UltraLiteJ

2009 年 2 月

11.0.1 版

版权和商标

版权所有 © 2009 iAnywhere Solutions, Inc. 部分版权所有 © 2009 Sybase, Inc. 保留所有权利。

本文档按原样提供，并不做任何形式的担保或承担任何责任（除非在您与 iAnywhere 达成的书面协议中另行规定）。

对本文档（全部或部分）的使用、打印、复制和分发须符合下列条件：1) 必须在整个或部分文档的所有副本中保留此声明和所有其它所有权声明，2) 不得修改本文档，3) 不得以任何形式表明您或 iAnywhere 之外的任何人是本文档的作者或提供者。

iAnywhere®、Sybase® 以及在 http://www.sybase.com/detail?id=1011207 上所列出的商标均为 Sybase, Inc. 或其子公司的商标。® 表示在美国注册。

文中提及的所有其它公司和产品名可能是与其相关的各个公司的商标。

目录

关于本书 vii

关于 SQL Anywhere 文档 viii

使用 UltraLiteJ 1

UltraLiteJ 简介 3
UltraLiteJ 概述 4
UltraLiteJ 功能 5
UltraLiteJ 功能限制 7
UltraLiteJ 数据库存储区 8
数据同步 10
开发 UltraLiteJ 应用程序 11
UltraLiteJ 开发简介 12
访问 UltraLiteJ 数据库存储 13
执行模式操作 16
使用 SQL 访问和操作数据 18
对数据进行加密和模糊处理 23
与 MobiLink 同步 25
部署 UltraLiteJ 应用程序 30
代码示例 31
教程：构建 BlackBerry 应用程序 65
UltraLiteJ 开发简介 66
第 1 部分：创建用于 BlackBerry 的 UltraLiteJ 应用程序 67
第 2 部分：向 BlackBerry 应用程序添加同步 76
教程的代码列表 80

UltraLiteJ 参考 87

UltraLiteJ API 参考 89
CollectionOfValueReaders 接口 91
CollectionOfValueWriters 接口 98
ColumnSchema 接口 104
ConfigFile 接口 109

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

ConfigNonPersistent 接口 110
ConfigObjectStore 接口（仅限 J2ME BlackBerry） 111
ConfigPersistent 接口 112
ConfigRecordStore 接口（仅限 J2ME） 119
Configuration 接口 120
Connection 接口 122
DatabaseInfo 接口 144
DatabaseManager 类 147
DecimalNumber 接口 153
Domain 接口 156
EncryptionControl 接口 169
ForeignKeySchema 接口 171
IndexSchema 接口 173
PreparedStatement 接口 176
ResultSet 接口 180
ResultSetMetadata 接口 183
SISListener 接口（仅限 J2ME BlackBerry） 184
SISRequestHandler 接口（仅限 J2ME BlackBerry） 185
SQLCode 接口 186
StreamHTTPParms 接口 205
StreamHTTPSParms 接口 209
SyncObserver 接口 213
SyncObserver.States 接口 215
SyncParms 类 219
SyncResult 类 234
SyncResult.AuthStatusCode 接口 238
TableSchema 接口 240
ULjException 类 248
Value 接口 252
ValueReader 接口 256
ValueWriter 接口 260
UltraLiteJ 系统表 265
systable 系统表 266
syscolumn 系统表 267
sysindex 系统表 268

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

sysindexcolumn 系统表 269
sysinternal 系统表 270
syspublications 系统表 271
sysarticles 系统表 272
sysforeignkey 系统表 273
sysfkcol 系统表 274
UltraLiteJ 实用程序 275
J2SE 的实用程序 276
J2ME 的实用程序（用于 BlackBerry 智能手机） 280

术语表 283

术语表 285

索引 313

关于本书

主题

本书介绍 UltraLiteJ。利用 UltraLiteJ，可以在支持 Java 的环境中开发和部署数据库应用程序。UltraLiteJ 支持 BlackBerry 智能手机和 Java SE 环境。UltraLiteJ 基于 iAnywhere UltraLite 数据库产品。

