以计算回归模型的统计有效性的值。

您可以使用回归分析函数作为窗口化集合函数（在该函数中指定窗口名称或规范使用的窗口函数类型）或作为不带 OVER 子句的简单集合函数。

加权 OLAP 集合
加权 OLAP 集合函数计算加权移动平均值:

- **EXP_WEIGHTED_AVG** - 计算指数加权移动平均值。加权确定构成平均值的每个数量的相对重要性。**EXP_WEIGHTED_AVG** 中的权重呈指数级减小。指数加权会为最新的值应用较多权重，而减小较旧值的权重，同时仍旧为较旧值应用一些权重。
- **WEIGHTED_AVG** - 计算线性加权移动平均值，其中权重随着时间的推移按算术级数减小。对于最新的数据点，权重从最高值减小，而对于最旧的数据点，权重减小为零。

窗口规范必须包含 **ORDER_BY** 子句。

非标准数据库行业扩展
在数据库行业使用的非 ANSI SQL/OLAP 集合函数扩展包括 **FIRST_VALUE**、**MEDIAN** 和 **LAST_VALUE**。

- **FIRST_VALUE** - 返回一组值中的第一个值。
- **MEDIAN** - 返回表达式中的中位数。
- **LAST_VALUE** - 返回一组值中的最后一个值。

FIRST_VALUE 和 **LAST_VALUE** 函数需要窗口规范。您可以使用 **MEDIAN** 函数作为窗口化集合函数（在该函数中指定窗口名称或规范使用的窗口函数类型）或作为不带 OVER 子句的简单集合函数。

另请参见
- 分布函数 （第 53 页）
- OLAP 优点 （第 20 页）
- OLAP 计算 （第 20 页）
- 排名函数 （第 43 页）

- 窗口化 （第 33 页）
- 窗口化集合函数 （第 47 页）
- OLAP 函数的 BNF 语法 （第 67 页）

行间函数

通过行间函数 **LAG** 和 **LEAD** 可以访问数据系列中的先前值或后续值，或表中的多个行。

行间函数还同时进行分区，且无需自连接。LAG 提供对在表或分区中 **CURRENT ROW** 前面且与之相距给定物理偏移量的行的访问。LEAD 提供对在表或分区中 **CURRENT ROW** 后面且与之相距给定物理偏移量的行的访问。

LAG 和 **LEAD** 使用的语法相同。这两个函数都需要 **OVER** (**ORDER_BY**) 窗口规范。例如：

```
LAG (value_expr) [, offset [, default]]) OVER ([PARTITION BY
  window partition] ORDER BY
  window ordering)
```

和：

```
LEAD (value_expr) [, offset [, default]]) OVER ([PARTITION
BY
  window partition] ORDER BY
  window ordering)
```

OVER (**ORDER_BY**) 子句中的 **PARTITION BY** 子句是可选的。**OVER** (**ORDER_BY**) 子句不能包含窗口构架 **ROWS**/**RANGE** 规范。

value_expr 是定义要从表返回的偏移数据的表列或表达式。可以在 *value_expr* 中定义其它函数（分析函数除外）。

对于这两个函数，都需要通过输入物理偏移量来指定目标行。*offset* 值是指当前行上面或下面的行数。请输入一个非负数值数据类型（输入负值会生成错误）。如果输入 0，Sybase IQ 会返回当前行。

可选的 *default* 值定义 *offset* 值超过表的范围时要返回的值。*default* 的缺省值为 **NULL**。数据类型 *default* 必须可隐式转换为数据类型 *value_expr*，否则 Sybase IQ 将生成转换错误。

LAG 示例 1 - 行间函数用于对数据流进行计算的金融服务应用程序（如股票交易）。此示例使用 **LAG** 函数来计算特定股票的交易价格的百分比变化。请考虑下面的来自 `stock_trades` 虚构表的交易数据：

```
traded at            symbol   price
-------------------  ------   ------
2009-07-13 06:07:12  SQL      15.84
2009-07-13 06:07:13  TST       5.75
2009-07-13 06:07:14  TST       5.80
2009-07-13 06:07:15  SQL      15.86
```

```
2009-07-13 06:07:16  TST       5.90
2009-07-13 06:07:17  SQL      15.86
```

注意： stock_trades 虚构表在 iqdemo 数据库中不可用。

查询按照股票符号对交易进行分区，按照交易时间对它们进行排序，并使用 **LAG** 函数来计算当前交易和以前交易之间的交易价格增加或降低百分比：

```
select  stock_symbol as 'Stock',
    traded_at    as 'Date/Time of Trade',
    trade_price  as 'Price/Share',
    cast ( ( ( (trade_price
        - (lag(trade_price, 1)
        over (partition by stock_symbol
            order by traded_at)))
        / trade_price)
    * 100.0) as numeric(5, 2) )
        as '% Price Change vs Previous Price'
from stock_trades
order by 1, 2
```

查询返回以下结果：

```
Stock   Date/Time of Trade   Price/  % Price Change_vs
symbol                       Share   Previous Price
------  -------------------  -----   -----------------
SQL     2009-07-13 06:07:12  15.84   NULL
SQL     2009-07-13 06:07:15  15.86   0.13
SQL     2009-07-13 06:07:17  15.86   0.00
TST     2009-07-13 06:07:13   5.75   NULL
TST     2009-07-13 06:07:14   5.80   0.87
TST     2009-07-13 06:07:16   5.90   1.72
```

第一和第四输出行中的 NULL 结果表明 **LAG** 函数对于两个分区中的第一行都超出范围。由于没有先前行以供比较，Sybase IQ 返回 *default* 变量指定的 NULL。

分布函数

SQL/OLAP 定义了多个用于处理有序集合的函数。

这两个逆分布函数是 **PERCENTILE_CONT** 和 **PERCENTILE_DISC**。这些分析函数将一个百分点值作为函数参数使用，并对 **WITHIN GROUP** 子句中指定的一组数据执行操作或对整个数据集执行操作。

这些函数为每个组返回一个值。对于 **PERCENTILE_DISC**（离散），结果的数据类型与在 **WITHIN GROUP** 子句中指定的它的 **ORDER BY** 项的数据类型相同。对于 **PERCENTILE_CONT**（连续），结果的数据类型是数字，但前提是 **WITHIN GROUP** 子句中的 **ORDER BY** 项是数字或双精度型，或者 **ORDER BY** 项是整数或浮点型。

逆分布分析函数需要 **WITHIN GROUP** (**ORDER BY**) 子句。例如：

```
PERCENTILE_CONT ( expression1 )
```

```
WITHIN GROUP ( ORDER BY
            expression2 [ ASC | DESC ] )
```

expression1 的值必须是数值数据类型的常量，范围从 0 到 1（包含这两个数）。如果参数为 NULL，将返回“wrong argument for percentile”（百分点的参数错误）错误。如果参数值小于 0 或大于 1，将返回“data value out of range”（数据值超出范围）错误。

ORDER BY 子句（必须存在）指定对其执行百分点函数的表达式以及每组中行的排序顺序。此 **ORDER BY** 子句只在 **WITHIN GROUP** 子句中使用，而不是用于 **SELECT** 语句的 **ORDER BY**。

WITHIN GROUP 子句将查询结果分布到排序数据集中，函数通过此数据集计算结果。

值 *expression2* 是一种排序规范，必须是涉及列引用的单个表达式。不允许多个表达式，并且在此排序表达式中不允许使用 rank 分析函数、集合函数或子查询。

ASC 或 DESC 参数指定排序顺序，如升序或降序。升序是缺省值。

子查询、**HAVING** 子句、视图或联合中允许使用逆分布分析函数。可以在使用简单非分析集合函数的任意位置使用逆分布函数。逆分布函数忽略数据集中的空值。

PERCENTILE_CONT 示例 - 此示例根据以下数据集使用 **PERCENTILE_CONT** 函数确定某个区域中要进入前 10 个百分点而应该达到的汽车销售量：

```
sales       region        dealer_name
-----       ---------     -----------
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

在以下示例查询中，**SELECT** 语句包含 **PERCENTILE_CONT** 函数：

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY ProductID DESC )
FROM ViewSalesOrdersSales GROUP BY region;
```

SELECT 语句的结果列出了某区域中要进入前 10 个百分点应该达到的汽车销售量：

```
region           percentile_cont
---------        ---------------
Canada                     601.0
```

```
Central                          700.0
Eastern                          700.0

South                           700.0
Western                          700.0
```

PERCENTILE_DISC 示例 - 此示例根据以下数据集使用 **PERCENTILE_DISC** 函数确定某个区域中要进入前 10 个百分点而应该达到的汽车销售量:

```
sales       region        dealer_name
-----       ---------     -----------
900         Northeast     Boston
800         Northeast     Worcester
800         Northeast     Providence
700         Northeast     Lowell
540         Northeast     Natick
500         Northeast     New Haven
450         Northeast     Hartford
800         Northwest     SF
600         Northwest     Seattle
500         Northwest     Portland
400         Northwest     Dublin
500         South         Houston
400         South         Austin
300         South         Dallas
200         South         Dover
```

在以下查询中，**SELECT** 语句包含 **PERCENTILE_DISC** 函数:

```
SELECT region, PERCENTILE_DISC(0.1) WITHIN GROUP    (ORDER BY sales
DESC ) FROM carSales GROUP BY region;
```

SELECT 语句的结果列出了每一个区域中要进入前 10 个百分点而应该达到的汽车销售量:

```
region           percentile_cont ---------        ---------------
Northeast        900 Northwest        800 South             500
```

有关分布函数的详细信息，请参见《参考：构件块、表和过程》>“SQL 函数”>“PERCENTILE_CONT 函数 [分析]”和“PERCENTILE_DISC 函数 [分析]”。

另请参见

- OLAP 优点 （第 20 页）
- OLAP 计算 （第 20 页）
- 排名函数 （第 43 页）
- 统计集合函数 （第 49 页）
- 窗口化 （第 33 页）
- 窗口化集合函数 （第 47 页）
- OLAP 函数的 BNF 语法 （第 67 页）

数值函数

Sybase IQ 支持的 OLAP 数值函数包括 **CEILING**（别名为 **CEIL**）、**EXP**（别名为 **EXPONENTIAL**）、**FLOOR**、**LN**（别名为 **LOG**）、**SQRT** 和 **WIDTH_BUCKET**。

```
<numeric value function> :: =
 <natural logarithm>
| <exponential function>
| <power function>
| <square root>
| <floor function>
| <ceiling function>
| <width bucket function>
```

表 4. 数值函数和语法

数值函数	语法
自然对数	**LN** (*numeric-expression*)
指数函数	**EXP** (*numeric-expression*)
Power 函数	**POWER** (*numeric-expression1*, *numeric-expression2*)
平方根	**SQRT** (*numeric-expression*)
Floor 函数	**FLOOR** (*numeric-expression*)
Ceiling 函数	**CEILING** (*numeric-expression*)
Width bucket 函数	**WIDTH_BUCKET** (*expression*, *min_value*, *max_value*, *num_buckets*)

数值函数的语义是：

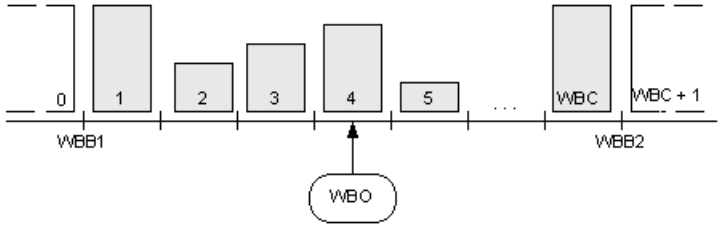
- **LN** - 返回参数值的自然对数。如果参数值为零或为负数，则会引发错误。LN 是 **LOG** 的同义词。
- **EXP** - 返回通过将 *e*（自然对数的底数）的值增加至参数值指定的幂而计算出的值。
- **POWER** - 返回通过将第一个参数的值增加至第二个参数的值指定的幂而计算出的值。如果第一个参数为 0，第二个参数也为 0，则返回 1。如果第一个参数为 0，而第二个参数为正数，则返回 0。如果第一个参数为 0，而第二个参数为负数，则引发异常。如果第一个参数为负数，第二个参数不是整数，则引发异常。
- **SQRT** - 返回参数值的平方根，该平方根由“**POWER** (*expression*, *0.5*)”的语法转换定义。
- **FLOOR** - 返回最接近于正无限大且不大于参数值的整数值。
- **CEILING** - 返回最接近于负无限小且不小于参数值的整数值。CEIL 是 CEILING 的同义词。

WIDTH_BUCKET 函数

WIDTH_BUCKET 函数比其它数值函数略微复杂一些。它接受四个参数：“实时值”、两个范围边界以及大小相同（或尽可能接近）的分区（边界指示的范围要拆分为这些

分区）的数目。WIDTH_BUCKET 返回一个数字，该数字指示应将实时值放置在的分区（基于它的值，它的值是范围上边界和下边界之间差值的百分比）。第一个分区的编号为 1。

为了避免在实时值超出边界范围时出错，小于范围下边界的值应该放置在第一个附加表元（即表元 0）中，大于范围上边界的值应该放在最后一个附加表元（即表元 N +1）中。



例如，**WIDTH_BUCKET** (*14, 5, 30, 5*) 返回 2，原因是：

- (30-5)/5 = 5，因此范围拆分为 5 个分区，每个分区的宽度为 5 个单位。
- 第一个表元表示从 0.00% 到 19.999 …% 的值，第二个表元表示从 20.00% 到 39.999 …% 的值，第五个表元表示从 80.00% 到 100.00% 的值。
- 所选表元是通过计算 (5*(14-5)/(30-5)) + 1（表元数乘以指定值距最小值的偏移与可能值范围的比率，再加上一，即 (5*9/25) + 1，结果是 2.8）来确定的。此值是编号为 2 的表元的值范围（2.0 到 2.999……），因此选择编号为 2 的表元。

WIDTH_BUCKET 示例

以下示例基于 `credit_limit` 列为样本表中马萨诸塞州的客户创建十表元直方图，并为每名客户返回表元号（“Credit Group”）。其信用额度超过最大值的客户将被分配到溢出表元 11：

注意： 此示例仅用于进行说明，不会使用 iqdemo 数据库生成此示例。

```
SELECT customer_id, cust_last_name, credit_limit,
  WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit
  Group"
  FROM customers WHERE territory = 'MA'
  ORDER BY "Credit Group";
```

```
CUSTOMER_ID  CUST_LAST_NAME  CREDIT_LIMIT  Credit Group
-----------  --------------  ------------  ------------
825          Dreyfuss                 500             1
826          Barkin                   500             1
853          Palin                    400             1
827          Siegel                   500             1
843          Oates                    700             2
844          Julius                   700             2
835          Eastwood                1200             3
840          Elliott                 1400             3
842          Stern                   1400             3
```

```
841          Boyer                    1400             3
837          Stanton                  1200             3
836          Berenger                 1200             3
848          Olmos                    1800             4
847           Streep                      5000                  11
```

如果将这些限定反转过来，表元将成为半开半闭区间。例如：**WIDTH_BUCKET** (*credit_limit, 5000, 0, 5*)。在此示例中，表元号 1 的上下限为 (4000, 5000]，表元号 2 的上下限为 (3000, 4000]，表元号 5 的上下限为 (0, 1000]。上溢表元的编号为 0 (5000, +infinity)，下溢表元的编号为 6 (-infinity, 0]。

另请参见
《参考：构件块、表和过程》>“SQL 函数”>“BIT_LENGTH 函数 [字符串]”、“EXP 函数 [数值]”、“FLOOR 函数 [数值]”、“POWER 函数 [数值]”、“SQRT 函数 [数值]”和“WIDTH_BUCKET 函数 [数值]”。

OLAP 规则和限制

下文概述了控制 OLAP 功能的规则和限制。

可以使用 OLAP 函数
Sybase IQ 为 SQL OLAP 函数提供了一些规则、约束和限制。

- 在 **SELECT** 列表中
- 在表达式中
- 作为标量函数的参数
- 在最终 **ORDER BY** 子句中（通过在查询中的其它位置使用 OLAP 函数的别名或位置引用）

不能使用 OLAP 函数
在以下情况下，*不能*使用 OLAP 函数：

- 在子查询中。
- 在 **WHERE** 子句的搜索条件中。
- 作为 **SET**（集合）函数的参数。例如，以下表达式无效：
  ```
  SUM(RANK() OVER(ORDER BY dollars))
  ```
- 窗口集合不能是另一个参数的参数，但如果在视图或派生表中生成内部参数则例外。对于排名函数，同样如此。
- 窗口集合函数和 **RANK** 函数不得在 **HAVING** 子句中使用。
- 窗口集合函数不得指定 **DISTINCT**。
- 窗口函数不能嵌套在其它窗口函数内部。
- **OVER** 子句不支持逆分布函数。
- 不得在窗口定义子句中使用外部引用。
- 允许在 OLAP 函数中使用相关引用，但不允许使用相关列别名。

OLAP 函数引用的列必须是同一查询块（OLAP 函数和 **GROUP BY** 子句显示在该查询块中）中的分组列或集合函数。OLAP 处理发生在分组和集合操作之后、应用最终 **ORDER BY** 子句之前；因此，必须可以从这些中间结果派生 OLAP 表达式。如果查询块中没有 **GROUP BY** 子句，则 OLAP 函数可以引用选择列表中的其它列。

Sybase IQ 限制

Sybase IQ 对 SQL OLAP 函数的限制如下：

- 不支持窗口构架定义中用户定义的函数。
- 在窗口构架定义中使用的常量必须是不带符号的数值，且不得超过 `BIG INT` 2^{63-1} 的最大值。
- 窗口集合函数和 **RANK** 函数不能在 **DELETE** 和 **UPDATE** 语句中使用。
- 窗口集合函数和 **RANK** 函数不得在子查询中使用。
- CUME_DIST 当前不受支持。
- 分组集当前不受支持。
- 相关函数和线性回归函数当前不受支持。

其它 OLAP 示例

此节提供了其它 OLAP 函数使用示例。

当处理中间结果行时，窗口的起点和终点可能会改变。例如，计算累计总和将涉及一个窗口，该窗口的起点固定在每个分区的第一行，终点沿分区行滑动以包括当前行。

像另一示例一样，窗口的起点和终点都是可变的，但它能为整个分区定义固定数量的行。使用这种构造，用户可以编写计算移动平均值的查询；例如，返回三天的股票价格的移动平均值的 SQL 查询。

示例：查询中的窗口函数

此查询列出了 2005 年 7 月和 8 月装运的所有产品以及到装运日期为止的累计装运数量：

```
SELECT p.id, p.description, s.quantity, s.shipdate,

SUM(s.quantity) OVER (PARTITION BY productid ORDER BYs.shipdate rows
between unbounded preceding andcurrent row)FROM SalesOrderItems s
JOIN Products p on(s.ProductID =p.id) WHERE s.ShipDate BETWEEN
'2001-05-01' and'2001-08-31' AND s.quantity > 40ORDER BY p.id;
```

以上查询的结果：

```
ID   description   quantity   ship_date   sum quantity
---  -----------   --------   ---------   ------------
302  Crew Neck           60  2001-07-02             60
400  Cotton Cap          60  2001-05-26             60
400  Cotton Cap          48  2001-07-05            108
401  Wool cap            48  2001-06-02             48
401  Wool cap            60  2001-06-30            108
```

```
401  Wool cap            48  2001-07-09           156
500  Cloth Visor         48  2001-06-21            48
501  Plastic Visor       60  2001-05-03            60
501  Plastic Visor       48  2001-05-18           108
501  Plastic Visor       48  2001-05-25           156
501  Plastic Visor       60  2001-07-07           216
601  Zipped Sweatshirt   60  2001-07-19            60
700  Cotton Shorts       72  2001-05-18            72
700  Cotton Shorts       48  2001-05-31           120
```

在此示例中，要在连接两个表和应用查询的 **WHERE** 子句之后，才执行 **SUM** 窗口函数的计算。查询使用行内窗口规格，该规范指定按如下方式处理的连接的输入行：

1. 根据 `prod_id` 属性的值将输入行分区（分组）。
2. 在每个分区中，按 `ship_date` 属性对行进行排序。
3. 对于分区中的每一行，计算数量属性的 **SUM()** 函数，并使用包括每个分区的第一（排序）行的滑动窗口，直到包含当前行。

另一种构造查询的方式是指定与使用它的函数不同的窗口。当根据同一窗口指定多个窗口函数时，这非常有用。在使用窗口函数的查询中，使用窗口子句（声明由累计标识的窗口）的构造如下所示：

```
SELECT p.id, p.description, s.quantity, s.shipdate, SUM(s.quantity)
OVER(cumulative ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW )
cumulative FROM SalesOrderItems s JOIN Products p On (s.ProductID
=p.id)WHERE s.shipdate BETWEEN ‘2001-07-01’ and ‘2001-08-31’Window
cumulative as (PARTITION BY s.productid ORDER BY s.shipdate)ORDER BY
p.id;
```

窗口子句显示在查询规范中的 **ORDER BY** 子句之前。当使用窗口子句时，存在以下限制：

- 行内窗口规格不得包含 **PARTITION BY** 子句。
- 在窗口子句中指定的窗口不得包含窗口构架子句。

  ```
  <WINDOW FRAME CLAUSE> ::=
    <WINDOW FRAME UNIT>
    <WINDOW FRAME EXTENT>
  ```
- 行内窗口规格或在窗口子句中指定的窗口规范都可以（但不能同时）包含窗口排序子句。

  ```
  <WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
  ```

示例：含多个函数的窗口

此查询定义一个（命名）窗口并通过它计算多个函数结果：

```
SELECT p.ID, p.Description, s.quantity, s.ShipDate,SUM(s.Quantity)
OVER ws1, MIN(s.quantity) OVER ws1FROM SalesOrderItems s JOIN
Products p ON (s.ProductID =p.ID) WHERE s.ShipDate BETWEEN
'2000-01-09' AND'2000-01-17' AND s.Quantity > 40 window ws1
AS(PARTITION BY productid ORDER BY shipdate rowsbetween unbounded
preceding and current row)ORDER BY p.id;
```

以上查询的结果：

```
ID   Description   quantity   shipDate     SUM   MIN
---  -----------   --------   -----------  ---   ---
400  Cotton Cap          48    2000-01-09   48    48
401  Wool cap            48    2000-01-09   48    48
500  Cloth Visor         60    2000-01-14   60    60
500  Cloth Visor         60    2000-01-15  120    60
501  Plastic Visor           60    2000-01-14   60    60
```

示例：计算累计总和

以下查询按部门和 **ORDER BY** `start_date` 计算薪水的累计总和。

```
SELECT dept_id, start_date, name, salary,
  SUM(salary) OVER (PARTITION BY dept_id ORDER BY
  start_date ROWS BETWEEN UNBOUNDED PRECEDING AND
  CURRENT ROW)
FROM emp1
ORDER BY dept_id, start_date;
```

以上查询的结果：

```
DepartmentID  start_date    name     salary       sum

                                                  (salary)

-------                   ----------    ----   ------   ---------
100                   1996-01-01       Anna
18000             18000
100                    1997-01-01       Mike
28000    46000
100                       1998-01-01    Scott     29000    75000
100                       1998-02-01    Antonia   22000    97000
100                       1998-03-12    Adam      25000   122000
100                       1998-12-01    Amy       18000   140000
200                       1998-01-01    Jeff      18000    18000
200                       1998-01-20    Tim       29000    47000
200                       1998-02-01    Jim       22000    69000
200                       1999-01-10    Tom       28000    97000
300                       1998-03-12    Sandy     55000    55000
300                       1998-12-01    Lisa      38000    93000
300                       1999-01-10    Peter     48000    141000
```

示例：计算移动平均值

以下查询生成连续三个月的销售额的移动平均值。窗口构架的大小是三行：两个前面的行加一个当前行。窗口从分区的开始滑至结尾。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

以上查询的结果：

```
prod_id   month_num      sales       avg(sales)
-------   ---------      ------      ----------
```

```
10                1         100          100.00
10                2         120          110.00
10                3         100          106.66
10                4         130          116.66
10                5         120          116.66
10                6         110          120.00
20                1          20           20.00
20                2          30           25.00
20                3          25           25.00
20                4          30           28.33
20                5          31           28.66
20                6          20           27.00
30                1          10           10.00
30                2          11           10.50
30                3          12           11.00
30                4           1            8.00
```

示例：ORDER BY 结果

在此示例中，查询顶部的 **ORDER BY** 子句应用到窗口函数的最终结果。窗口子句中的 **ORDER BY** 应用到窗口函数的输入数据。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id desc, month_num;
```

以上查询的结果：

```
prod_id  month_num       sales   avg(sales)
-------  ---------       -----   ----------
30               1          10       10.00
30               2          11       10.50
30               3          12       11.00
30               4           1        8.00
20               1          20       20.00
20               2          30       25.00
20               3          25       25.00
20               4          30       28.33
20               5          31       28.66
20               6          20       27.00
10               1         100      100.00
10               2         120      110.00
10               3         100      106.66
10               4         130      116.66
10               5         120      116.66
10                   6         110       120.00
```

示例：查询中的多个集合函数

此示例针对查询中的不同窗口计算集合值。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
  (WS1 ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS
  CAvg, SUM(sales) OVER(WS1 ROWS BETWEEN UNBOUNDED
```

```
  PRECEDING AND CURRENT ROW) AS CSum
FROM sale WHERE rep_id = 1 WINDOW WS1 AS (PARTITION BY
  prod_id
ORDER BY month_num)
ORDER BY prod_id, month_num;
```

以上查询的结果：

```
prod_id  month_num    sales     CAvg     CSum
-------  ---------    -----     ----     ----
10               1      100   110.00      100
10               2      120   106.66      220
10               3      100   116.66      320
10               4      130   116.66      450
10               5      120   120.00      570
10               6      110   115.00      680
20               1       20    25.00       20
20               2       30    25.00       50
20               3       25    28.33       75
20               4       30    28.66      105
20               5       31    27.00      136
20               6       20    25.50      156
30               1       10    10.50       10
30               2       11    11.00       21
30               3       12     8.00       33
30            4        1      6.50       34
```

示例：对 ROWS 和 RANGE 进行比较的窗口构架

此查询对 ROWS 和 RANGE 进行比较。数据在每个 **ORDER BY** 子句中包含重复的 ROWS。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (ws1 RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) AS
    Range_sum, SUM(sales) OVER
  (ws1 ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
    Row_sum
FROM sale window ws1 AS (PARTITION BY prod_id ORDER BY
  month_num)
ORDER BY prod_id, month_num;
```

以上查询的结果：

```
prod_id  month_num   sales   Range_sum   Row_sum
-------  ---------   -----   ----------  -------
10               1     100         250      100
10               1     150         250      250
10               2     120         370      370
10               3     100         470      370
10               4     130         350      350
10               5     120         381      350
10               5      31         381      281
10               6     110         391      261
20               1      20          20       20
20               2      30          50       50
20               3      25          75       75
```

```
20                4       30           85         85
20                5       31           86         86
20                6       20           81         81
30                1       10           10         10
30                2       11           21         21
30                3       12           33         33
30                4        1           25         24
30                4        1           25         14
```

示例：不包括当前行的窗口构架

在此示例中，您可以定义窗口构架以排除当前行。查询计算四个行（不包括当前行）的总和。

```
SELECT prod_id, month_num, sales, sum(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN 6 PRECEDING AND 2 PRECEDING)
FROM sale
ORDER BY prod_id, month_num;
```

以上查询的结果：

```
prod_id   month_num       sales    sum(sales)
-------   ---------       -----    ----------
10                1         100        (NULL)
10                1         150        (NULL)
10                2         120        (NULL)
10                3         100          250
10                4         130          370
10                5         120          470
10                5          31          470
10                6         110          600
20                1          20        (NULL)
20                2          30        (NULL)
20                3          25           20
20                4          30           50
20                5          31           75
20                6          20          105
30                1          10        (NULL)
30                2          11        (NULL)
30                3          12           10
30                4           1           21
30                    4           1           21
```

示例：RANGE 的窗口构架

此查询说明 RANGE 窗口构架。汇总中使用的行数是可变的。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN 1 FOLLOWING AND 3 FOLLOWING)
FROM sale
ORDER BY prod_id, month_num;
```

以上查询的结果：

```
prod_id   month_num       sales    sum(sales)
-------   ---------       -----    ----------
10                1         100           350
10                1         150           350
10                2         120           381
10                3         100           391
10                4         130           261
10                5         120           110
10                5          31           110
10                6         110        (NULL)
20                1          20            85
20                2          30            86
20                3          25            81
20                4          30            51
20                5          31            20
20                6          20        (NULL)
30                1          10            25
30                2          11            14
30                3          12             2
30                4           1        (NULL)
30                4           1        (NULL)
```

示例：Unbounded preceding and unbounded following

在此示例中，窗口构架可以包括分区中的所有行。查询计算整个分区（一个月中没有重复行）的最大销售额。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num ROWS
  BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

以上查询的结果：

```
prod_id   month_num       sales    SUM(sales)
-------   ---------       -----    ----------
10                1         100           680
10                2         120           680
10                3         100           680
10                4         130           680
10                5         120           680
10                6         110           680
20                1          20           156
20                2          30           156
20                3          25           156
20                4          30           156
20                5          31           156
20                6          20           156
30                1          10            34
30                2          11            34
30                3          12            34
30                4           1            34
```

本示例中的查询相当于：

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id )
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

示例：RANGE 的缺省窗口构架

此查询说明 RANGE 的缺省窗口构架：

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num)
FROM sale
ORDER BY prod_id, month_num;
```

以上查询的结果：

```
prod_id   month_num       sales    SUM(sales)
-------   ---------       -----    ----------
10                1         100           250
10                1         150           250
10                2         120           370
10                3         100           470
10                4         130           600
10                5         120           751
10                5          31           751
10                6         110           861
20                1          20            20
20                2          30            50
20                3          25            75
20                4          30           105
20                5          31           136
20                6          20           156
30                1          10            10
30                2          11            21
30                3          12            33
30                4           1            35
30                4           1            35
```

本示例中的查询相当于：

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
  (PARTITION BY prod_id ORDER BY month_num RANGE
  BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM sale
ORDER BY prod_id, month_num;
```

OLAP 函数的 BNF 语法

Backus-Naur Form 语法概述了各种 ANSI SQL 分析函数（其中有许多在 Sybase IQ 中实现）的特定语法支持。

语法规则 1

```
<SELECT LIST EXPRESSION> ::=
  <EXPRESSION>
  | <GROUP BY EXPRESSION>
  | <AGGREGATE FUNCTION>
  | <GROUPING FUNCTION>
  | <TABLE COLUMN>
  | <WINDOWED TABLE FUNCTION>
```

语法规则 2

```
<QUERY SPECIFICATION> ::=
  <FROM CLAUSE>
  [ <WHERE CLAUSE> ]
  [ <GROUP BY CLAUSE> ]
  [ <HAVING CLAUSE> ]
  [ <WINDOW CLAUSE> ]
[ <ORDER BY CLAUSE> ]
```

语法规则 3

```
<ORDER BY CLAUSE> ::= <ORDER SPECIFICATION>
```

语法规则 4

```
<GROUPING FUNCTION> ::=
  GROUPING <LEFT PAREN> <GROUP BY EXPRESSION>
  <RIGHT PAREN>
```

语法规则 5

```
<WINDOWED TABLE FUNCTION> ::=
  <WINDOWED TABLE FUNCTION TYPE> OVER <WINDOW NAME OR
  SPECIFICATION>
```

语法规则 6

```
<WINDOWED TABLE FUNCTION TYPE> ::=
  <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
  | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
  | <WINDOW AGGREGATE FUNCTION>
```

语法规则 7

```
<RANK FUNCTION TYPE> ::=
  RANK | DENSE RANK | PERCENT RANK | CUME_DIST
```

语法规则 8

```
<WINDOW AGGREGATE FUNCTION> ::=
  <SIMPLE WINDOW AGGREGATE FUNCTION>
  | <STATISTICAL AGGREGATE FUNCTION>
```

语法规则 9

```
<AGGREGATE FUNCTION> ::=
  <DISTINCT AGGREGATE FUNCTION>
  | <SIMPLE AGGREGATE FUNCTION>
  | <STATISTICAL AGGREGATE FUNCTION>
```

语法规则 10

```
<DISTINCT AGGREGATE FUNCTION> ::=
  <BASIC AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <DISTINCT> <EXPRESSION> <RIGHT PAREN>
  | LIST <LEFT PAREN> DISTINCT <EXPRESSION>
  [ <COMMA> <DELIMITER> ]
  [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

语法规则 11

```
<BASIC AGGREGATE FUNCTION TYPE> ::=
  SUM | MAX | MIN | AVG | COUNT
```

语法规则 12

```
<SIMPLE AGGREGATE FUNCTION> ::=
  <SIMPLE AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <EXPRESSION> <RIGHT PAREN>
  | LIST <LEFT PAREN> <EXPRESSION> [ <COMMA>
  <DELIMITER> ]
  [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

语法规则 13

```
<SIMPLE AGGREGATE FUNCTION TYPE> ::= <SIMPLE WINDOW AGGREGATE
FUNCTION TYPE>
```

语法规则 14

```
<SIMPLE WINDOW AGGREGATE FUNCTION> ::=
  <SIMPLE WINDOW AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <EXPRESSION> <RIGHT PAREN>
| GROUPING FUNCTION
```

语法规则 15

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
  <BASIC AGGREGATE FUNCTION TYPE>
  | STDDEV | STDDEV_POP | STDDEV_SAMP
  | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

语法规则 16

```
<STATISTICAL AGGREGATE FUNCTION> ::=
  <STATISTICAL AGGREGATE FUNCTION TYPE> <LEFT PAREN>
  <DEPENDENT EXPRESSION> <COMMA> <INDEPENDENT
  EXPRESSION> <RIGHT PAREN>
```

语法规则 17

```
<STATISTICAL AGGREGATE FUNCTION TYPE> ::=
  CORR | COVAR_POP | COVAR_SAMP | REGR_R2 |
  REGR_INTERCEPT | REGR_COUNT | REGR_SLOPE |
  REGR_SXX | REGR_SXY | REGR_SYY | REGR_AVGY |
  REGR_AVGX
```

语法规则 18

```
<WINDOW NAME OR SPECIFICATION> ::=
  <WINDOW NAME> | <IN-LINE WINDOW SPECIFICATION>
```

语法规则 19

```
<WINDOW NAME> ::= <IDENTIFIER>
```

语法规则 20

```
<IN-LINE WINDOW SPECIFICATION> ::= <WINDOW SPECIFICATION>
```

语法规则 21

```
<WINDOW CLAUSE> ::= <WINDOW WINDOW DEFINITION LIST>
```

语法规则 22

```
<WINDOW DEFINITION LIST> ::=
  <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
  } . . . ]
```

语法规则 23

```
<WINDOW DEFINITION> ::=
  <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

语法规则 24

```
<NEW WINDOW NAME> ::= <WINDOW NAME>
```

语法规则 25

```
<WINDOW SPECIFICATION> ::=
  <LEFT PAREN> <WINDOW SPECIFICATION> <DETAILS> <RIGHT
  PAREN>
```

语法规则 26

```
<WINDOW SPECIFICATION DETAILS> ::=
  [ <EXISTING WINDOW NAME> ]
  [ <WINDOW PARTITION CLAUSE> ]
  [ <WINDOW ORDER CLAUSE> ]
  [ <WINDOW FRAME CLAUSE> ]
```

语法规则 27

```
<EXISTING WINDOW NAME> ::= <WINDOW NAME>
```

语法规则 28

```
<WINDOW PARTITION CLAUSE> ::=
  PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

语法规则 29

```
<WINDOW PARTITION EXPRESSION LIST> ::=
  <WINDOW PARTITION EXPRESSION>
  [ { <COMMA> <WINDOW PARTITION EXPRESSION> } . . . ]
```

语法规则 30

```
<WINDOW PARTITION EXPRESSION> ::= <EXPRESSION>
```

语法规则 31

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

语法规则 32

```
<WINDOW FRAME CLAUSE> ::=
  <WINDOW FRAME UNIT>
  <WINDOW FRAME EXTENT>
```

语法规则 33

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

语法规则 34

```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW FRAME
BETWEEN>
```

语法规则 35

```
<WINDOW FRAME START> ::=
  UNBOUNDED PRECEDING
  | <WINDOW FRAME PRECEDING>
  | CURRENT ROW
```

语法规则 36

```
<WINDOW FRAME PRECEDING> ::= <UNSIGNED VALUE SPECIFICATION>
PRECEDING
```

语法规则 37

```
<WINDOW FRAME BETWEEN> ::=
  BETWEEN <WINDOW FRAME BOUND 1> AND <WINDOW FRAME
  BOUND 2>
```

语法规则 38

```
<WINDOW FRAME BOUND 1> ::= <WINDOW FRAME BOUND>
```

语法规则 39

```
<WINDOW FRAME BOUND 2> ::= <WINDOW FRAME BOUND>
```

语法规则 40

```
<WINDOW FRAME BOUND> ::=
  <WINDOW FRAME START>
  | UNBOUNDED FOLLOWING
  | <WINDOW FRAME FOLLOWING>
```

语法规则 41

```
<WINDOW FRAME FOLLOWING> ::= <UNSIGNED VALUE SPECIFICATION>
FOLLOWING
```

语法规则 42

```
<GROUP BY EXPRESSION> ::= <EXPRESSION>
```

语法规则 43

```
<SIMPLE GROUP BY TERM> ::=
  <GROUP BY EXPRESSION>
  | <LEFT PAREN> <GROUP BY EXPRESSION> <RIGHT PAREN>
  | <LEFT PAREN> <RIGHT PAREN>
```