目标读者

本手册适用于想要使用基于 Java 的数据库来进行数据存储和同步的应用程序开发人员。

关于 SQL Anywhere 文档

完整的 SQL Anywhere 文档以四种形式提供，但所包含信息均相同。

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

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

  管理工具使用同一联机文档来实现帮助功能。

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

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

  使用 DocCommentXchange 可以执行以下任务：

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

  访问 http://dcx.sybase.com。

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

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

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

关于文档集中的手册

SQL Anywhere 文档由以下手册组成：

- **SQL Anywhere 11 - 简介**　本手册介绍 SQL Anywhere 11，一个提供数据管理和数据交换技术的综合数据包，通过它可以为服务器环境、台式机环境、移动环境以及远程办公环境快速开发由数据库驱动的应用程序。

- **SQL Anywhere 11 - 更改和升级**　本手册介绍 SQL Anywhere 11 以及该软件以前版本中的新功能。

- **SQL Anywhere 服务器 - 数据库管理**　本手册介绍如何运行、管理及配置 SQL Anywhere 数据库。它介绍了数据库连接、数据库服务器、数据库文件、备份过程、安全性、高可用性、使用复制服务器进行复制以及管理实用程序和选项。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

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

文档约定

本节列出了本文档中使用的约定。

操作系统

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

为了简化对操作系统的提及，本文档按如下方式对支持的操作系统进行分组：

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

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

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

目录和文件名

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

下面是文档编写中用于简化目录和文件名的约定：

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

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

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

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

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

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

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

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

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

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

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

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

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

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

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

命令提示符和命令 shell 语法

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

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

下面是命令行语法的一些示例，不同的平台可能会有不同的形式：

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

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

  如果括号导致出现语法问题，用大括号替代：

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

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

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

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

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

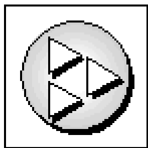
  在许多 shell 中，密钥的值为 `my "secret" key`。

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

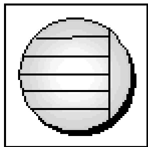
图标

本文档中使用了下列图标。

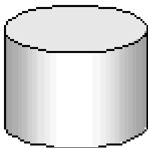
- 客户端应用程序。



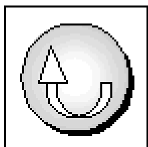
- 数据库服务器，如 Sybase SQL Anywhere。



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



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



- 编程接口。

联系文档小组

我们欢迎您就本帮助文档提出意见、建议和反馈信息。

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

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

DocCommentXchange

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

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

访问 http://dcx.sybase.com。

查找详细信息并请求技术支持

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

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

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

新闻组位于 *forums.sybase.com* 新闻服务器上。

这些新闻组包括：

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

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

新闻组免责声明

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

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

使用 UltraLiteJ

UltraLiteJ 简介 3
开发 UltraLiteJ 应用程序 11
教程：构建 BlackBerry 应用程序 65

UltraLiteJ 简介

目录

UltraLiteJ 概述 4
UltraLiteJ 功能 5
UltraLiteJ 功能限制 7
UltraLiteJ 数据库存储区 8
数据同步 10

UltraLiteJ 概述

UltraLiteJ 是 UltraLite 中基于 Java 的一部分，专为 Java ME 和 SE 平台以及 BlackBerry 智能手机而设计。通过 UltraLiteJ，可以将事务、主键、外键、索引以及关系数据库的其它功能引入 BlackBerry 智能手机。UltraLiteJ 提供了内置的更改跟踪和同步功能，这些功能使您可以在 BlackBerry 应用程序中构建数据同步。将 UltraLiteJ 与 MobiLink 服务器一起使用时，可以将 Oracle、SQL Server、DB2、Sybase ASE 以及 SQL Anywhere 数据库扩展到移动设备。

UltraLiteJ 包括关系数据库的许多典型特性，其中有在表中存储数据、使用主键以及采用事务性数据库存储。

如果要重新分发 UltraLiteJ，可以向应用程序添加 Java 档案（Java Archive，简称 JAR）文件。Java ME、Java SE（1.5 版或更高版本）以及运行 OS 4.1 版及更高版本的 BlackBerry 智能手机上都支持 UltraLiteJ。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