语法规则 44

```
<SIMPLE GROUP BY TERM LIST> ::=
  <SIMPLE GROUP BY TERM> [ { <COMMA> <SIMPLE GROUP BY
  TERM> } . . . ]
```

语法规则 45

```
<COMPOSITE GROUP BY TERM> ::=
  <LEFT PAREN> <SIMPLE GROUP BY TERM>
  [ { <COMMA> <SIMPLE GROUP BY TERM> } . . . ]
  <RIGHT PAREN>
```

语法规则 46

```
<ROLLUP TERM> ::= ROLLUP <COMPOSITE GROUP BY TERM>
```

语法规则 47

```
<CUBE TERM> ::= CUBE <COMPOSITE GROUP BY TERM>
```

语法规则 48

```
<GROUP BY TERM> ::=
  <SIMPLE GROUP BY TERM>
  | <COMPOSITE GROUP BY TERM>
  | <ROLLUP TERM>
  | <CUBE TERM>
```

语法规则 49

```
<GROUP BY TERM LIST> ::=
  <GROUP BY TERM> [ { <COMMA> <GROUP BY TERM> } ⋯ ]
```

语法规则 50

```
<GROUP BY CLAUSE> ::= GROUP BY <GROUPING SPECIFICATION>
```

语法规则 51

```
<GROUPING SPECIFICATION> ::=
  <GROUP BY TERM LIST>
  | <SIMPLE GROUP BY TERM LIST> WITH ROLLUP
  | <SIMPLE GROUP BY TERM LIST> WITH CUBE
  | <GROUPING SETS SPECIFICATION>
```

语法规则 52

```
<GROUPING SETS SPECIFICATION> ::=
  GROUPING SETS <LEFT PAREN> <GROUP BY TERM LIST>
  <RIGHT PAREN>
```

语法规则 53

```
<ORDER SPECIFICATION> ::= ORDER BY <SORT SPECIFICATION LIST>
  <SORT SPECIFICATION LIST> ::= <SORT SPECIFICATION>
  [ { <COMMA> <SORT SPECIFICATION> } . . . ]
  <SORT SPECIFICATION> ::= <SORT KEY>
  [ <ORDERING SPECIFICATION> ] [ <NULL ORDERING> ]
  <SORT KEY> ::= <VALUE EXPRESSION>
  <ORDERING SPECIFICATION> ::= ASC | DESC
  <NULL ORDERING> := NULLS FIRST | NULLS LAST
```

另请参见

- 分布函数 （第 53 页）
- OLAP 优点 （第 20 页）
- OLAP 计算 （第 20 页）

- 排名函数 （第 43 页）
- 统计集合函数 （第 49 页）
- 窗口化 （第 33 页）
- 窗口化集合函数 （第 47 页）

Sybase IQ 作为数据服务器

Sybase IQ 通过 ODBC 或 JDBC 支持客户端应用程序连接。本章介绍如何将 Sybase IQ 用作客户端应用程序的数据服务器。

由于某些限制，Sybase IQ 还可能对于某些客户端应用程序表现为 Open Server。本章还简要说明了创建和运行这些应用程序的限制。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 编程”>“SQL Anywhere 数据访问 API”>“Sybase Open Client API”>“Open Client 体系结构”。

本章所介绍的功能不为 Windows 和 Sun Solaris 系统上的 IQ 用户提供远程数据访问。远程数据访问由组件集成服务 (CIS) 提供，CIS 为 OmniConnect™ 的核心互操作功能。

Sybase IQ 的客户端/服务器接口

为了简便起见，可将 Sybase 应用程序或第三方客户端应用程序与 Sybase IQ 一起使用。

了解这些部件如何协同工作可能会有助于您配置数据库和设置应用程序。本节介绍这些部件如何协同工作。有关第三方客户端应用程序的详细信息，请参见《安装和配置指南》。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“Open Client、Open Server 和 TDS”。

使用 iqdsedit 配置 IQ 服务器

Sybase IQ 可以与网络上的其它 Adaptive Server、Open Server 应用程序和客户端软件通信。

客户端可以与一个或多个服务器交谈，而服务器可以通过远程过程调用与其它服务器进行通信。为了使产品之间能够进行交互，每种产品都需要知道其它产品在网络中的位置。该网络服务信息存储在 interfaces 文件中。

注意： Sybase IQ 提供具有有限功能的、用于启用 INSERT...LOCATION 的各种版本 Open Client 实用程序，包括：

- iqisql
- iqdsedit
- iqdscp（仅 UNIX）
- iqocscfg（仅 Windows）

interfaces 文件

当使用 Open Client™ 程序连接到数据库服务器时，该程序在 interfaces 文件中查找服务器名，然后连接到使用该地址的服务器。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“interfaces 文件”。

iqdsedit 数据库管理实用程序

使用 **iqdsedit** 实用程序可以配置 interfaces 文件（interfaces 或 SQL.ini）。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“使用 DSEdit 实用程序”。

启动 iqdsedit

在 Windows 中，**iqdsedit** 可执行文件位于 `%SYBASE%\IQ-15_3\bin32` 或 `%SYBASE%\IQ-15_3\bin64` 目录中，该目录在安装期间自动添加到您的路径中。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“启动 DSEdit”。

打开目录服务会话

您可以在“选择目录服务”窗口中添加、修改或删除服务器条目（包括 Sybase IQ 服务器）。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“打开目录服务会话”。

添加服务器条目

服务器条目将显示在“服务器”字段中。要指定服务器的属性，您必须修改该条目。

此处输入的服务器名称不需要与在 Sybase IQ 命令行处提供的名称匹配。使用服务器地址（而非服务器名称）来标识和定位服务器。

服务器名称字段完全是 Open Client 的标识符。对于 Sybase IQ，如果服务器装载了多个数据库，则 IQDSEDIT 服务器名称条目用来标识要使用的数据库。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“添加服务器条目”。

添加或更改服务器地址

输入“服务器名”后，需要修改“服务器地址”以完成 interfaces 文件条目。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“添加或更改服务器地址”。

端口号
您输入的端口号必须与在 Sybase IQ 数据库服务器命令行处指定的端口匹配。Sybase IQ 服务器的缺省端口号为 2638。

以下为有效的服务器地址条目:

```
elora,2638
123.85.234.029,2638
```

另请参见
- 将数据库服务器作为 Open Server 启动 （第 79 页）

验证服务器地址
在 Windows 中，可以在“服务器对象”菜单中使用 **Ping** 服务器命令来验证网络连接。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“验证服务器地址”。

重命名服务器条目
可以在 `dsedit` 会话窗口中重命名服务器条目。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“重命名服务器条目”。

删除服务器条目
可以在 `dsedit` 会话窗口中删除服务器条目。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“配置 Open Server”>“删除服务器条目”。

Sybase 应用程序和 Sybase IQ

Sybase IQ 充当 Open Server 这一功能使 Sybase 应用程序（如 OmniConnect）能够与 Sybase IQ 协同工作。

要使用 Open Client 库，客户端应用程序只能使用支持的系统表、视图和存储过程。

OmniConnect 支持
Sybase OmniConnect 为组织中的不同数据提供统一的视图，使用户可以不必了解数据的模样或其所在的位置，即可访问多个数据源。此外，OmniConnect 对整个企业中的数据执行异构连接，实现了目标（例如 DB2、Sybase Adaptive Server® Enterprise、SQL Anywhere、Oracle 和 VSAM）的跨平台表连接。

通过使用 Open Server 接口，Sybase IQ 可以充当 OmniConnect 的数据源。

Open Client 应用程序和 Sybase IQ

您可以构建 Open Client 应用程序，以使用 Open Client 库直接从 C 或 C++ 编程环境（如 PowerSoft Power++™）访问 Sybase IQ 基表中的数据。如果此类应用程序引用目录表、视图或系统存储过程，则 Adaptive Server Enterprise（Transact-SQL™ 语法）和 Sybase IQ 都必须支持这些对象。

请参见《参考：构件块、表和过程》>“附录 A‘与其它 Sybase 数据库的兼容性’”。

配置 Open Client

在使用 Open Client 连接到 Sybase IQ 或使用 **INSERT...LOCATION** 语法时，可以在 Open Client 运行环境配置 (`.cfg`) 文件中设置多个 Open Client 配置参数。

例如，可以更改最大缺省连接数，该数量由 **CS_MAX_CONNECT** 选项的值控制。

INSERT...LOCATION 的应用程序名称为 `Sybase IQ`。（单词之间的空格是必需的。）此应用程序名称是在 Open Client 连接级别（而不是 Open Client 上下文级别）设置的。有关使用 Open Client 运行环境配置文件以及可用选项的详细信息，请参见 Open Client 的 Client-Library C Reference Manual。

要使 `.cfg` 生效，请停止并重新启动 Sybase IQ 服务器。此外，还可以在 **INSERT...LOCATION** 命令行中指定某些配置参数。在 **INSERT...LOCATION** 中设置的参数将被配置文件中设置的参数取代。

当用作远程服务器时，Sybase IQ 支持 Tabular Data Steam (TDS) 口令加密。Sybase IQ 服务器接受由客户端发送的使用加密口令的连接。有关针对口令加密设置的连接属性的信息，请参见《Open Server 15.5》>“Open Client Client-Library/C 参考手册”>“Client-Library 主题”>“安全性功能”>“Adaptive Server Enterprise 安全性功能”>“安全握手：加密口令”。

注意： 口令加密需要 Open Client 15.0。TDS 口令加密需要 Open Client 15.0 ESD #7 或更高版本。

要使 Sybase IQ 服务器接受具有加密口令的 jConnect 连接，请将 jConnect **ENCRYPT_PASSWORD** 连接属性设置为 true。

Sybase IQ 作为 Open Server

本节介绍如何设置 Sybase IQ 服务器，以接收来自 Open Client 应用程序的连接。

系统要求

对使用 Sybase IQ 作为 Open Server 的客户端和服务器有单独的要求。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“将 SQL Anywhere 设置为 Open Server”>“系统要求”。

注意： 使用 OmniConnect 从本地 SQL Anywhere Enterprise 服务器连接到远程 Sybase IQ 时，请使用以下服务器类：

- 要连接到 Sybase IQ 12 或更高版本，请使用服务器类 `asaodbc` 和 `sajdbc`。
- 要连接到 Sybase IQ 11.x，请使用服务器类 `asiq`。

将数据库服务器作为 Open Server 启动

如果您要将 Sybase IQ 用作 Open Server，必须确保使用 TCP/IP 协议启动它。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“将 SQL Anywhere 设置为 Open Server”>“将数据库服务器作为 Open Server 启动”。

计算机上每个使用 TCP/IP 的应用程序都使用不同的 TCP/IP *端口*，以使网络包发送到正确的应用程序。Sybase IQ 的缺省端口为 2638，该端口用于共享内存通信。您可以指定另一个端口号：

```
start_iq -x tcpip{port=2629} -n myserver iqdemo.db
```

另请参见

- 添加或更改服务器地址 （第 76 页）

配置数据库以与 Open Client 一起使用

数据库必须为 Sybase IQ 12.0 或更高版本。

如果要将 Sybase IQ 与 Adaptive Server Enterprise 一起使用，需要确保创建的数据库与 Adaptive Server Enterprise 具有最大的兼容性。

连接到作为 Open Server 的 Sybase IQ 时，应用程序经常假定已提供它们在 Adaptive Server Enterprise 下所期望的服务。这些服务并非总是存在。

请参见《参考：构件块、表和过程》>“附录 A ‘与其它 Sybase 数据库的兼容性’”。

Open Client 和 jConnect 连接的特性

当 Sybase IQ 通过 TDS 为应用程序提供服务时，它会自动将相关数据库选项的值设置为与 SQL Anywhere Server 的缺省行为兼容。这些选项是临时设置的，它们仅用于此连接期间。客户端应用程序可以随时覆盖这些选项。

注意： Sybase IQ 不支持 `ANSI_BLANKS`、`FLOAT_AS_DOUBLE` 和 `TSQL_HEX_CONSTANT` 选项。

虽然 Sybase IQ 允许更长的用户名和口令，但 TDS 客户端用户名和口令不能超过 30 字节。如果您的口令或用户 ID 超过 30 字节，则尝试通过 TDS 进行连接（例如，使用 jConnect）会返回“用户 `ID` 或口令无效”错误。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“复制”>“将 SQL Anywhere 用作 Open Server”>“将 SQL Anywhere 设置为 Open Server”>“Open Client 和 jConnect 连接的特性”。

注意： ODBC 应用程序（包括 Interactive SQL 应用程序）会自动将某些数据库选项设置为 ODBC 规范所规定的值。这样将覆盖通过 `LOGIN_PROCEDURE` 数据库选项所做的设置。有关详细信息和解决方法，请参见《参考：语句和选项》>“LOGIN_PROCEDURE 选项”。

具有多个数据库的服务器

使用 Open Client Library，可以连接到包含多个数据库的服务器上的特定数据库。

- 在服务器的 `interfaces` 文件中设置条目。
- 使用 **start_iq** 命令的 **-n** 参数为数据库名称设置快捷方式。
- 在 **isql** 命令中用数据库名称指定 **-S** *database_name* 参数。进行连接时，需要此参数。

可以在不更改程序本身的情况下，通过将快捷方式名称放到程序中并只更改快捷方式定义，对多个数据库运行相同的程序。

例如，以下 `interfaces` 文件摘录定义了 `live_sales` 和 `test_sales` 两个服务器：

```
live_sales

    query tcp ether myhostname 5555     master tcp ether myhostname
5555

test_sales

    query tcp ether myhostname 7777     master tcp ether myhostname
7777
```

启动服务器并为特定数据库设置别名。以下命令将 `live_sales` 设置为等效于 `salesbase.db`：

```
start_iq -n sales_live <other parameters> -x \ ‘tcpip{port=5555}’
salesbase.db -n live_sales
```

要连接到 `live_sales` 服务器：

```
isql -Udba -Psql -Slive_sales
```

一个服务器名称只能在 `interfaces` 文件中出现一次。因为与 Sybase IQ 的连接现在基于数据库名称，所以数据库名称必须唯一。如果所有脚本都设置为在 `salesbase` 数据库上工作，则不必修改脚本，脚本即可与 `live_sales` 或 `test_sales` 一起工作。

访问远程数据

Sybase IQ 可以访问位于单独的服务器（Sybase 和非 Sybase）上的数据，就像这些数据存储在本地服务器上一样。

Sybase IQ 和远程数据

SQL Anywhere 远程数据访问允许您访问其它数据源中的数据。您可以使用此功能将数据迁移到 SQL Anywhere 数据库。您还可以使用此功能查询各个数据库中的数据。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“Open Client 和 jConnect 连接的特性”。

访问远程数据的要求

访问远程数据需要具备几个基本元素。

远程表映射

Sybase IQ 将表提供给客户端应用程序的方式就像表中的所有数据都存储在与该应用程序连接的数据库中一样。

从内部来看，当 Sybase IQ 执行涉及远程表的查询时，它会确定存储位置并访问远程位置来检索数据。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“远程表映射”。

服务器类

为每个远程服务器指派一个*服务器类*。服务器类指定用于与服务器进行交互的访问方法。不同类型的远程服务器要求使用不同的访问方法。

服务器类为 Sybase IQ 提供详细的服务器功能信息。Sybase IQ 根据这些功能对它与远程服务器的交互进行调整。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“服务器类”。

注意： IPv6 不支持 OMNI JDBC 类。

远程服务器

必须先定义远程对象所在的远程服务器，然后才能将远程对象映射到本地代理表。

创建远程服务器

使用 **CREATE SERVER** 语句设置远程服务器定义。

对于某些系统（包括 Sybase IQ 和 SQL Anywhere），每个数据源都描述一个数据库，因此每个数据库都需要一个单独的远程服务器定义。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用远程服务器”>“使用 CREATE SERVER 语句创建远程服务器”。

装载没有本机类的远程数据

使用 DirectConnect 装载数据。

本机类使用 DirectConnect 来访问远程数据源：

- 在 64 位 UNIX 平台上
- 在没有 ODBC 驱动程序可用的 32 位平台上（例如 Microsoft SQL Server）

将 MS SQL Server 数据装载到 UNIX 上的 IQ 服务器中

本远程数据示例将 MS SQL Server 数据装载到 UNIX 上的 IQ 服务器中。

对于本示例，假设以下情况：

- 名为 *mssql* 的 Enterprise Connect Data Access (ECDA) 服务器位于 UNIX 主机 *myhostname* 端口 12530 上。
- 要从主机 *myhostname* 端口 1433 上的 MS SQL Server *2000* 中检索数据。

1. 使用 DirectConnect 文档为您的数据源配置 DirectConnect。
2. 确保 ECDA 服务器 (*mssql*) 列在 Sybase IQ interfaces 文件中：

   ```
   mssql
   master tcp ether myhostname 12530
   query tcp ether myhostname 12530
   ```
3. 使用服务器 *mssql* 的用户 ID 和口令，添加新用户：

   ```
   isql -Udba -Psql -Stst_iqdemo
   grant connect to chill identified by chill
   grant dba to chill
   ```
4. 以新用户身份登录，在 Sybase IQ 上创建本地表：

   ```
   isql -Uchill -Pchill -Stst_iqdemo
   create table billing(status char(1), name varchar(20), telno int)
   ```
5. 插入数据：

   ```
   insert into billing location ‘mssql.pubs’ { select * from
   billing }
   ```

查询没有本机类的数据

请遵循以下准则来查询没有本机类的数据。

1. 使用远程服务器和代理配置 ASE/CIS，以通过 DirectConnect 连接。例如，将 DirectConnect for Oracle 用于 Oracle 服务器。
2. 使用 ASE 服务器类 ASEJDBC 为 Sybase IQ 配置远程服务器。（ASEODBC 类不可用，因为 ASE 没有 64 位 Unix ODBC 驱动程序。）
3. 使用 **CREATE EXISTING TABLE** 语句创建指向 ASE 中的代理表的代理表，而 ASE 中的代理表又指向 Oracle。

使用 DirectConnect 和 UNIX 中的代理表查询远程数据

使用 DirectConnect 查询数据。

本示例显示如何访问 MS SQL Server 数据。对于本示例，假设情况如下所示：

- Sybase IQ 服务器位于主机 *myhostname* 端口 7594 上。
- Adaptive Server Enterprise 服务器位于主机 *myhostname* 端口 4101 上。
- 名为 *mssql* 的 Enterprise Connect Data Access (ECDA) 服务器位于主机 *myhostname* 端口 12530 上。
- 要从主机 *myhostname* 端口 1433 上的 MS SQL Server *2000* 中检索数据。

设置 Adaptive Server Enterprise 以查询 MS SQL Server

设置 Adaptive Server 和组件集成服务 (CIS) 以通过 DirectConnect 查询 MS SQL Server。

对于本示例，假设服务器名为 *jones_1207*。

1. 将一个条目添加到 ASE interfaces 文件，以连接至 *mssql*：

```
mssql

master tcp ether hostname 12530

query tcp ether hostname 12530
```

2. 从 ASE 服务器启用 CIS 和远程过程调用处理。例如，如果缺省情况下，CIS 已启用：

```
sp_configure 'enable cis'

Parameter Name Default Memory Used Config Value Run Value

enable cis                          1         0
1             1

(1 row affected)
(return status=0)

sp_configure 'cis rpc handling', 1

Parameter Name Default Memory Used Config Value Run Value
```

```
enable cis                          0         0
0             1
```

```
(1 row affected)
Configuation option changed. The SQL Server need not be restarted
since the option is dynamic.
```

在旧版本（例如 Sybase IQ 12.5 版）中，启用 CIS 远程过程调用处理之后，可能需要重新启动 Adaptive Server Enterprise 服务器。

3. 将 DirectConnect 服务器添加到 ASE 服务器的 SYSSERVERS 系统表中。

```
sp_addserver mssql, direct_connect, mssql
```

```
Adding server 'mssql', physical name 'mssql'
Server added.
(Return status=0)
```

4. 在 Adaptive Server Enterprise 中创建用户，该用户在 Sybase IQ 中用于连接到 ASE。

```
sp_addlogin tst, tsttst
```

```
Password correctly set.
Account unlocked. New login created.
(return status = 0)
```

```
grant role sa_role to tst
use tst_db
sp_adduser tst
```

```
New user added.
(return status = 0)
```

5. 从主数据库添加外部登录名：

```
use master
sp_addexternlogin mssql, tst, chill, chill
```

```
User 'tst' will be known as 'chill' in remote server 'mssql'.
(return status = 0)
```

6. 从所需数据库，以所添加用户身份创建 ASE 代理表：

```
isql -Utst -Ttsttst
use test_db
create proxy_table billing_tst at 'mssql.pubs..billing'
select * from billing_tst
```

```
status                name                telno
------                -----------               -----
D                BOTANICALLY               1
B                BOTANICALL                2
(2 rows affected)
```

设置 Sybase IQ 以连接到 ASE 服务器

请按照以下步骤来查询 Adaptive Server Enterprise 数据。

1. 将条目添加到 Sybase IQ interfaces 文件：

```
jones_1207
master tcp ether jones 4101
query tcp ether jones 4101
```

2. 创建用户以连接到 ASE：

```
grant connect to tst identified by tsttst
grant dba to tst
```

3. 以所添加用户的身份登录，创建“asejdbc”服务器类并添加外部登录名：

```
isql -Utst -Ptsttst -Stst_iqdemo
create SERVER jones_1207 CLASS 'asejdbc' USING 'jones:4101/tst_db'
create existing table billing_iq at
'jones_1207.tst_db..billing_txt'
select * from billing_iq
```

```
status                name                telno
------                -----------                -----
D                BOTANICALLY                1
B                BOTANICALL                2
(2 rows affected)
```

删除远程服务器

使用 Sybase Central 或 **DROP SERVER** 语句从 ISYSSERVER 系统表中删除远程服务器。

必须删除在该服务器上定义的所有远程表才能使此操作成功。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用远程服务器”>“删除远程服务器”。

更改远程服务器

使用 **ALTER SERVER** 语句修改服务器的属性。这些更改直到下一次与远程服务器连接时才生效。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用远程服务器”>“更改远程服务器”。

列出服务器上的远程表

配置 Sybase IQ 时，拥有特定服务器上所提供远程表的列表的访问权限会很有帮助。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用远程服务器”>“列出服务器上的远程表”。

另请参见《参考：构件块、表和过程》>“sp_remote_tables 系统过程”。

列出远程服务器功能

sp_servercaps 过程显示有关远程服务器功能的信息。Sybase IQ 使用此功能信息来确定可以向远程服务器传递多少 SQL 语句。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用远程服务器”>“列出远程服务器功能”。

另请参见《参考：构件块、表和过程》>“sp_servercaps 系统过程”。

外部登录

Sybase IQ 代表其客户端连接到远程服务器时会使用其客户端的名称和口令。但是，您可以通过创建外部登录名来取代这一行为。

外部登录名是与远程服务器通信时使用的替代登录名和口令。

当 Sybase IQ 连接到远程服务器时，**INSERT...LOCATION** 将使用当前连接的用户 ID 的远程登录名，但前提是已使用 **CREATE EXTERNLOGIN** 创建远程登录名并且已使用 **CREATE SERVER** 语句定义了远程服务器。如果未定义远程服务器，或者未为当前连接的用户 ID 创建远程登录，则 IQ 将使用当前连接的用户 ID 和口令连接。有关 **INSERT...LOCATION** 使用远程登录名的详细信息和示例，请参见《参考：语句和选项》>“INSERT 语句”。

如果您使用集成登录，则 Sybase IQ 客户端的 Sybase IQ 名称和口令将与 Sybase IQ 用户 ID 映射到 `syslogins` 中的数据库登录 ID 和口令相同。

创建外部登录名

只有 DBA 帐户或具有 USER ADMIN 权限的帐户才能添加或修改外部登录名。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用远程服务器”>“创建外部登录名”。

有关详细信息，请参见《参考：语句和选项》>“SQL 语句”>“CREATE EXTERNLOGIN 语句”。

删除外部登录名

使用 **DROP EXTERNLOGIN** 语句从 Sybase IQ 系统表中删除外部登录名。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用远程服务器”>“删除外部登录名”。

有关详细信息，请参见《参考：语句和选项》>“SQL 语句”>“DROP EXTERNLOGIN 语句”。

代理表

远程数据的位置透明性是通过创建映射到远程对象的本地代理表启用的。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用代理表”。

指定代理表位置

CREATE TABLE 和 **CREATE EXISTING TABLE** 都可以使用 **AT** 关键字定义现有对象的位置。

该位置字符串由句点或分号隔开的四部分组成。分号允许在数据库和所有者字段中使用文件名和扩展名。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用代理表”>“指定代理表位置”。

示例

下面的示例说明了位置字符串的用法：

- Sybase IQ：

```
'testiq..DBA.employee'
```

创建代理表

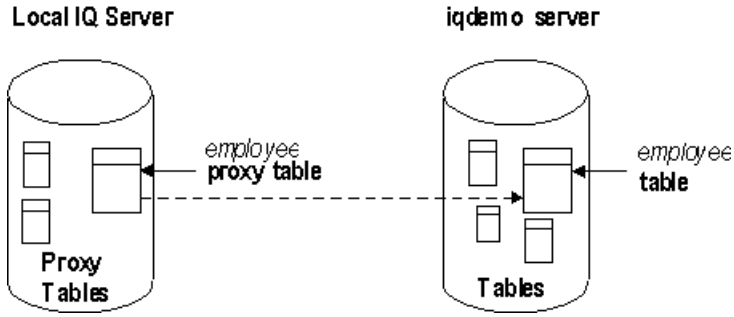
CREATE EXISTING TABLE 语句创建映射到远程服务器上现有表的代理表。

Sybase IQ 从位于远程位置的对象派生列属性和索引信息。

示例

要在当前服务器上创建一个映射到 `iqdemo1` 服务器上 `employee` 远程表的名为 `p_employee` 的代理表，请使用以下语法：

```
CREATE EXISTING TABLE p_employee
AT 'iqdemo1..DBA.employee'
```



请参见《参考：语句和选项》>“CREATE EXISTING TABLE 语句”。

CREATE TABLE 语句

CREATE TABLE 语句在远程服务器上创建新表，如果使用 **AT** 选项，则可以为该表定义代理表。

使用 Sybase IQ 数据类型定义列。Sybase IQ 会自动将数据转换为远程服务器的本机类型。

如果您使用 **CREATE TABLE** 语句创建本地表和远程表，并随后使用 **DROP TABLE** 语句删除代理表，则远程表也将被删除。但是，您可以使用 **DROP TABLE** 语句来删除使用 **CREATE EXISTING TABLE** 语句创建的代理表。这种情况下，未删除远程表。

示例

以下语句在远程服务器 `iqdemo1` 上创建名为 `Employees` 的表，并创建映射到该远程位置的名为 `members` 的代理表：

```
CREATE TABLE members
( membership_id INTEGER NOT NULL,
member_name CHAR(30) NOT NULL,
office_held CHAR( 20 ) NULL)
AT 'iqdemo1..DBA.Employees'
```

有关详细信息，请参见《参考：语句和选项》>“INSERT 语句”。

列出远程表上的列

sp_remote_columns 系统过程生成远程表上列的列表和这些数据类型的说明。

如果您正在输入 **CREATE EXISTING TABLE** 语句并指定列的列表，则获取远程表上可用列的列表可能很有帮助。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用代理表”>“列出远程表上的列”。

有关详细信息，请参见《参考：构件块、表和过程》>“系统过程”>“系统存储过程”>“目录存储过程”>“sp_remote_columns 系统过程”。

示例：两个远程表之间的连接

下图说明演示数据库中映射到名为 `testiq` 的本地服务器的远程 Sybase IQ 表 `employee` 和 `department`。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“连接远程表”。

多个本地数据库

Sybase IQ 服务器上可能有多个本地数据库在同时运行。通过将其它本地 Sybase IQ 数据库中的表定义为远程表，您可以执行跨数据库连接。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“连接多个本地数据库中的表”。

将本机语句发送到远程服务器

使用 **FORWARD TO** 语句将一个或多个语句以其本机语法发送到远程服务器。

可以通过两种方式使用此语句。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“将本机语句发送到远程服务器”。

远程过程调用 (RPC)

Sybase IQ 用户可以向支持过程调用功能的远程服务器发出过程调用。

Sybase IQ、SQL Anywhere 和 Adaptive Server Enterprise 以及 Oracle 和 DB2 都支持此功能。发出远程过程调用与使用本地过程调用类似。

创建远程过程

使用以下过程之一发出远程过程调用。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“使用远程过程调用 (RPC)”>“创建远程过程”。

事务管理和远程数据

事务提供一种对 SQL 语句进行分组的方式，这样可以将这些语句作为一个整体对待，即要么将这些语句执行的所有工作都提交到数据库，要么一个也不提交。

对远程表的事务管理和对本地 Sybase IQ 表的处理方法稍有不同。对远程表的事务管理的处理方法在很大程度上就像在 SQL Anywhere 中一样，但有一些差异。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“创建数据库”>“使用事务和隔离级别”。

有关 Sybase IQ 中事务的一般性讨论，请参见《系统管理指南第一卷》>“事务和版本控制”。

远程事务管理概述

这种涉及远程服务器的事务管理方法使用*两阶段提交*协议。

Sybase IQ 执行的策略可以确保大多数情况的事务完整性。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“事务管理和远程数据”>“远程事务管理概述”。

事务管理的限制

事务管理具有保存点和嵌套语句限制。

事务管理的限制为:

- 不将保存点传播到远程服务器。
- 如果涉及远程服务器的事务中包含嵌套的 **BEGIN TRANSACTION** 和 **COMMIT TRANSACTION** 语句，则只处理最外层的一组语句。系统不会将包含 **BEGIN TRANSACTION** 和 **COMMIT TRANSACTION** 语句的最里层一组语句传输到远程服务器。

内部操作

本节介绍 SQL Anywhere 代表客户端应用程序对远程服务器执行的基础操作。

查询分析

当从客户端接收到语句时，数据库服务器将对该语句进行分析。如果该语句不是有效的 SQL Anywhere SQL 语句，数据库服务器将引发错误。

查询规范化

在查询规范化中，系统将验证引用的对象并检查数据类型兼容性。

例如，请考虑下列查询:

```
SELECT *
FROM t1
WHERE c1 = 10
```

查询规范化阶段验证系统表中是否存在具有列 `c1` 的表 `t1`。它还验证列 `c1` 的数据类型是否和值 10 兼容。例如，如果该列的数据类型为 `DATETIME`，则系统将拒绝此语句。

查询预处理

查询预处理准备进行优化的查询。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“内部操作”>“查询预处理”。

服务器功能

定义到 Sybase IQ 的每个远程服务器都有一组与其关联的功能。这些功能存储在 syscapabilities 系统表中。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“内部操作”>“服务器功能”。

语句的完整直通

处理语句最有效的方式通常是尽可能多地向涉及的远程服务器传递原始语句。

缺省情况下，Sybase IQ 尝试尽可能多地传递语句。在很多情况下，这是最初提供给 Sybase IQ 的完整语句。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“内部操作”>“语句的完整直通”。

语句的部分直通

如果某个语句包含对多个服务器的引用，或者使用远程服务器不支持的 SQL 功能，则会将查询分解为多个较简单的部分。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“内部操作”>“语句的部分直通”。

远程数据访问故障排除

本节提供远程服务器访问故障排除的一些建议。

不支持远程数据的功能

有些功能从不受 Sybase IQ 支持。有些功能仅针对本地数据而受到支持。

Sybase IQ 对 SQL Anywhere 列表具有以下附加功能：

- 不支持 Java 数据类型。
- 在某些地理区域中使用组件集成服务 (CIS) 时，连接尝试将返回错误

  “无合适的驱动程序”

  。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“远程数据访问故障排除”>“不支持远程数据的功能”。

区分大小写

您的 Sybase IQ 数据库的区分大小写设置应该和所访问的任何远程服务器使用的设置匹配。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“远程数据访问故障排除”>“区分大小写”。

连接问题

若要验证您是否可以连接到远程服务器，请执行到远程服务器的简单的直通语句以检查您的连接和远程登录配置。

例如：

```
FORWARD TO testiq {select @@version}
```

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“远程数据访问故障排除”>“连接测试”。

查询的一般问题

如果您遇到某些类型的问题涉及 Sybase IQ 对远程表处理查询的方式，则了解 Sybase IQ 如何执行该查询通常会很有帮助。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“远程数据访问故障排除”>“查询的一般问题”。

管理远程数据访问连接

如果您通过 ODBC 访问远程数据库，到远程服务器的连接就会被指定一个名称。

该名称可用于断开连接，作为取消远程请求的一种方法。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“访问远程数据”>“远程数据访问故障排除”>“管理通过 ODBC 执行的远程数据访问连接”。

用于进行远程数据访问的服务器类

本章介绍 Sybase IQ 如何与各种服务器类相连接。

服务器类概述

远程连接的行为由 **CREATE SERVER** 语句中的服务器类决定。服务器类为 Sybase IQ 提供详细的服务器功能信息。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”。

基于 JDBC 的服务器类

当 Sybase IQ 在内部使用 Java 虚拟机和 jConnect™ for JDBC™ 5.5 连接到远程服务器时，使用的便是基于 JDBC 的服务器类。

基于 JDBC 的服务器类有：

- Sybase IQ 和 SQL Anywhere
- Sybase SQL Anywhere 和 Adaptive Server Enterprise（版本 10 和更高版本）。

JDBC 类的配置说明

在访问用基于 JDBC 的类定义的远程服务器时，请考虑最佳性能、远程服务器访问和远程服务器连接。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 JDBC 的服务器类”>“JDBC 类的配置说明”。

服务器类 sajdbc

对于 Sybase IQ 或 SQL Anywhere 数据源，没有任何特殊的配置要求。

CREATE SERVER 语句中的参数值

CREATE SERVER 语句中的 **USING** 参数采用以下形式：*hostname:portnumber [/ databasename]*。

其中：

- **hostname** – 运行远程服务器的计算机

- **portnumber** – 远程服务器监听的 TCP/IP 端口号。Sybase IQ 监听的缺省端口号是 2638。
- **databasename** – 连接将使用的 Sybase IQ 数据库。这是启动服务器时在 **-n** 开关中指定的名称，或是在 **DBN (DatabaseName)** 连接参数中指定的名称。

Sybase IQ 示例
若要配置名为 `testiq` 的 Sybase IQ 服务器（该服务器位于 apple 计算机上，并监听端口号 2638），请使用：

```
CREATE SERVER testiq
CLASS 'sajdbc'
USING 'apple:2638'
```

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 JDBC 的服务器类”>“服务器类 sajdbc”>“CREATE SERVER 语句中的 USING 参数”。

服务器类 asejdbc

具有服务器类 asejdbc 的服务器可以是 Adaptive Server Enterprise 或 SQL Anywhere 版本 10 和更高版本。

虽然 Adaptive Server Enterprise 数据源通常不需要进行特殊配置，但是 ASE 15.5 需要 jConnect-6_0 元数据存储过程和表。请参见《访问远程数据》>“用于远程数据访问的服务器类”>“基于 JDBC 的服务器类”>“服务器类 aseodbc”>“安装 jConnect 6.0 元数据”。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 JDBC 的服务器类”>“服务器类 asejdbc”。

数据类型转换

当您发出 **CREATE TABLE** 语句创建代理表时，Sybase IQ 会自动将数据类型转换为相应的 Adaptive Server Enterprise 数据类型。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 JDBC 的服务器类”>“服务器类 asejdbc”。

安装 jConnect 6.0 元数据

Adaptive Server Enterprise 15.5 数据的代理表需要使用 jConnect 6.0 元数据。

如果没有 jConnect 6.0 元数据，**CREATE EXISTING TABLE** 语句可能会返回错误“SQL Anywhere 错误 -667: 无法访问表的列信息”。

使用 **isql**：

1. 连接到 Adaptive Server Enterprise 数据库。

2. 按以下格式输入命令：

```
isql -I<path to interfaces>
-Usa -P
-S<ASE_server>
-i$SYBASE/jConnect-6_0/sp/sql_server15.0.sql
```

基于 ODBC 的服务器类

Sybase IQ 支持多种基于 ODBC 的服务器类。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”。

ODBC 外部服务器

定义基于 ODBC 的服务器的最常用方法是将其基于 ODBC 数据源。

要进行这样的定义，您必须在 ODBC 管理器中创建数据源名称 (DSN)。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“定义 ODBC 外部服务器”。

Sybase IQ 示例
与 Sybase IQ 的连接可以是：

```
CREATE SERVER testiq
CLASS 'asaodbc'
USING 'driver=adaptive server IQ 12.0;
eng=testasaiq;dbn=iqdemo;links=tcpip{}'
```

有关创建 Sybase IQ 的 ODBC 数据源的详细信息，请参见《系统管理指南第一卷》>“Sybase IQ 连接”>“ODBC 数据源”。

服务器类 saodbc

若要访问可以支持多个数据库的 SQL Anywhere 数据库服务器，请创建 ODBC 数据源名称，并定义与每一个数据库的连接。为创建的每一个 ODBC 数据源名称发出 **CREATE SERVER** 语句。