UltraLiteJ 功能

数据库存储

UltraLiteJ 支持以内存（非持久性）和设备（持久性）存储的数据库。对于 Java SE，文件系统是指设备存储。对于 BlackBerry 智能手机，文件系统是指 BlackBerry 对象存储。

事务

事务是两个提交或两个回退之间的一组操作。对于持久性数据库存储，提交操作可使上次提交或回退操作以后的所有更改成为永久性更改。回退操作能将数据库返回到调用上一个提交操作时其所处的状态。

UltraLiteJ 中每个事务和行级操作都是原子操作。涉及多列的插入操作要么是将数据插入到所有列，要么是不插入到任何列。

检查点和恢复

UltraLiteJ 提供了自动和手工两种检查点功能。设置为自动时，COMMIT 语句会使表行和索引进行更新；设置为手工时，将使用 Connection 接口的检查点方法调用检查点。

并发和锁定

UltraLiteJ 使用隔离级别 0（读取未提交数据）来提供最大并发级别。

- **锁定**　两个不同的连接无法同时修改同一行。如果两个连接尝试对同一个行进行操作，则在一个连接结束之前另一个连接将一直被阻塞。
- **可见性**　一个连接对数据库的操作会立即对其它连接可见。

高速缓存管理

持久存储区以页为基础，UltraLiteJ 在高速缓存中对页进行操作。UltraLiteJ 在高速缓存中维护一组工作页，并使用先入先出 (FIFO) 方案对其进行管理。当前正在使用的页面在高速缓存中锁定，以免被交换出高速缓存。

可以配置 UltraLiteJ 高速缓存的大小。

UltraLiteJ 可以使用延缓加载索引和行页来改善持久性数据库的启动。索引和行页只有在首次被应用程序访问时才装载。

加密

使用 Configuration 对象的 setEncryption 方法设置加密，该方法利用 EncryptionControl 加密和解密页。您必须提供自己的加密控制。

内置更改跟踪

作为一种 MobiLink 同步客户端，UltraLiteJ 包含一个内置的、基于事务日志的更改跟踪系统，从而能够同步数据库更改。

HTTP 和 HTTPS 通信

可以使用 HTTP 或 HTTPS 网络协议执行数据同步。HTTPS 同步提供了对 MobiLink 服务器的安全加密。

同步发布

为了更高效地使用网络资源，UltraLiteJ 提供了一个发布模型，该模型允许您同步从数据库中选择的表。

字符集和归类

UltraLiteJ 使用 Unicode（在数据库中编码为 UTF-8）。而所使用的归类是 Java 的缺省排序顺序，等同于 SQL Anywhere 支持的 UTF8BIN 归类。在与 MobiLink 服务器同步期间，UltraLiteJ 会通知 MobiLink 它使用 UTF8 字符集和归类。

UltraLiteJ 功能限制

常规限制

下表列出了对于 UltraLiteJ 数据库的常规限制：

功能	限制
Blob 大小	最大 2^{24} 字节。
BlackBerry SD 卡	不支持。
数据类型	请参见“Domain 接口”一节第 156 页。
数据库访问	每次只能有一个应用程序可访问某个数据库。不支持同时访问。
数据库页面大小	最小 256 字节，最大 32 KB。
行大小	行容量（经过可行的压缩）不得超过数据库页面大小。
每个数据库中的表	最大 32 K。

UltraLiteJ 数据库存储区

支持的数据库存储区

以下是 UltraLiteJ 所支持的数据库存储类型的列表：

数据库存储类型	平台支持
文件系统存储	Java SE
RIM 对象存储 (RIM1x)	Java ME
记录存储	Java ME
非持久存储	所有 Java 平台