具有服务器类 `saodbc` 的服务器如下所示：

- Sybase IQ 版本 12 或更高版本
- SQL Anywhere

对于 SQL Anywhere 或 Sybase IQ 数据源，没有任何特殊的配置要求。

服务器类 aseodbc

Sybase IQ 要求必须在本地安装 Adaptive Server Enterprise ODBC 驱动程序和 Open Client 连接库，才能连接到具有 `aseodbc` 类的远程 Adaptive Server。不过，其性能要优于具有 `asejdbc` 类的 Adaptive Server。

具有服务器类 `aseodbc` 的服务器为：

- Adaptive Server Enterprise
- SQL Anywhere（版本 10 和更高版本）

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“服务器类 aseodbc”。

服务器类 db2odbc

具有服务器类 db2odbc 的服务器是 IBM DB2。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“服务器类 db2odbc”。

服务器类 oraodbc

使用服务器类 oraodbc 的服务器是 Oracle 10.0 或更高版本。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“服务器类 oraodbc”。

Sybase IQ 到 Oracle 数据类型映射

当您使用 CREATE TABLE 语句在 Oracle 服务器上创建远程表时，Sybase IQ 会将 IQ 数据类型转换为相应的 Oracle 数据类型：

表 5. 数据映射到新的远程 Oracle 表

Sybase IQ 数据类型	Oracle 数据类型
BIGINT	NUMBER(20,0)
BINARY(n)	if (n > 255) LONG RAW else RAW(n)
BIT	NUMBER(1,0)
CHAR(n)	If (n > 255) LONG else VARCHAR(n)
CHARACTER VARYING(n)	VARCHAR2(n)
CHARACTER(n)	VARCHAR2(n)

Sybase IQ 数据类型	Oracle 数据类型
DATE	DATE
DATETIME	DATE
DECIMAL(prec, scale)	NUMBER(prec, scale)
DOUBLE	FLOAT
FLOAT	FLOAT
INT	NUMBER(11,0)
LONG BINARY	LONG RAW
LONG VARCHAR	LONG or CLOB
MONEY	NUMBER(19,4)
NUMERIC(prec, scale)	NUMBER(prec, scale)
REAL	FLOAT
SMALLDATETIME	DATE
SMALLINT	NUMBER(5,0)
SMALLMONEY	NUMBER(10,4)
TIME	DATE
TIMESTAMP	DATE
TINYINT	NUMBER(3,0)
UNIQUEIDENTIFIERSTR	CHAR(36)
UNSIGNED BIGINT	NUMBER(20,0)
UNSIGNED INT	NUMBER(11,0)
UNSIGNED INTEGER	NUMBER(11,0)
VARBINARY(n)	if (n > 255) LONG RAW else RAW(n)
VARCHAR(n)	VARCHAR2(n)

Oracle 到 Sybase IQ 数据映射

当您使用 **CREATE EXISTING** 语句为现有的 Oracle 表创建代理表时，Sybase IQ 会将 Oracle 数据类型转换为相应的 IQ 数据类型。

表 6. 数据映射到现有的 Oracle 表

Oracle 数据类型	IQ 数据类型
BFILE	LONG BINARY
BLOB	LONG BINARY
CHAR(n)	CHAR(n)
CLOB	LONG VARCHAR
DATE	TIMESTAMP
DEC(prec, scale)	NUMERIC(prec, scale)
DECIMAL(prec, scale)	NUMERIC(prec, scale)
DOUBLE PRECISION	DOUBLE
FLOAT	DOUBLE
INT	NUMERIC(38.0)
INTEGER	NUMERIC(38.0)
NCHAR(n)	NCHAR(n)
NCLOB	LONG NVARCHAR
NUMBER(prec, scale)	NUMERIC(prec, scale)
NUMERIC(prec, scale)	NUMERIC(prec, scale)
NVARCHAR2(n)	VARCHAR(n)
RAW(n)	VARBINARY(n)
REAL	DOUBLE
SMALLINT	NUMERIC(38.0)
TIMESTAMP	TIMESTAMP
VARCHAR2(n)	VARCHAR(n)

注意：

- Sybase IQ 允许您将代理表映射到 Oracle 视图。由于 Oracle 标识符始终以大写字母显示，因此您必须使用大写字母来创建或引用要映射到 Oracle 视图的任何代理表。
- 请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“服务器类 oraodbc”。

服务器类 mssodbc

具有 mssodbc 服务器类的服务器是 Microsoft SQL Server 6.5 版（带 Service Pack 4）。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“服务器类 mssodbc”。

服务器类 odbc

不具有自己的服务器类的 ODBC 数据源使用 odbc 服务器类。您可以使用任何 ODBC 驱动程序。

可以通过 Microsoft 数据访问组件 (MDAC) 发布（可在 Microsoft 下载中心上找到）获得 Microsoft ODBC 驱动程序的最新版本。列出的 Microsoft 驱动程序版本是 MDAC 2.0 的组成部分。

Microsoft Excel (Microsoft 3.51.171300)

每一个 Excel 工作簿都可以被视为包含若干个表的数据库。

数据库中的表可以映射到工作簿中的工作表。当您在 ODBC 驱动程序管理器中配置 ODBC 数据源名称时，可以指定与该数据源相关联的缺省工作簿名称。不过，当您发出 **CREATE TABLE** 语句时，可以替换该缺省值并在位置字符串中指定工作簿名称。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“服务器类 odbc”>“Microsoft Excel (Microsoft 3.51.171300)”。

Microsoft Foxpro (Microsoft 3.51.171300)

您可以将多个 Foxpro 表一起存储在一个 Foxpro 数据库文件 (`.dbc`) 内；或者，可以将每个表单独存储在各个 `.dbf` 文件中。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“服务器类 odbc”>“Microsoft FoxPro (Microsoft 3.51.171300)”。

Lotus Notes SQL 2.0 (2.04.0203)

您可以从 Lotus Web 站点获取 Lotus Notes SQL 2.0 (2.04.0203) 驱动程序。

请阅读附随的文档，以便了解有关如何将 Notes 数据映射到关系表的说明。您可以很轻松地将 IQ 表映射到 Notes 表单。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“远程数据和批量操作”>“用于远程数据访问的服务器类”>“基于 ODBC 的服务器类”>“服务器类 odbc”>“Lotus Notes SQL 2.0”。

设置 IQ 访问 Address 示例文件

设置 IQ 访问 Address 示例文件。

1. 使用 NotesSQL 驱动程序创建 ODBC 数据源。

   数据库将是示例名文件 `c:\notes\data\names.nsf`。应选中“映射特殊字符”选项。对于此示例，“Data Source Name”是 *my_notes_dsn*。

2. 创建 IQ 服务器：

   ```
   CREATE SERVER names
   CLASS 'odbc'
   USING 'my_notes_dsn'
   ```

3. 将 Person 表单映射到 IQ 表中：

   ```
   CREATE EXISTING TABLE Person
   AT 'names...Person'
   ```

4. 查询该表

   ```
   SELECT * FROM Person
   ```

使用日程表和事件自动完成任务

本章介绍如何使用 Sybase IQ 的调度和事件处理功能来自动执行数据库管理和其它任务。

调度和事件处理简介

对于很多数据库管理任务，最好是有计划有步骤地将其完成。

例如，定期备份过程是合理数据库管理过程的重要组成部分。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“日程表和事件使用简介”。

日程表

通过对活动进行调度，您可以确保在一组预置的时间执行一组操作。调度信息和事件处理程序都存储在数据库本身中。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“了解日程表”。

Sybase IQ 示例

注意： 例如，使用 Sybase IQ 演示数据库 `iqdemo.db`。

```
Create table OrderSummary(c1 date, c2 int);create event
Summarizeschedulestart time '6:00 pm'on ('Mon', 'Tue', 'Wed', 'Thu',
'Fri')handlerbegin   insert into DBA.OrderSummary   select
max(OrderDate), count(*)   from GROUPO.SalesOrders where OrderDate =
current dateend
```

定义日程表

为了增加灵活性，日程表定义由多个组件组成。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“了解日程表”>“定义日程表”。

事件

数据库服务器会跟踪几种系统事件。在数据库服务器检查某系统事件且相应事件符合提供的*触发条件*时，会触发事件处理程序。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“了解系统事件”。

选择系统事件

Sybase IQ 跟踪若干个系统事件。每个系统事件都提供一个您可以在其上挂接一组操作的挂钩。

数据库服务器为您跟踪事件，并在需要时执行操作（操作在事件处理程序中定义）。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“了解系统事件”。

定义事件的触发器状态

每个事件定义都有一个与其关联的系统事件。该事件也有一个或多个触发条件。

在满足系统事件的触发条件时，会触发事件处理程序。

注意： 与 Sybase IQ 事件关联的触发器状态不同于 SQL Anywhere 触发器或 Adaptive Server Enterprise 触发器，后两种触发器在用户尝试在指定表上执行指定的数据修改语句时自动执行。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“了解系统事件”>“定义事件的触发器状态”。

Sybase IQ 示例

注意： 例如，使用 Sybase IQ 演示数据库 `iqdemo.db`。

```
create event SecurityCheck
type ConnectFailed
handler
begindeclare num_failures int;declare mins int;

insert into FailedConnections( log_time )values ( current
timestamp );
select count( * ) into num_failuresfrom FailedConnectionswhere
log_time >= dateadd( minute, -5,
    current timestamp );if( num_failures >= 3 ) then
    select datediff( minute, last_notification,        current
timestamp ) into mins    from Notification;
    if( mins > 30 ) then        update Notification        set
last_notification = current timestamp;        call
```

```
xp_sendmail( recipient='DBAdmin',                subject='Security
Check',"message"=        'over 3 failed connections in last 5
minutes' )    end ifend ifend
```

事件处理程序

事件处理程序不在触发相应事件的操作所在的连接上执行，因此不与客户端应用程序进行交互。它们是使用事件创建者的权限执行的。

开发事件处理程序

不管是用于调度事件的处理，还是用于系统事件处理，事件处理程序都包含复合语句，而且在很多方面都与存储过程类似。可以添加循环、条件执行等等，还可以使用 Sybase IQ 调试工具来调试事件处理程序。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“了解系统事件”>“开发事件处理程序”。

EVENT_PARAMETER 函数提供事件处理程序的上下文信息。请参见《参考：构件块、表和过程》。

有关使用事件处理的示例，请参见《系统管理指南第一卷》>“使用日程表和事件自动完成任务”>“管理用户帐户和连接”。

日程表和事件深入解析

本节说明了数据库服务器如何处理日程表和事件定义。

数据库服务器如何检查系统事件

事件按在 **CREATE EVENT** 语句中直接指定的*事件类型*进行分类。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“日程表和事件深入解析”>“数据库服务器如何检查系统事件”。

数据库服务器如何检查预定时间

在数据库服务器启动时以及每当调度的事件处理程序完成时，都会进行调度的事件时间的计算。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“日程表和事件深入解析”>“数据库服务器如何检查调度事件”。

事件处理程序是如何执行的

触发事件处理程序时，将建立临时的内部连接，并在其上执行事件处理程序。

该处理程序不在导致该处理程序触发的连接中执行，因此与客户端应用程序交互的语句（例如 **MESSAGE … TO CLIENT**）在事件处理程序中没有任何意义。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“事件处理任务”>“将事件添加到数据库中”。

调度和事件处理任务

本节全面介绍了与自动处理日程表和事件相关的任务。

将日程表或事件添加到数据库中

可以在 Sybase Central 中和使用 SQL 添加日程表和事件。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“事件处理任务”>“将事件添加到数据库中”。

将手动触发的事件添加到数据库中

如果您创建的事件处理程序没有可触发它的日程表或系统事件，则仅当手动触发时它才会执行。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“事件处理任务”>“将手动触发的事件添加到数据库中”。

使用 **ALTER EVENT** 语句更改事件。请参见《参考：语句和选项》。

触发事件处理程序

任何事件处理程序都可以手动触发，以及因日程表或系统事件而执行。您会发现，在开发事件处理程序过程中，手动触发事件是很有用的；对于生产环境中的某些事件，它也是很有用的。

例如，您可能已经有了一个按月调度的销售报表，但有时并未到月末也希望获得销售报表。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“事件处理任务”>“触发事件处理程序”。

有关触发的详细信息，请参见《参考：语句和选项》中的 `TRIGGER EVENT` 语句。

调试事件处理程序

调试是软件开发必不可少的一个环节。在开发过程中可以调试事件处理程序。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“维护数据库”>“使用日程表和事件自动完成任务”>“事件处理任务”>“调试事件处理程序”。

检索有关事件或日程表的信息

Sybase IQ 在系统表 SYSEVENT、SYSEVENTTYPE 和 SYSSCHEDULE 中存储有关事件、系统事件和日程表的信息。

使用 **ALTER EVENT** 语句更改事件时，需要指定事件名称和（可选）日程表名称。使用 **TRIGGER EVENT** 语句触发事件时，需要指定事件名称。

通过查询系统表 SYSEVENT，可以列出事件名称。例如：

```
SELECT event_id, event_name FROM SYSEVENT
```

通过查询系统表 SYSSCHEDULE，可以列出日程表名称。例如：

```
SELECT event_id, sched_name FROM SYSSCHEDULE
```

每个事件都具有唯一的事件 ID。可使用 SYSEVENT 和 SYSSCHEDULE 的 event_id 列将事件与关联的日程表匹配。

使用 JDBC 访问数据

本附录介绍如何使用 JDBC 访问数据。

您既可从客户端应用程序使用 JDBC，也可从数据库内部使用 JDBC。在数据库中加入编程逻辑时，使用 JDBC 的 Java 类也可以起到 SQL 存储过程的作用，而且功能比 SQL 存储过程更强。

JDBC 概述

JDBC 为 Java 应用程序提供了 SQL 接口：如果您要从 Java 访问关系数据，则可利用 JDBC 调用进行此访问。

本附录并不深入讲解 JDBC 数据库接口，而是提供一些简单示例来介绍 JDBC，并阐释如何在服务器内部和外部使用 JDBC。此外，本附录将提供有关服务器端使用 JDBC (在数据库服务器内部运行）的详细信息。

这些示例说明在 Sybase IQ 中使用 JDBC 时的一些独特功能。有关 JDBC 编程的详细信息，请参见讲解 JDBC 编程的书籍。

JDBC 和 Sybase IQ

您可以通过以下方式来配合使用 JDBC 和 Sybase IQ:

- Java 客户端应用程序可对 Sybase IQ 发出 JDBC 调用。该连接通过 Sybase jConnect JDBC 驱动程序或通过 iAnywhere JDBC 驱动程序进行。
  本附录中的术语*客户端应用程序*指用户计算机上运行的应用程序和中层应用程序服务器上运行的逻辑。
- 数据库中安装的服务器 Java 类中的 JDBC 可使用内部 JDBC 驱动程序进行 JDBC 调用，以此来访问和修改数据库中的数据。

本附录重点介绍服务器端的 JDBC。

JDBC 资源

- *必需的软件*
  使用 Sybase jConnect 驱动程序需要 TCP/IP。
  您的计算机上可能已有 Sybase jConnect 驱动程序，具体有无取决于所安装的 Sybase IQ。

另请参见

- jConnect 驱动程序文件 （第 127 页）

选择 JDBC 驱动程序

现为 Sybase IQ 提供了两种 JDBC 驱动程序：

表 7.

驱动程序	定义
jConnect	此驱动程序是 100% 的 Java 驱动程序。它使用 TDS 客户端/服务器协议与 Sybase IQ 进行通信。
iAnywhere JDBC 驱动程序	此驱动程序使用命令序列客户端/服务器协议与 Sybase IQ 进行通信。它的行为与 ODBC、嵌入式 SQL 和 OLE DB 应用程序是一致的。

有关 jConnect 文档，请参见《jConnect for JDBC》。

在选择使用哪个驱动程序时，可能需要考虑下列因素：

- 功能 - 这两个驱动程序都兼容 JDK 2。iAnywhere JDBC 驱动程序提供完全可滚动的游标，jConnect 则不能。
- 纯 Java - jConnect 驱动程序是纯 Java 解决方案。iAnywhere JDBC 驱动程序需要 Sybase IQ 或 Adaptive Server Anywhere ODBC 驱动程序，它不是纯 Java 解决方案。
- 性能 - 在大多数情况下，iAnywhere JDBC 驱动程序所提供的性能要比 jConnect 驱动程序好一些。
- 兼容性 - jConnect 驱动程序使用的 TDS 协议可与 Adaptive Server Enterprise 共享。该驱动程序行为的某些方面受此协议的控制，并被配置为与 Adaptive Server Enterprise 兼容。

这两种驱动程序都可用于 Windows 95/98/Me 和 Windows NT/2000/2003/XP，以及受支持的 UNIX 和 Linux 操作系统。

JDBC 注意事项

运行 Java 应用程序时请注意以下事项：

- 使用 iAnywhere JDBC 驱动程序通过 dbisql Java 连接到 Sybase IQ 12.5 服务器时存在问题。有关详细信息，请参见《系统管理指南第一卷》>“故障排除提示”>“数据截断或数据转换错误”。
- 运行在 Sybase IQ 中的 Java 应用程序比运行在 Sun Java 虚拟机 (JVM) 外的应用程序慢。尽管存在此限制，但 Sybase 仍然建议您通过增加 IQ JVM 用于数据库选项 `JAVA_HEAP_SIZE` 和 `JAVA_NAMESPACE_SIZE` 的可用内存来优化您的应用程序；有关这两个选项的信息，请参见《参考：语句和选项》>“数据库选项”>“JAVA_HEAP_SIZE”。

JDBC 程序结构

JDBC 程序结构具有一系列事件。

JDBC 应用程序中通常会发生以下事件序列：

- 创建连接对象 - 调用 **DriverManager** 类的 **getConnection** 类方法即可创建 **Connection** 对象，并与数据库建立连接。
- 生成语句对象 - **Connection** 对象会生成 **Statement** 对象。
- 传递 SQL 语句 - 在数据库环境内部执行的 SQL 语句传递到 **Statement** 对象。如果该语句是一个查询，则此操作会返回 **ResultSet** 对象。

  ResultSet 对象包含 SQL 语句返回的数据，但一次只显示一行（与游标的工作方式类似）。
- 循环访问结果集中的行 - **ResultSet** 对象的 **next** 方法可执行两种操作：
  - 当前行（通过 **ResultSet** 对象显示的结果集中的行）前进一行。
  - 返回布尔值 (true/false)，指示要前进到的目标行是否真正存在。
- 为每行检索值 - 通过识别 **ResultSet** 对象中每一列的名称或位置来检索每一列的值。可以使用 **getDate** 方法来获取当前行中某列的值。

Java 对象可使用 JDBC 对象与数据库交互作用并获取数据供自己使用，例如，获取数据后加以操纵或将其用在其它查询中。

服务器端 JDBC 功能

JDBC 1.2 是 JDK 1.1 的一部分。JDBC 2.0 是 Java 2 (JDK 1.2) 的一部分。

数据库中的 Java 提供 JDK 1.1 版的子集，因此内部 JDBC 驱动程序支持 JDBC 1.2 版。

内部 JDBC 驱动程序 (`asajdbc`) 使您能从服务器端的 Java 应用程序使用 JDBC 2.0 的某些功能，但未提供完整的 JDBC 2.0 支持。

属于数据库支持中 Java 一部分的 **java.sql** 包中的 JDBC 类处于 1.2 级。属于 JDBC 2.0 一部分的服务器端功能在 **sybase.sql.ASA** 包中实现。要使用 JDBC 2.0 功能，您必须将 JDBC 对象转换为 **sybase.sql.ASA** 包中相应的类，而不是 **java.sql** 包中的类。声明为 **java.sql** 的类仅限于 JDBC 1.2 功能。

sybase.sql.ASA 中的类如下所示：

JDBC 类	Sybase 内部驱动程序类
java.sql.Connection	sybase.sql.ASA.SAConnection
java.sql.Statement	sybase.sql.ASA.SAStatement
java.sql.PreparedStatement	sybase.sql.ASA.SAPreparedStatement
java.sql.CallableStatement	sybase.sql.ASA.SACallableStatement
java.sql.ResultSetMetaData	sybase.sql.ASA.SAResultSetMetaData
java.sql.ResultSet	sybase.sql.SAResultSet
java.sql.DatabaseMetaData	sybase.sql.SADatabaseMetaData

以下函数为预准备语句提供一个 **ResultSetMetaData** 对象，无需 **ResultSet**，也不用执行该语句。此函数不是 JDBC 标准的一部分。

```
ResultSetMetaData sybase.sql.ASA.SAPreparedStatement.describe()
```

JDBC 2.0 限制
以下类是 JDBC 2.0 核心接口的一部分，但 **sybase.sql.ASA** 包中没有这些类：

- java.sql.Blob
- java.sql.Clob
- java.sql.Ref
- java.sql.Struct
- java.sql.Array
- java.sql.Map

sybase.sql.ASA 包中没有以下 JDBC 2.0 核心函数：

sybase.sql.ASA 中的类	缺少的函数
SAConnection	java.util.Map getTypeMap() void setTypeMap(java.util.Map map)
SAPreparedStatement	void setRef(int pidx, java.sql.Ref r) void setBlob(int pidx, java.sql.Blob b) void setClob(int pidx, java.sql.Clob c) void setArray(int pidx, java.sql.Array a)
SACallableStatement	Object getObject(pidx, java.util.Map map) java.sql.Ref getRef(int pidx) java.sql.Blob getBlob(int pidx) java.sql.Clob getClob(int pidx) java.sql.Array getArray(int pidx)

sybase.sql.ASA 中的类	缺少的函数
SAResultSet	Object getObject(int cidx, java.util.Map map)
	java.sql.Ref getRef(int cidx)
	java.sql.Blob getBlob(int cidx)
	java.sql.Clob getClob(int cidx)
	java.sql.Array getArray(int cidx)
	Object getObject(String cName, java.util.Map map)
	java.sql.Ref getRef(String cName)
	java.sql.Blob getBlob(String cName)
	java.sql.Clob getClob(String cName)
	java.sql.Array getArray(String cName)

客户端与服务器端 JDBC 连接的区别

客户端 JDBC 与数据库服务器中的 JDBC 之间的区别是它们与数据库环境建立连接的方式不同。

- 客户端 - 在客户端 JDBC 中，建立连接需要 Sybase jConnect JDBC 驱动程序。将参数传递给 **DriverManager.getConnection** 就建立了连接。从客户端应用程序的角度看，数据库环境是外部应用程序。
- 服务器端 - 在数据库服务器中使用 JDBC 时，连接已经存在。**jdbc:default:connection** 的值传递给 **DriverManager.getConnection**，从而使 JDBC 应用程序能在当前用户连接中工作。这是一个快速、有效且安全的操作，因为客户端应用程序已通过了数据库安全检查建立了连接。用户 ID 和口令一经提供，便不需要再次提供。asajdbc 驱动程序只能连接到当前连接的数据库。

您可以用这样一种方式编写 JDBC 类：使用一个条件语句构造 URL，让 JDBC 类既能在客户端运行，又能在服务器端运行。外部连接需要计算机名称和端口号，而内部连接需要 **jdbc:default:connection**。

建立 JDBC 连接

本节介绍那些用来从 Java 应用程序建立 JDBC 数据库连接的类。

使用 jConnect 从 JDBC 客户端应用程序建立连接

如果希望从 JDBC 应用程序访问数据库系统表（数据库元数据），必须向数据库添加一组 jConnect 系统对象。

如果希望从 JDBC 应用程序访问数据库系统表（数据库元数据），必须向数据库添加一组 jConnect 系统对象。Asajdbc 和 jConnect 共享相同的数据库元数据支持的存储过程。这些过程会缺省安装到所有数据库。**iqinit** 开关 **-i** 会阻止此安装。

-i 开关对 **iqinit** 和 SQL Anywhere 实用程序 **dbinit** 是通用的。有关 **-i** 开关的说明，请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 数据库管理”>“管理数据库”>“数据库管理实用程序”>“初始化实用程序 (dbinit)”。

以下完整的 Java 应用程序是一个命令行应用程序，它连接到正在运行的数据库，在命令行显示一组信息，然后结束。

对于任何 JDBC 应用程序，要使用数据库数据，第一步都必须建立连接。

另请参见

- 从服务器端的 JDBC 类建立连接 （第 117 页）
- Sybase jConnect JDBC 驱动程序 （第 126 页）
- 运行外部连接示例 （第 116 页）
- 分布式应用程序示例 （第 131 页）

外部连接示例代码

用来建立连接的方法的源代码。

源代码位于 Sybase IQ 安装目录下 Windows 上的 `C:\Documents and Settings\All Users\SybaseIQ\samples\SQLAnywhere\JDBC` 目录内或 UNIX 上的 `$SYBASE/IQ-15_3/samples/sqlanywhere/JDBC` 内的文件 `JDBCExamples.java` 中的 **main** 方法和 **ASAConnect** 方法内：

```
// Import the necessary classes
import java.sql.*;              // JDBC
import com.sybase.jdbc.*;       // Sybase jConnect
import java.util.Properties;    // Properties
import sybase.sql.*;            // Sybase utilities
import asademo.*;               // Example classes

private static Connection conn;
public static void main(String args[])  {

    conn = null;
    String machineName;
    if ( args.length != 1 ) {
      machineName = "localhost";
    } else {
      machineName = new String( args[0] );
    }
```

```
    ASAConnect( "dba", "sql", machineName );
    if( conn!=null ) {
        System.out.println( "Connection successful" );
    }else{
        System.out.println( "Connection failed" );
    }

    try{
      serializeVariable();
      serializeColumn();
      serializeColumnCastClass();
    }
    catch( Exception e ) {
      System.out.println( "Error: " + e.getMessage() );
      e.printStackTrace();
    }
  }
}
private static void ASAConnect( String UserID,
                                String Password,
                                String Machinename ) {
    // uses global Connection variable

    String _coninfo = new String( Machinename );

    Properties _props = new Properties();
    _props.put("user", UserID );
    _props.put("password", Password );

    // Load the Sybase Driver
    try {
      Class.forName("com.sybase.jdbc.SybDriver").newInstance();

      StringBuffer temp = new StringBuffer();
      // Use the Sybase jConnect driver...
      temp.append("jdbc:sybase:Tds:");
      // to connect to the supplied machine name...
      temp.append(_coninfo);
      // on the default port number for ASA...
      temp.append(":2638");
      // and connect.
      System.out.println(temp.toString());
     conn = DriverManager.getConnection( temp.toString() , _props );
    }
    catch ( Exception e ) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
```

外部连接示例如何工作

外部连接示例是 Java 命令行应用程序。

导入包

该应用程序需要若干库，这些库在 `JDBCExamples.java` 的最先几行导入：

- **java.sql** 包中包含所有 JDBC 应用程序都需要的 Sun Microsystems JDBC 类。该类位于 Java 子目录下的 `classes.zip` 文件中。
- 使用 jConnect 建立连接的所有应用程序都需要从 **com.sybase.jdbc** 导入的 Sybase jConnect JDBC 驱动程序。该类位于 Java 子目录下的 `jdbcdrv.zip` 文件中。
- 该应用程序使用*属性列表*。处理属性列表需要 **java.util.Properties** 类。该类位于 `Java` 子目录下的 `classes.zip` 文件中。
- **sybase.sql** 包中包含用于序列化的实用程序。该类位于 `Java` 子目录下的 `asajdbc.zip` 文件中。
- **asademo** 包中包含某些示例中用到的类。该类位于 `java` 子目录下的 `asademo.jar` 文件中。

main 方法

每个 Java 应用程序都需要一个具有 **main** 方法的类，该方法是在程序启动时调用的方法。在此简单示例中，**JDBCExamples.main** 是应用程序中的唯一方法。

JDBCExamples.main 方法执行以下任务：

- 如果提供了计算机名称，则用计算机名称处理命令行参数。缺省情况下，计算机名称是适用于个人数据库服务器的 *localhost*。
- 调用 ASAConnect 方法建立连接。
- 执行将数据滚动到命令行的多个方法。

ASAConnect 方法

JDBCExamples.ASAConnect 方法执行以下任务：

- 使用 Sybase jConnect 连接到缺省的运行数据库。
  - **Class.forName** 装载 jConnect。使用 **newInstance** 方法能解决某些浏览器中的问题。
  - **StringBuffer** 语句会根据命令行所提供的文字字符串和已有的计算机名称建立连接字符串。
  - **DriverManager.getConnection** 使用连接字符串建立连接。
- 将控制权还给调用方法。

运行外部连接示例

本节介绍如何运行外部连接示例

1. 在系统命令提示符下，转到 Sybase IQ 安装目录。

2. 转到 `IQ-15_3/java` 子目录
3. 确保 CLASSPATH 环境变量包含当前目录 (.) 以及导入的 zip 文件。例如，在命令提示符下（应全部在一行上输入）：

   ```
   set classpath=..\java\jdbcdrv.zip;.;..\java
   \asajdbc.zip;asademo.jar
   ```

   Java 的默认 zip 文件名为 `classes.zip`。对于任何名为 `classes.zip` 文件中的类，您只需 CLASSPATH 变量中的目录名，无需 zip 文件名本身。对于其它名称的文件中的类，您必须提供 zip 文件名。

   您需要 CLASSPATH 中的当前目录来运行示例。
4. 确保数据库已装载到运行 TCP/IP 的数据库服务器上。您可以在本地计算机上使用以下命令（从 `IQ-15_3/samples/sqlanywhere` 子目录）启动该服务器：

   在 UNIX 平台上： `start_iq .../iqdemo`

   在 Windows 平台上： `start_iq ...\iqdemo`
5. 在命令提示符下输入以下内容，运行该示例：

   ```
   java JDBCExamples
   ```

   如果希望对另一计算机上运行的服务器尝试此命令，必须输入该计算机的正确名称。缺省名称为 `localhost`，它是当前计算机名称的别名。
6. 确认命令提示符下出现人员和产品列表。

   如果连接尝试失败，则显示的是错误消息。请确认是否已执行了所需的全部步骤。请检查 CLASSPATH 是否正确。不正确的 CLASSPATH 将导致无法找到类。

另请参见

- 从服务器端的 JDBC 类建立连接 （第 117 页）
- 使用 jConnect 从 JDBC 客户端应用程序建立连接 （第 114 页）
- Sybase jConnect JDBC 驱动程序 （第 126 页）

从服务器端的 JDBC 类建立连接

JDBC 中的 SQL 语句是使用 **Connection** 对象的 **createStatement** 方法构建的。即使是在服务器内部运行的类，也需要建立连接以创建 **Connection** 对象。

从服务器端的 JDBC 类建立连接比建立外部连接更为直接。因为已连接的用户将执行服务器端的类，而该类只需使用当前连接。

有关 JDBC 连接的说明

- 自动提交行为 - JDBC 规范要求在每个数据修改语句后缺省执行 COMMIT。当前，服务器端 JDBC 行为是提交。您可使用以下语句控制这一行为：

  ```
  conn.setAutoCommit( false ) ;
  ```

其中 **conn** 是当前的连接对象。

- 连接缺省值 - 从服务器端的 JDBC 对 **getConnection("jdbc:default:connection")** 的调用中，只有第一个调用使用缺省值创建新连接。后续调用返回当前连接的包装，所有连接属性均保持不变。如果在初始连接中将 AutoCommit 设为 OFF，同一 Java 代码中，任何后续 **getConnection** 调用所返回的连接中的 AutoCommit 均设置为 OFF。

  您可能希望确保关闭连接会将连接属性重置为缺省值，这样，所获得的后续连接就会采用标准的 JDBC 值。以下代码示范可实现这一点：

  ```
  Connection conn = DriverManager.getConnection("");
  boolean oldAutoCommit = conn.getAutoCommit();
  try {
       // do code here
  }
  finally {
      conn.setAutoCommit( oldAutoCommit );
  }
  ```

  此处的讨论不仅适用于 AutoCommit，也适用于 TransactionIsolation 和 ReadOnly 等其它连接属性。

另请参见

- 使用 jConnect 从 JDBC 客户端应用程序建立连接 （第 114 页）
- Sybase jConnect JDBC 驱动程序 （第 126 页）
- 运行外部连接示例 （第 116 页）

服务器端连接的示例代码

以下是服务器端连接的示例源代码。

该源代码位于 Sybase IQ 安装目录下 `C:\Documents and Settings\All Users\SybaseIQ\samples\SQLAnywhere\JDBC` 目录中的 `JDBCExamples.java` 的 **InternalConnect** 方法中：

```
public static void InternalConnect() {
  try {
      conn = DriverManager.getConnection("jdbc:default:connection");
      System.out.println("Hello World");
    }
    catch ( Exception e ) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

服务器端连接示例如何工作

在此简单示例中，**InternalConnect()** 是应用程序中使用的唯一方法。

该应用程序只需要在 `JDBCExamples.java` 类的第一行导入一个库 (JDBC)。其它库供外部连接使用。名为 **java.sql** 的包中包含 JDBC 类。

InternalConnect() 方法执行以下任务：

1. 使用当前连接与缺省的运行数据库建立连接：
   - **DriverManager.getConnection** 使用 **jdbc:default:connection** 的连接字符串建立连接。
2. 在当前标准输出（服务器窗口）中显示 `Hello World`。**System.out.println** 执行这一显示任务。
3. 如果尝试连接时出现错误，服务器窗口将显示一条错误消息，同时显示出现错误的位置。

   try 和 **catch** 指令提供了处理错误的框架。
4. 该类终止。

运行服务器端连接示例

本节介绍如何运行服务器端连接示例。

1. 如果尚未编译 `JDBCExamples.java` 文件，先编译该文件。如果您使用的是 JDK，可以在命令提示符下从 Sybase IQ 安装目录下的 `C:\Documents and Settings\All Users\SybaseIQ\samples\SQLAnywhere\JDBC` 目录中执行以下任务：

   ```
   javac JDBCExamples.java
   ```
2. 启动使用演示数据库的数据库服务器。您可以在本地计算机上使用以下命令（从 `/ASIQ-12_7/java` 子目录）启动该服务器：

   在 UNIX 平台上： `start_iq .../iqdemo`

   在 Windows 平台上： `start_iq ...\iqdemo`

   在这种情况下，由于未使用 jConnect，因此可以不用 TCP/IP 网络协议。但您必须至少有 8 Mb 可用高速缓存，才能使用数据库中的 Java 类。
3. 在演示数据库中安装类。连接到演示数据库后，即可使用以下命令从 Interactive SQL 进行安装：

   ```
   INSTALL JAVA NEW
   FROM FILE 'C:\Documents and Settings\All Users\SybaseIQ\samples
   \SQLAnywhere\JDBC\JDBCExamples.class'
   ```

   其中 *path* 是安装目录的路径。

   您也可使用 Sybase Central 安装此类。在连接到演示数据库时，打开“Java Objects”文件夹，然后双击“添加类”(Add Class)。然后按向导中的说明进行操作。
4. 您现在可调用此类的 **InternalConnect** 方法，就像调用存储过程一样：

   ```
   CALL JDBCExamples>>InternalConnect()
   ```

   第一次在会话中调用 Java 类时，必须装载内部 Java 虚拟机。这可能需要几秒钟。
5. 确认服务器屏幕上是否出现 `Hello World` 这一消息。

使用 JDBC 访问数据

在数据库中保存某些类或全部类的 Java 应用程序大大优于传统的 SQL 存储过程。不过，在入门阶段，最好并行使用 SQL 存储过程来证实 JDBC 的功能，这样或许会有所帮助。

在下面的示例中，我们编写了向 `Department` 表中插入行的 Java 类。

与其它接口一样，JDBC 中的 SQL 语句要么为*静态*，要么为*动态*。静态 SQL 语句在 Java 应用程序中构造，然后会发送到数据库。数据库服务器会分析该语句，选择执行计划，然后执行该语句。语法分析与选择执行计划统称为*准备*语句。

如果某条类似的语句必须被执行多次（例如，向一个表中插入多次），使用静态 SQL 会带来很大的开销，因为每次都必须执行准备步骤。

相对而言，动态 SQL 语句包含占位符。只需用这些占位符准备一次语句，就可以多次执行语句，省去了额外的准备开销。

本节使用静态 SQL。动态 SQL 将在后面一节讨论。

其它 JDBC 说明

- 访问权限 - 与数据库中的所有 Java 类一样，包含 JDBC 语句的类可由任何用户访问。没有等同于 GRANT EXECUTE 语句（该语句授予执行过程的权限）的语句，也无需用类所有者的名称限定类名称。
- 执行权限 - 执行 Java 类时需要具有执行这些类的连接权限。此行为与存储过程的行为不同，执行存储过程需要所有者权限。

安装 JDBCExamples 类

本节介绍如何安装 JDBCExamples.class 以及为本附录其余部分的 JDBC 示例做准备。

示例代码

本节中的代码段取自安装目录下的完整类 `C:\Documents and Settings\All Users\SybaseIQ\samples\SQLAnywhere\JDBC\JDBCExamples.java`。

安装 JDBCExamples 类

1. 如果尚未安装 JDBCExamples 类，请在演示数据库中安装 `JDBCExamples.class` 文件。
2. 从 Interactive SQL 连接到演示数据库后，请在“SQL 语句”窗格中输入以下命令：

   ```
   INSTALL JAVA NEW
   FROM FILE 'C:\Documents and Settings\All Users\SybaseIQ\samples
   \SQLAnywhere\JDBC\JDBCExamples.class'
   ```

   其中 *path* 是安装目录的路径。

您也可使用 Sybase Central 安装此类。在连接到演示数据库时，打开“Java Objects”文件夹，然后双击“添加 Java 类”(Add Java Class) 或 JAR。然后按向导中的说明进行操作。

使用 JDBC 执行插入、更新和删除

Statement 对象执行静态 SQL 语句。您可使用 **Statement** 对象的 **executeUpdate** 方法执行 INSERT、UPDATE 和 DELETE 等 SQL 语句，这些语句不返回结果集。诸如 CREATE TABLE 的语句及其它数据定义语句也可使用 **executeUpdate** 来执行。

以下代码段说明了 JDBC 如何执行 INSERT 语句。它使用名为 **conn** 的 Connection 对象中所包含的内部连接。使用 JDBC 从外部应用程序插入值的代码需要使用另一不同连接，但其它方面均保持不变。