BlackBerry 对象存储的限制

在 BlackBerry 智能手机上，数据库存储区的大小受可用对象句柄数量的限制。可用的对象句柄数量由闪存大小确定：

闪存	持久句柄	句柄
8 MB	12000	24000
16 MB	27000	56000
32 MB	65000	132000

UltraLiteJ 需要使用对象句柄以在数据库中存储值。例如，列数为 10 而索引数为 2 的表行至少需要 12 个对象句柄。

要使数据库存储区更大，UltraLiteJ 需要为每个数据库页使用持久对象句柄。

持久存储区配置和恢复

创建数据库时，使用 Configuration 对象为应用程序选择以下一种持久性形式。

- **非持久性**　创建 NonPersist 对象可以配置仅存在于内存中的数据库存储区。数据库在应用程序启动时创建，并在应用程序运行时使用，然后在应用程序关闭时也随之被放弃。应用程序关闭时，将删除非持久存储区中包含的所有数据。
- **最终写入持久性**　使用持久性配置对象的 setWriteAtEnd 方法启用最终写入持久性后，只有在释放连接后才向数据库写入数据。虽然此形式的持久性会改善事务的整体速度，但如果应用程序异常终止，数据会丢失。
- **影子分页持久性**　影子分页是最强大的持久性形式。它可通过持久性配置对象的 setShadowPaging 方法启用。如果在启动时就启用该持久性形式，它会将数据库恢复到上一个检查点状态，即使应用程序意外中止。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

- **简单分页持久性**　未启用影子分页时，会使用简单分页持久性。所有数据都会写入从中读取数据的同一页。从更新开始后到完成前的时间段内，将数据库视为处于损坏状态。在启动时可以检测到损坏的数据库，但无法恢复。与影子分页相比而言，简单分页使用的内存明显要小，并且可提高性能。

数据同步

UltraLiteJ 能够使数据与 MobiLink 11 进行同步。当与 MobiLink 同步时，必须使用 -x MobiLink 服务器选项。

UltraLiteJ 支持：

- MobiLink 用户验证
- MobiLink 用户验证脚本
- 基于发布的同步
- HTTP 和 HTTPS 网络协议
- 仅上载、仅下载、仅强制回应和完整上载/下载模式
- 同步观察器 API

在 BlackBerry 环境中，数据在设备和 BlackBerry Enterprise Server (BES) 之间始终处于加密状态。如果需要加密 BES 和 MobiLink 服务器之间的通信，则使用 HTTPS。

并发同步

通常在 UltraLiteJ 运行时每次仅允许一个线程。但在同步期间此规则可以出现例外。一个连接执行同步操作时，其它连接可以访问 UltraLiteJ 运行时，但具有以下特定限制：

- 同步期间，无法对数据库 Connection 对象执行任何模式操作（disableSynchronization、dropDatabase、dropTable、dropPublication、enableSynchronization、renameTable 或 schemaCreateBegin）。
- 同步操作正在进行时，其它任何线程都无法对正在同步的数据库调用同步方法。

连接可在同步期间（即提交下载的行之前）访问已下载的行，如果同步失败则这些行随后将消失。如果在同步期间连接修改了同步随后要更改的行，则同步将失败。如果在同步期间连接尝试修改同步已经更改的行，此修改尝试将失败。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

开发 UltraLiteJ 应用程序

目录

UltraLiteJ 开发简介 12
访问 UltraLiteJ 数据库存储 13
执行模式操作 16
使用 SQL 访问和操作数据 18
对数据进行加密和模糊处理 23
与 MobiLink 同步 25
部署 UltraLiteJ 应用程序 30
代码示例 31

本节介绍 UltraLiteJ 应用程序编程接口（Application Programming Interface，简称 API）。

UltraLiteJ 开发简介

UltraLiteJ 为 Java 应用程序提供了基本数据库功能。专为与 BlackBerry 智能手机配合使用而设计，但与 J2ME 和 J2SE 环境完全兼容。UltraLiteJ API 包含使用 SQL 语句连接到 UltraLiteJ 数据库、执行模式操作和维护数据所需的所有方法，而且还支持数据加密和同步等高级操作。

每个受支持平台的 API 存储在 UltraLiteJ 目录的 *UltraLite.jar* 文件中，该文件通常位于 SQL Anywhere 安装目录的 *UltraLite\UltraLiteJ* 文件夹下。

创建 UltraLiteJ 应用程序的基本步骤

创建 UltraLiteJ 应用程序时，通常要完成以下任务：

1. 创建一个新的 Configuration 对象。

   Configuration 对象定义 UltraLiteJ 数据库所在的位置，或者应在何处创建该数据库。还指定连接到数据库所需的用户名和口令。对不同的设备和非持久性数据库存储有 Configuration 对象各种变体可供使用。请参见“Configuration 接口”一节第 120 页。

2. 创建一个新的 Connection 对象。

   Connection 对象使用 Configuration 对象中定义的说明连接到 UltraLiteJ 数据库。如果该数据库不存在，则会自动创建。请参见“Connection 接口”一节第 122 页。

3. 应用 TableSchema、IndexSchema 和 ForeignKeySchema 对象。

   Connection 对象提供的模式方法允许您创建表、列、索引和外键。请参见“schemaCreateBegin 方法”一节第 140 页。

4. 生成 PreparedStatement 对象。

   PreparedStatement 对象查询与 Connection 对象关联的数据库。它接受支持的 SQL 语句，这些语句作为字符串进行传递。通过 PreparedStatement 对象，您可以更新数据库的内容。请参见“PreparedStatement 接口”一节第 176 页。

5. 生成 ResultSet 对象。

   当 Connection 对象执行了包含 SQL SELECT 语句的 PreparedStatement 时，便会创建 ResultSet 对象。通过 ResultSet 对象，您可以查看数据库的表内容。请参见“ResultSet 接口”一节第 180 页。

设置 UltraLiteJ 应用程序

在您首选的 Java IDE 中设置 UltraLiteJ 应用程序时，请确保您的项目配置正确，已可以使用位于 UltraLiteJ 目录下的 *UltraLite.jar* 资源文件。

使用以下语句将 UltraLiteJ 包导入您的 Java 文件中：

```
import ianywhere.ultralitej.*;
```

本文档包含的所有编码示例和教程都假定上述语句已指定，并且您很熟悉在首选 IDE 中开发 Java 应用程序。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

访问 UltraLiteJ 数据库存储

应用程序必须先连接到 UltraLiteJ 数据库，然后才能对数据执行操作。本节介绍如何使用指定的口令创建或连接到数据库。

Configuration 对象的实现

Configuration 用于创建并连接到数据库。API 中提供了 Configuration 的多种不同的实现方式。UltraLiteJ 支持的每种类型的数据库存储都存在唯一的实现方式。每种实现方式提供了一组不同的用于访问数据库存储的方法。

- **RIM 对象存储** 通过 ConfigObjectStore 实现。
- **记录存储** 通过 ConfigRecordStore 实现。
- **文件系统存储** 通过 ConfigFile 实现。
- **非持久性存储** 通过 ConfigNonPersistent 实现。

Connection 对象的属性

- **事务** 必须使用 Connection 的 commit 方法将事务提交到数据库。这些事务可以使用 rollback 方法回退。
- **预准备 SQL 语句** PreparedStatement 接口提供了用于处理 SQL 语句的方法。可使用 Connection 的 prepareStatement 方法创建 PreparedStatement。
- **同步** 可以通过 Connection 访问用于控制 MobiLink 同步的一组对象。
- **表操作** 使用 Connection 接口所提供的方法对 UltraLiteJ 数据库表进行访问和维护。

创建新的 UltraLiteJ 数据库

只能使用 API 创建 UltraLiteJ 数据库。您不能使用 Sybase Central 或 UltraLite 命令行实用程序创建新的数据库。

♦ 创建数据库

1. 创建一个新的引用数据库名的 Configuration。

   相应的语法取决于 Java 平台和客户端设备。在以下示例中，config 为 Configuration 对象的名称，DBname.ulj 为新数据库的名称。

   对于 J2ME BlackBerry 设备：

   ```
   ConfigObjectStore config =
       DatabaseManager.createConfigurationObjectStore("DBname.ulj");
   ```

   对于所有其它 J2ME 设备：

   ```
   ConfigRecordStore config =
       DatabaseManager.createConfigurationRecordStore("DBname.ulj");
   ```

   对于 J2SE 设备：

   ```
   ConfigFile config =
       DatabaseManager.createConfigurationFile("DBname.ulj");
   ```

或者，您也可以创建所有平台都支持的非持久性数据库 Configuration：

```
ConfigNonPersistent config =
    DatabaseManager.createConfigurationNonPersistent("DBname.ulj");
```