```
public static void InsertFixed()  {
    // returns current connection
    conn = DriverManager.getConnection("jdbc:default:connection");
    // Disable autocommit
    conn.setAutoCommit( false );

    Statement stmt = conn.createStatement();

    Integer IRows = new Integer( stmt.executeUpdate
       ("INSERT INTO Department (dept_id, dept_name )"
        + "VALUES (201, 'Eastern Sales')"
         ) );
    // Print the number of rows updated
    System.out.println(IRows.toString() + "row inserted" );
  }
```

注意： 此代码段是 **JDBCExamples** 类的 **InsertFixed** 方法的一部分。在 Windows 上，可以使用 `C:\Documents and Settings\All Users\SybaseIQ\samples\SQLAnywhere\JDBC` 中的 `build.bat` 来构建此类。

- **setAutoCommit** 方法关闭自动提交行为，因此只有执行显式 COMMIT 指令，才会提交更改。
- **executeUpdate** 方法返回一个整数，该数字反映受操作影响的行数。在此情况下，成功的 INSERT 会返回值一 (1)。
- 整数返回类型转换为 **Integer** 对象。Integer 类是基本 **int** 数据类型的包装，它提供一些有用的方法，如 **toString()**。
- 整数 **IRows** 转换为要输出的字符串，输出将显示在服务器窗口中。

运行 JDBC Insert 示例

创建非常简单的 JDBC 类。

1. 以用户 ID `dba` 的身份使用 Interactive SQL 连接到演示数据库。
2. 确保已安装 JDBCExamples 类。它是与其它 Java 示例类一起安装的。
3. 按以下方式调用方法：

```
CALL JDBCExamples>>InsertFixed()
```

4. 确认 `department` 表中已添加一行。

```
SELECT *
FROM department
```

不提交 ID 为 201 的行。您可执行 **ROLLBACK** 语句以删除该行。

向 Java 方法传递参数

可以展开 **InsertFixed** 方法，来说明如何将参数传递给 Java 方法。

以下方法使用在调用中传递给该方法的参数作为插入值：

```
public static void InsertArguments(
                  String id, String name)  {
try {
      conn = DriverManager.getConnection(
                     "jdbc:default:connection" );

String sqlStr =  "INSERT INTO Department "
    + " ( dept_id, dept_name )"
    + " VALUES (" +  id +  ", '"  + name + "')";

      // Execute the statement
      Statement stmt = conn.createStatement();
      Integer IRows = new
Integer( stmt.executeUpdate( sqlStr.toString() ) );

      // Print the number of rows updated
      System.out.println(IRows.toString() + " row inserted" );
    }
    catch ( Exception e ) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
```

使用带参数的 Java 方法

- 这两个参数是部门 ID（整数）和部门名称（字符串）。由于两个参数是 SQL 语句字符串的一部分，因此它们在此处都作为字符串传递给方法。
- INSERT 是静态语句，除 SQL 自身外，它不带任何参数。
- 如果提供的参数数目或类型有错误，将出现“`Procedure Not Found`”错误。

1. 如果尚未在演示数据库中安装 `JDBCExamples.class` 文件，请安装该文件。
2. 从 Interactive SQL 连接到演示数据库，然后输入以下命令：

```
call JDBCExamples>>InsertArguments( '203', 'Northern Sales' )
```

3. 检查 Department 表中是否已另外添加了一行：

```
SELECT *
FROM Department
```

4. 回退更改，使数据库保持原样：

```
ROLLBACK
```

使用 JDBC 的查询

Statement 对象执行静态查询和不返回结果集的语句。对于查询，应使用 **Statement** 对象的 **executeQuery** 方法。这将在 **ResultSet** 对象中返回结果集。

以下代码段说明了如何在 JDBC 中处理查询。此代码段将产品的总库存值放在名为 **inventory** 的变量中。产品名称包含在 **String** 变量 **prodname** 中。此示例可用作 **JDBCExamples** 类的 **Query** 方法。

此示例假定已建立了内部或外部连接，该连接包含在名为 **conn** 的 Connection 对象中。它还假定了变量

```
public static void Query () {
int max_price = 0;
    try{
      conn = DriverManager.getConnection(
                  "jdbc:default:connection" );

      // Build the query
      String sqlStr =  "SELECT id, unit_price "
    + "FROM product" ;

      // Execute the statement
      Statement stmt = conn.createStatement();
      ResultSet result = stmt.executeQuery( sqlStr );

      while( result.next() ) {
    int price = result.getInt(2);
    System.out.println( "Price is "  + price );
    if( price > max_price ) {
      max_price = price ;
    }
      }
    }
    catch( Exception e ) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
      return max_price;
  }
```

运行示例
在演示数据库中安装 **JDBCExamples** 类后，可以在 Interactive SQL 中使用以下语句执行该方法：

```
select JDBCExamples>>Query()
```

注释

- 该查询选择名为 **prodname** 的所有产品的数量和单价。这些结果返回到名为 **result** 的 **ResultSet** 对象中。
- 遍历结果集内的每一行。该循环使用 **next** 方法。
- 对于每一行，每列的值都使用 **getInt** 方法检索到一个整数变量中。**ResultSet** 也有用于其它数据类型的方法，例如 **getString**、**getDate** 和 **getBinaryString**。
  getInt 方法的参数是列的索引号，索引号是从 1 开始编排的。
  从 SQL 到 Java 的数据类型转换是按照《Sybase IQ 参考手册》的“SQL 数据类型”一章的“Java/SQL 数据类型转换”中介绍的信息执行的。
- Sybase IQ 支持双向滚动游标。不过，JDBC 只提供 **next** 方法，相当于在结果集中向前滚动。
- 该方法向调用环境返回 **max_price** 的值，Interactive SQL 在“结果”窗格中显示该值。

另请参见

- 分布式应用程序 （第 129 页）
- 插入和检索对象 （第 125 页）

使用预准备语句进行更有效的访问

如果使用 **Statement** 接口，则将分析每条发送到数据库的语句，生成访问计划，然后执行该语句。实际执行语句前的步骤称为*准备*语句。

如果使用 **PreparedStatement** 接口，将会在性能方面获得一些益处。这样您就可以使用占位符来准备语句，然后在执行语句时向占位符赋值。

使用预准备语句在执行多个类似的操作（如插入多行）时特别有用。

有关预准备语句的详细信息，请参见《参考：语句和选项》>“SQL 语句”>“PREPARE 语句 [ESQL]”。

示例

以下示例说明如何使用 **PreparedStatement** 接口，只不过插入单行并未有效地利用预准备语句。

JDBCExamples 类的以下方法执行一个预准备语句：

```
public static void JInsertPrepared(int id, String name) try {
      conn = DriverManager.getConnection(
                    "jdbc:default:connection");

      // Build the INSERT statement
      // ? is a placeholder character
      String sqlStr = "INSERT INTO Department "
    + "( dept_id, dept_name ) "
    + "VALUES ( ? , ? )" ;
```

```
      // Prepare the statement
      PreparedStatement stmt = conn.prepareStatement( sqlStr );

      stmt.setInt(1, id);
      stmt.setString(2, name );
      Integer IRows = new Integer(
                          stmt.executeUpdate() );

      // Print the number of rows updated
      System.out.println(IRows.toString() + " row inserted" );
    }
    catch ( Exception e ) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
```

运行示例
在演示数据库中安装 **JDBCExamples** 类后，可输入以下语句执行此示例：

```
call JDBCExamples>>InsertPrepared(
                              202, 'Eastern Sales' )
```

字符串参数括在单引号中，这是适用于 SQL 的规范。如果从 Java 应用程序调用此方法，则应使用双引号来分隔该字符串。

插入和检索对象

JDBC 作为关系数据库的接口，其设计目的是检索和操纵传统的 SQL 数据类型。

Sybase IQ 也以 Java 类的形式提供了抽象数据类型。使用 JDBC 来访问这些 Java 类的方式取决于您是要插入对象还是要检索对象。

另请参见

- 分布式应用程序 （第 129 页）
- 使用 JDBC 的查询 （第 123 页）

检索对象

您可以通过以下方式检索对象、对象字段及对象方法：

- 访问方法和字段 - Java 方法和字段可以包含在查询的选择列表中。这样，方法或字段会显示为结果集中的列，通过标准的 **ResultSet** 方法（如 **getInt** 或 **getString**）就能访问这些列。
- 检索对象 - 如果查询选择列表中包括了使用 Java 类数据类型的列，您可以使用 **ResultSet getObject** 方法将对象检索到 Java 类中。然后，在 Java 类中访问该对象的方法和字段。Java 对象只能存储在 Catalog 存储中。

插入对象

您可从服务器端的 Java 类使用 JDBC **setObject** 方法将对象插入到数据类型为 Java 类数据类型的列中。

您可使用预准备语句插入对象。例如，以下代码段将 MyJavaClass 类型的对象插入到表 T 的列中：

```
java.sql.PreparedStatement ps =
    conn.prepareStatement("insert T values( ? )" );
ps.setObject( 1, new MyJavaClass() );
ps.executeUpdate();
```

另一方法是建立保存对象的 SQL 变量，然后将该变量插入表中。

Sybase jConnect JDBC 驱动程序

如果希望从客户端应用程序或小程序使用 JDBC，必须将 Sybase jConnect 连接到 Sybase IQ 数据库。

Sybase IQ 可能包含或不含 Sybase jConnect，具体取决于您收到的安装包。您必须要有 jConnect，才能从客户端应用程序使用 JDBC。如果没有 jConnect，您可以使用服务器端的 JDBC。

有关 jConnect 及其用于 Sybase IQ 的完整说明，请参见联机手册或 jConnect Web 站点上提供的 jConnect 文档

注意： 在应用程序中使用 jConnect 之前，需要先输入以下语句来装载驱动程序：

```
Class.forName("com.sybase.jdbc.SybDriver").newInstance();
```

使用 **newInstance** 方法能解决某些浏览器中的问题。

另请参见

- 从服务器端的 JDBC 类建立连接 （第 117 页）
- 使用 jConnect 从 JDBC 客户端应用程序建立连接 （第 114 页）
- 运行外部连接示例 （第 116 页）

Sybase IQ 随附的 jConnect 的版本

Sybase IQ 提供两种版本的 Sybase jConnect JDBC 驱动程序：

- 完整版本 - 如果选择安装 jConnect，则 jConnect 子目录将添加到您的 Sybase IQ 安装中。其中保存了一个有全部 jConnect 文件的目录树。
- zip 文件 - 远程数据访问功能以及 Java 调试工具都使用 jConnect 连接数据库。同时还提供了一个基本 jConnect 类的 zip 文件，以使您即便没有完整开发版本的驱动程序，也可使用 jConnect。

jConnect 驱动程序文件

Sybase jConnect 驱动程序安装在 Sybase IQ 安装目录的 `jConnect` 子目录的一组目录中。如果您没有安装 jConnect，则可以使用安装在 Java 子目录中的 `jdbcdrv.zip` 文件。

jConnect 的类路径设置

为了让应用程序使用 jConnect，jConnect 类在编译时和运行时必须位于 CLASSPATH 环境变量中，这样，Java 编译器和 Java 运行时才能找到必要的文件。

例如，以下命令将 jConnect 驱动程序类路径添加到现有的 CLASSPATH 环境变量中，其中 *path* 是 Sybase IQ 安装目录。

```
set classpath=%classpath%;path\jConnect\classes
```

以下替代命令将 `jdbcdrv.zip` 文件添加到 CLASSPATH。

```
set classpath=%classpath%;path\java\jdbcdrv.zip
```

导入 jConnect 类

jConnect 中的所有类都在 **com.sybase** 包中。客户端应用程序需要访问 **com.sybase.jdbc** 中的类。为了应用程序可以使用 jConnect，您必须在每个源文件的开头导入这些类：

```
import com.sybase.jdbc.*
```

另请参见

- JDBC 概述 （第 109 页）

在数据库中安装 jConnect 系统对象

如果要用 jConnect 访问系统表信息（数据库元数据），必须将 jConnect 系统对象添加到数据库中。

缺省情况下，jConnect 系统对象将添加到使用版本 12.7 创建的任何数据库，以及添加到升级到版本 12.7 的任何数据库。

您可以选择在创建或升级数据库时将 jConnect 对象添加到数据库，也可以选择稍后再添加。

您可以从 Interactive SQL 安装 jConnect 系统对象。

从 Sybase Central 将 jConnect 系统对象添加到版本 12.7 数据库：

1. 以具有 DBA 权限的用户身份从 Sybase Central 连接到数据库。
2. 在 Sybase Central 主查看器的左窗格中，右键单击数据库图标，然后从弹出菜单中选择“重新安装 jConnect 元数据支持”。

从 Interactive SQL 将 jConnect 系统对象添加到版本 12.7 数据库

以具有 DBA 授权的用户身份从 Interactive SQL 连接到数据库，然后在“SQL 语句”窗格中输入以下命令：

```
read path\scripts\jcatalog.sql
```

其中 *path* 是 Sybase IQ 安装目录。

注意： 您也可使用命令提示符向版本 12.7 数据库添加 jConnect 系统对象。在命令提示符下键入：

```
dbisql -c "uid=user;pwd=pwd" path\scripts\jcatalog.sql
```

其中 *user* 和 *pwd* 标识具有 DBA 权限的用户，而 *path* 是 Sybase IQ 安装目录。

提供服务器的 URL

要通过 jConnect 连接到数据库，您需要提供数据库的统一资源定位符 (URL)。

本节中提供了以下示例：

```
StringBuffer temp = new StringBuffer();
// Use the Sybase jConnect driver...
temp.append("jdbc:sybase:Tds:");
// to connect to the supplied machine name...
temp.append(_coninfo);
// on the default port number for ASA...
temp.append(":2638");
// and connect.
System.out.println(temp.toString());
conn = DriverManager.getConnection(temp.toString() , _props );
```

完整的 URL 如下：

```
jdbc:sybase:Tds:machine-name:port-number
```

各组成部分包括：

- jdbc:sybase:Tds - 使用 TDS 应用程序协议的 Sybase jConnect JDBC 驱动程序。
- machine-name - 运行服务器的计算机的 IP 地址或名称。如果要在同一计算机内建立连接，可使用 `localhost`，表示当前计算机。
- port-number - 数据库服务器监听的端口号。分配给 Sybase IQ 的端口号是 2638。如果没有特殊原因，请使用该端口号。

连接字符串长度必须小于 253 个字符。

指定服务器上的数据库

每个 Sybase IQ 服务器一次可以装载一个或多个数据库。上面介绍的 URL 指定服务器，但不会指定数据库。尝试连接的数据库是服务器上的缺省数据库。

您可以通过下列任一方式提供 URL 的扩展形式，由此而指定特定数据库。

使用 ServiceName 参数

```
jdbc:sybase:Tds:machine-name:port-number?ServiceName=DBN
```

使用问号，后面跟随一系列赋值，这是向 URL 提供参数的标准方式。**ServiceName** 是否大写并不重要，但 = 号两边不能有空格。*DBN* 参数是数据库名称。

使用 RemotePWD 参数

另一更常用的方法可用来提供其它连接参数，如数据库名称或数据库文件，这种方法就是使用 **RemotePWD** 字段。您可用 **setRemotePassword()** 方法将 **RemotePWD** 设置成一个 Properties 字段。

下面的示例代码说明了如何使用该字段。

```
sybDrvr = (SybDriver)Class.forName(
     "com.sybase.jdbc2.jdbc.SybDriver" ).newInstance();
props = new Properties();
props.put( "User", "DBA" );
props.put( "Password", "SQL" );
sybDrvr.setRemotePassword(
    null, "dbf=asiqdemo.db", props );
Connection con = DriverManager.getConnection(
    "jdbc:sybase:Tds:localhost", props );
```

利用数据库文件参数 **DBF**，您可使用 jConnect 启动服务器上的数据库。缺省情况下，数据库启动时的设置为 autostop=YES。如果指定 **utility_db** 的 DBF 或 DBN，实用程序数据库将自动启动。

有关实用程序数据库的信息，请参见《系统管理指南第一卷》>“Sybase IQ 系统管理概述”以及“管理用户 ID 和权限”。

TDS 客户端中的特定于 IQ 的连接参数应该在 RemotePWD 中指定。

以下示例显示如何指定特定于 IQ 的连接参数，其中的 myconnection 为 IQ 连接名：

```
p.put(“RemotePWD“, “,,CON=myconnection”);
```

其中的 myconnection 为 IQ 连接名。

分布式应用程序

在*分布式应用程序*中，部分应用程序逻辑在一台计算机上运行，另外一部分在另一台计算机上运行。利用 Sybase IQ，您可创建分布式 Java 应用程序，让部分逻辑在数据库服务器中运行，另一部分逻辑在客户端计算机上运行。

Sybase IQ 能够与外部 Java 客户端交换 Java 对象。

让客户端应用程序从数据库检索 Java 对象是分布式应用程序中的关键任务。本节说明如何完成此任务。

分布式应用程序的功能
`JDBCExamples.java` 中还有其它两种使用分布式计算的方法：

- serializeVariable - 此方法创建由数据库服务器上的 SQL 变量引用的本地 Java 对象，并将其传递回客户端应用程序。
- serializeColumnCastClass - 此方法与 **serializeColumn** 方法类似，但讲的是如何重新构造子类。被查询的列（**product** 表的 **JProd**）数据类型为 **asademo.Product**。某些行为 **asademo.Hat**，它是 Product 类的子类。相应的类会在客户端得以重新构造。

分布式应用程序的要求
建立分布式应用程序涉及到两个任务。

- 服务器中运行的所有类都必须实现 Serializable 接口。这非常简单。
- 客户端应用程序必须导入类，这样才能在客户端重新构造对象。

以下各节对这些任务逐一进行说明。

另请参见
- 插入和检索对象 （第 125 页）
- 使用 JDBC 的查询 （第 123 页）

Serializable 接口

对象以*序列化*的形式从服务器传递到客户端应用程序。为了让对象传送到客户端应用程序，对象必须实现 Serializable 接口。好在该任务非常简单。

Serializable 接口不包含任何方法和变量。序列化对象会将该对象转换为字节流，这样它就可以保存到磁盘或发送到其它 Java 应用程序，然后可以在磁盘或该 Java 应用程序中重新构建它或对它进行*反序列化*。

数据库服务器中的序列化 Java 对象被发送到客户端应用程序并反序列化后，各方面仍与其原始状态完全相同。不过，对象中的某些变量要么不需要被序列化，要么由于安全原因而不应被序列化。应使用关键字 **transient** 声明那些变量，如下面的变量声明中所示。

```
transient String password;
```

带有该变量的对象被反序列化后，总是包含其缺省值 NULL。

在类中添加 **writeObject()** 和 **readObject()** 方法即可完成自定义序列化。

有关序列化的详细信息，请参见 Sun Microsystems 的 Java Development Kit (JDK)。

实现 Serializable 接口

实现 Serializable 接口就等于直接声明类可以被序列化。
将 **implements java.io.Serializable** 字样添加到类定义中。

例如，位于 `$SADIR/samples/asa/java/asademo` (UNIX) 或 `%SADIR%\samples\asa\java\asademo` (Windows) 子目录中的 Product 类借助于以下声明实现 Serializable 接口：

```
public class Product implements java.io.Serializable
```

在客户端导入类

在客户端，检索对象的任何类都必须具备对相应类定义的访问权限才能使用对象。

如果要使用 **Product** 类（它是 **asademo** 包的一部分），应用程序中必须包括下面一行：

```
import asademo.*
```

CLASSPATH 中必须包括 `asademo.jar` 文件，这样才能找到此包。

分布式应用程序示例

`JDBCExamples.java` 类包含说明分布式 Java 计算的三种方法。这些方法都从 **main** 方法调用。

下面是 JDBCExamples 类中的 **getObjectColumn** 方法。

```
private static void getObjectColumn() throws Exception {
// Return a result set from a column containing
// Java objects
    asademo.ContactInfo ci;
    String name;
    String sComment ;

    if ( conn != null ) {
      Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery(
           "SELECT JContactInfo FROM jdba.contact"
    );
    while ( rs.next() ) {
    ci = ( asademo.ContactInfo )rs.getObject(1);
  System.out.println( "\n\tStreet: " + ci.street +
      "City: " + ci.city +
      "\n\tState: " + ci.state +
      "Phone: " + ci.phone +
      "\n" );
      }
    }
  }
```

getObject 方法的使用方式与在内部 Java 中的使用方式相同。

另请参见

- 使用 jConnect 从 JDBC 客户端应用程序建立连接 （第 114 页）

调试数据库中的逻辑

本附录介绍了如何使用 Sybase 调试工具协助开发 SQL 存储过程和事件处理程序以及 Java 存储过程。

调试数据库简介

您可以在开发期间使用调试工具。

您可以使用以下对象：

- SQL 存储过程、事件处理程序和用户定义的函数。
- 数据库中的 Java 存储过程。

调试工具的功能

可以在开发 SQL 存储过程、触发器、事件处理程序和用户定义的函数期间使用调试工具。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“调试过程、函数、触发器和事件”>“SQL Anywhere 调试工具简介”。

使用调试工具的要求

要使用调试工具，您需要：

- 权限 - 您必须具有 DBA 权限或在 SA_DEBUG 组中已被授予权限。创建所有数据库时，都会将该组自动添加到数据库中。
- Java 类的源代码 - 必须为调试工具提供应用程序的源代码。对于 Java 类，源代码保存在硬盘上的目录中。对于存储过程，源代码保存在数据库中。
- 编译选项 - 要调试 Java 类，编译这些类时必须使其包含调试信息。例如，如果使用 Sun Microsystems JDK 编译器 `javac.exe`，就必须使用 `-g` 命令行选项对 Java 类进行编译。

注意： Sybase IQ 演示数据库是 iqdemo.db。

教程 1：调试工具快速入门

这些教程说明如何启动调试工具、连接到数据库，以及如何调试 Java 类。

第 1 课：连接到数据库并启动调试工具

本教程讲解如何启动调试工具、连接到数据库及使用连接以进行调试。它使用的是 Sybase IQ 演示数据库。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“调试过程、函数、触发器和事件”>“教程：调试工具快速入门”>“第 1 课：连接到数据库并启动调试工具”。

教程 2：调试存储过程

本教程通过一个示例会话讲解如何调试存储过程。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“调试过程、函数、触发器和事件”>“教程：调试工具快速入门”>“第 2 课：调试存储过程”。

教程 3：调试 Java 类

在本教程中，您将从 Interactive SQL (dbisql) 中调用 **JDBCExamples.Query()**，在调试工具中中断执行，然后跟踪执行此方法的源代码。

JDBCExamples.Query() 方法对演示数据库执行以下查询：

```
SELECT ID, UnitPrice
FROM Products
```

然后循环检查结果集中的所有行，并返回单价最高的那一行。

您必须使用 `javac -g` 选项编译类才能对其进行调试。示例类已按调试要求进行编译。

注意： 要使用 Java 示例，必须在演示数据库中安装 Java 示例类。

演示数据库 Java 示例类

如果要运行 Java 示例，则需要在演示数据库中安装 Java 示例类。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - 编程”>“SQL Anywhere 数据访问 API”>“SQL Anywhere JDBC 驱动程序”>“使用 JDBC 访问数据”>“准备示例”。

在调试工具中显示 Java 源代码

调试工具在一组位置中查找扩展名为 `.java` 的源代码文件。

“Java 源代码路径”(Java Source Code Path) 窗口包含调试工具在其中查找 Java 源代码的目录列表。将应用查找软件包的 Java 规则。调试工具也搜索源代码的当前 CLASSPATH。

1. 在左侧文件夹视图中选择 Sybase IQ 15。
2. 在 Sybase Central 中，选择“模式”>“调试”。
3. 当系统提示选择要调试的用户时，指定 * 以表示所有用户，然后单击“确定”。
4. 从调试工具界面，选择“调试”>“设置 Java 源代码路径”。
5. 输入 Sybase IQ 安装目录的 `java` 子目录的路径。例如，如果您将 Sybase IQ 安装在 `%IQDIR15%` 中，请输入：

   ```
   %IQDIR15%\java
   ```
6. 单击“浏览文件夹”(Browse Folder) 从调试程序在其中查找 Java 源代码的文件夹或各个文件的列表中进行选择。
7. 单击“浏览文件”(Browse File) 来定位要添加到列表中的文件。
8. 单击“确定”，然后关闭该窗口。

设置断点

可以在 Query() 方法的开头设置断点。调用该方法时，将在断点处停止执行。

1. 在“源代码”窗口中，向下滚动，直到在类的结尾附近看到 **Query()** 方法的开始部分，其开头为：

   ```
   public static int Query() {
   ```
2. 单击该方法第一行左侧的绿色指示符，直至其变为红色。该方法的第一行为：

   ```
   int max_price = 0;
   ```

   重复单击该指示符将切换其状态。设置断点后，Java 类不需要重新编译。

运行方法

您可以从 Interactive SQL (dbisql) 调用 **Query()** 方法，并看到其执行在断点处中断。

1. 启动 Interactive SQL。使用 ID `DBA` 和口令 `sql` 连接到演示数据库。

   连接将出现在调试工具的“连接”窗口列表中。
2. 要调用方法，请在 Interactive SQL 中输入以下命令：

   ```
   SELECT JDBCExamples.Query()
   ```

该查询不会完成。在调试工具中该查询将在执行到断点处停止。在 Interactive SQL 中，“停止”按钮处于活动状态。在调试工具的“源”窗口中，红色箭头指示当前行。

现在可以在调试工具中单步执行源代码并完成调试活动。

单步执行源代码

上一节结束后，调试工具应在 **JDBCExamples.Query()** 方法的第一条语句处停止执行该方法：

1. 选择“调试”>“单步执行跳过过程”，或按 F10 单步执行到当前方法中的下一行。请尝试执行两三次。
2. 使用鼠标单击以下行的结尾，然后选择“调试”>“运行至游标”，或按 CTRL + F10 以运行到该行并中断：
3. 选择以下行（第 292 行）并按 F9 在该行设置断点：

   ```
   return max_price;
   ```

   左侧列中将出现一个星号以标记该断点。按 F5 可执行到该断点。
4. 尝试使用不同的方法单步执行代码。按 F5 完成执行。

   完成执行后，Interactive SQL 数据窗口将显示值 24。
5. 要移动到下一个断点，请添加 F5。

   完成执行后，Interactive SQL 数据窗口将显示值 24。

“**Run**”菜单上显示了用于单步执行源代码的完整选项集。有关详细信息，可查看调试工具的联机帮助。

检查和修改变量

可以在调试工具中检查局部变量（在方法中声明）和类静态变量的值。

还可以在调试工具的“调试程序”(Debugger) 窗口中显示类级变量（静态变量），并检查它们的值。有关详细信息，请参见调试工具的联机帮助。

在单步执行代码时，可以检查方法中局部变量的值，以更好地了解执行情况。

注意： 要使用 Java 示例，必须在演示数据库中安装 Java 示例类。

1. 在 **JDBCExamples.Query** 方法的第一行设置断点。该行如下所示：

   ```
   int max_price = 0
   ```
2. 在 Interactive SQL 中，再次执行该方法：

   ```
   SELECT JDBCExamples.Query()
   ```

   查询只执行到断点处。
3. 按 F7 以进入下一行。现在已声明了 **max_price** 变量，并已将其初始化为零。

4. 如果未显示“局部”窗口，请选择“窗口”>“局部”来显示它。

   “局部”窗口显示存在一些局部变量。**max_price** 的值为零。所有其它变量列出为 `variable not in scope`（不在范围中的变量），这表示它们还未初始化。
5. 在“局部”窗口中，双击 **max_price** 的“值”列条目，然后将 **max_price** 的值更改为 45。

   值 45 高于其它任何价格。现在，该查询返回的最高价格不再是 24，而是 45。
6. 在“源”(Source) 窗口中，反复按 F7 以逐步调试代码。变量的值出现在“局部”(Locals) 窗口中。单步执行，直到 **stmt** 和 **result** 变量具有值。
7. 单击 **result** 对象旁边的图标，将其展开，或者将游标放在该行上，然后按 Enter。这将显示该对象中的字段的值。
8. 在体验过检查和修改变量后，按 F5 完成该查询的执行并结束本教程。

断点

断点控制何时调试工具中断源代码的执行。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“调试过程、函数、触发器和事件”>“使用断点”。

查看和编辑变量行为

使用调试工具可以在单步执行代码时查看和编辑变量的行为。

调试工具提供了“调试工具详细信息”窗格，该窗格显示存储过程使用的各种不同变量。当 Sybase Central 在调试模式下运行时，“调试工具详细信息”窗格显示在 Sybase Central 底部。

请参见《SQL Anywhere 11.0.1》>“SQL Anywhere Server - SQL 用法”>“存储过程和触发器”>“调试过程、函数、触发器和事件”>“使用变量”。

编写调试工具脚本

调试工具允许您以 Java 编程语言编写脚本。脚本为扩展 **sybase.asa.procdebug.DebugScript** 类的 Java 类。

调试工具运行脚本时，将装载该类并调用其 **run** 方法。**run** 方法的第一个参数为指向类实例的指针。通过此接口可以与调试工具交互并对其进行控制。

可以使用如下命令编译脚本：

```
javac -classpath %asany%/procdebug/ProcDebug.jar;%classpath%
myScript.Java.
```

sybase.asa.procdebug.DebugScript 类

DebugScript 类如下所示：

```
// All debug scripts must inherit from this class

package sybase.asa.procdebug;

abstract public class DebugScript
{
    abstract public void run( IDebugAPI db, String args[] );
    /*
        The run method is called by the debugger
        - args will contain command line arguments
    */

    public void OnEvent( int event ) throws DebugError {}
    /*
    - Override the following methods to process debug events
    - NOTE: this method will not be called unless you call
        DebugAPI.AddEventHandler( this );
    */

}
```

sybase.asa.procdebug.IDebugAPI 接口

IDebugAPI 接口如下所示：

```
package sybase.asa.procdebug;
import java.util.*;
public interface IDebugAPI

{
    // Simulate Menu Items

    IDebugWindow MenuOpenSourceWindow() throws DebugError;
    IDebugWindow MenuOpenCallsWindow() throws DebugError;
    IDebugWindow MenuOpenClassesWindow() throws DebugError;
    IDebugWindow MenuOpenClassListWindow() throws DebugError;
    IDebugWindow MenuOpenMethodsWindow() throws DebugError;
    IDebugWindow MenuOpenStaticsWindow() throws DebugError;
    IDebugWindow MenuOpenCatchWindow() throws DebugError;
    IDebugWindow MenuOpenProcWindow() throws DebugError;
    IDebugWindow MenuOpenOutputWindow() throws DebugError;
    IDebugWindow MenuOpenBreakWindow() throws DebugError;
    IDebugWindow MenuOpenLocalsWindow() throws DebugError;
    IDebugWindow MenuOpenInspectWindow() throws DebugError;
    IDebugWindow MenuOpenRowVarWindow() throws DebugError;
    IDebugWindow MenuOpenQueryWindow() throws DebugError;
    IDebugWindow MenuOpenEvaluateWindow() throws DebugError;
    IDebugWindow MenuOpenGlobalsWindow() throws DebugError;
    IDebugWindow MenuOpenConnectionWindow() throws DebugError;
    IDebugWindow MenuOpenThreadsWindow() throws DebugError;
    IDebugWindow GetWindow( String name ) throws DebugError;
```

```
    void MenuRunRestart() throws DebugError;
    void MenuRunHome() throws DebugError;
    void MenuRunGo() throws DebugError;
    void MenuRunToCursor() throws DebugError;
    void MenuRunInterrupt() throws DebugError;
    void MenuRunOver() throws DebugError;
    void MenuRunInto() throws DebugError;
    void MenuRunIntoSpecial() throws DebugError;
    void MenuRunOut() throws DebugError;
    void MenuStackUp() throws DebugError;
    void MenuStackDown() throws DebugError;
    void MenuStackBottom() throws DebugError;
    void MenuFileExit() throws DebugError;
    void MenuFileOpen( String name ) throws DebugError;
    void MenuFileAddSourcePath( String what ) throws DebugError;
    void MenuSettingsLoadState( String file ) throws DebugError;
    void MenuSettingsSaveState( String file ) throws DebugError;
    void MenuWindowTile() throws DebugError;
    void MenuWindowCascade() throws DebugError;
    void MenuWindowRefresh() throws DebugError;
    void MenuHelpWindow() throws DebugError;
    void MenuHelpContents() throws DebugError;
    void MenuHelpIndex() throws DebugError;
    void MenuHelpAbout() throws DebugError;
    void MenuBreakAtCursor() throws DebugError;
    void MenuBreakClearAll() throws DebugError;
    void MenuBreakEnableAll() throws DebugError;
    void MenuBreakDisableAll() throws DebugError;
    void MenuSearchFind( IDebugWindow w, String what ) throws
DebugError;
    void MenuSearchNext( IDebugWindow w ) throws DebugError;
    void MenuSearchPrev( IDebugWindow w ) throws DebugError;
    void MenuConnectionLogin() throws DebugError;
    void MenuConnectionReleaseSelected() throws DebugError;

    // output window
    void OutputClear();
    void OutputLine( String line );
    void OutputLineNoUpdate( String line );
    void OutputUpdate();

    // Java source search path

    void SetSourcePath( String path ) throws DebugError;
    String GetSourcePath() throws DebugError;

    // Catch java exceptions
    Vector GetCatching();
    void Catch( boolean on, String name ) throws DebugError;

    // Database connections
    int  ConnectionCount();
    void ConnectionRelease( int index );
    void ConnectionAttach( int index );
    String ConnectionName( int index );
```

```
    void ConnectionSelect( int index );

    // Login to database
    boolean LoggedIn();
    void Login( String url, String userId, String password, String
userToDebug ) throws DebugError;
    void Logout();

    // Simulate keyboard/mouse actions
    void DeleteItemAt( IDebugWindow w, int row ) throws DebugError;
    void DoubleClickOn( IDebugWindow w, int row ) throws DebugError;

    // Breakpoints
    Object BreakSet( String where ) throws DebugError;
    void  BreakClear( Object b ) throws DebugError;
    void BreakEnable( Object b, boolean enabled ) throws DebugError;
    void BreakSetCount( Object b, int count ) throws DebugError;
    int  BreakGetCount( Object b ) throws DebugError;
    void BreakSetCondition( Object b, String condition ) throws
DebugError;
    String BreakGetCondition( Object b ) throws DebugError;
    Vector GetBreaks() throws DebugError;

    // Scripting
    void RunScript( String args[] ) throws DebugError;
    void AddEventHandler( DebugScript s );
    void RemoveEventHandler( DebugScript s );

    // Miscellaneous
    void EvalRun( String expr ) throws DebugError;
    void QueryRun( String query ) throws DebugError;
    void QueryMoreRows() throws DebugError;
    Vector GetClassNames();
    Vector GetProcedureNames();
    Vector WindowContents( IDebugWindow window ) throws DebugError;
    boolean AtBreak();
    boolean IsRunning();
    boolean AtStackTop();
    boolean AtStackBottom();
    void SetStatusText( String msg );
    String GetStatusText();
    void WaitCursor();
    void OldCursor();
    void Error( Exception x );
    void Error( String msg );
    void Warning( String msg );
    String Ask( String title );
    boolean MenuIsChecked( String cmd );
    void MenuSetChecked( String cmd, boolean on );
    void AddInspectItem( String s ) throws DebugError;

    // Constants for DebugScript.OnEvent parameter
    public static final int EventBreak = 0;
    public static final int EventTerminate = 1;
    public static final int EventStep = 2;
    public static final int EventInterrupt = 3;
```

```
    public static final int EventException = 4;
    public static final int EventConnect = 5;
};
```

sybase.asa.procdebug.IDebugWindow 接口

IDebugWindow 接口如下所示：

```
// this interface represents a debugger window
package sybase.asa.procdebug;
public interface IDebugWindow
{
    public int GetSelected();
    /*
        get the currently selected row, or -1 if no selection
    */

    public boolean SetSelected( int i );
    /*
       set the currently selected row.  Ignored if i < 0 or i > #rows
    */

    public String StringAt( int row );
    /*
       get the String representation of the Nth row of the window.
Returns null if row > # rows
    */

    public java.awt.Rectangle GetPosition();
    public void SetPosition( java.awt.Rectangle r );
    /*
        get/set the windows position within the frame
    */

    public void Close();
    /*
        Close (destroy) the window
    */
}
```

索引

符号

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z