2. 使用 setPassword 方法设置新数据库口令：

```
config.setPassword("my_password");
```

3. 创建一个新的 Connection：

```
Connection conn = DatabaseManager.createDatabase(config);
```

createDatabase 方法结束数据库创建过程并连接到该数据库。调用该方法后，您可以执行模式操作和数据操作，但不能再更改数据库的名称、口令和页面大小。

连接到现有数据库

UltraLiteJ 数据库必须已存在于客户端设备上，您才能与其连接。

♦ 连接到现有数据库

1. 创建一个新的引用数据库名的 Configuration。

   相应的语法取决于 Java 平台和客户端设备。在以下示例中，config 为 Configuration 对象的名称，DBname.ulj 为数据库的名称。

   对于 J2ME BlackBerry 设备：

```
ConfigObjectStore config =
    DatabaseManager.createConfigurationObjectStore("DBname.ulj");
```

   对于所有其它 J2ME 设备：

```
ConfigRecordStore config =
    DatabaseManager.createConfigurationRecordStore("DBname.ulj");
```

   对于 J2SE 设备：

```
ConfigFile config =
    DatabaseManager.createConfigurationFile("DBname.ulj");
```

   或者，您也可以连接到所有平台都支持的非持久性数据库 Configuration：

```
ConfigNonPersistent config =
    DatabaseManager.createConfigurationNonPersistent("DBname.ulj");
```

2. 使用 setPassword 方法指定数据库口令：

```
config.setPassword("my_password");
```

3. 创建一个新的 Connection：

```
Connection conn = DatabaseManager.connect(config);
```

connect 方法会结束数据库连接过程。如果该数据库不存在，则抛出错误。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

从数据库断开连接

使用 DatabaseManager 类的 release 方法来断开与 UltraLiteJ 数据库的连接。release 方法会关闭 Connection 以及与其关联的所有属性。

另请参见

- “示例：创建数据库”一节第 32 页
- “DatabaseManager 类”一节第 147 页

执行模式操作

UltraLiteJ 提供了允许您在数据库中创建表、列、索引和键的模式方法。本节介绍如何执行模式操作。

模式对象的类型

- **表模式**　Connection 接口中的方法用于访问表属性。模式通过直接使用 TableSchema 接口访问。请参见“TableSchema 接口”一节第 240 页。
- **列模式**　TableSchema 接口中的方法用于访问列属性。模式通过直接使用 ColumnSchema 接口访问。请参见“ColumnSchema 接口”一节第 104 页。
- **索引模式**　TableSchema 接口中的方法用于访问索引。模式通过直接使用 IndexSchema 接口访问。请参见“IndexSchema 接口”一节第 173 页。
- **外键模式**　Connection 接口中的方法用于访问外键。模式通过直接使用 ForeignKeySchema 接口访问。请参见“ForeignKeySchema 接口”一节第 171 页。

创建新表、列、索引和键

必须使用 API 执行所有表、列、索引和键操作，并且仅当连接的数据库处于 "模式创建" 模式时才能执行。

♦ 执行模式操作

1. 连接到 UltraLiteJ 数据库。

   本示例假定数据库已连接到 Connection 对象 conn。有关如何连接到 UltraLiteJ 数据库的详细信息，请参见“访问 UltraLiteJ 数据库存储”一节第 13 页。

2. 使用以下代码将 Connection 置于 "模式创建" 模式：

   ```
   conn.schemaCreateBegin();
   ```

   "模式创建" 模式会禁止执行数据操作并将其它到数据库的连接封锁在外。

3. 执行所有表、列、索引和键操作。

   以下过程演示如何创建一个包含名为 emp_number 的整数列的新 Employee 表。emp_number 列是 Employee 表的主索引，并同时充当引用 Security 表中的整数列 access_number 的外键。

   - **创建新表**　使用 createTable 方法来定义表的名称并将结果指派给 TableSchema：

     ```
     TableSchema table_schema = conn.createTable("Employee");
     ```

     将结果指派给 TableSchema 后，您便可对该表执行更加复杂的模式操作。有关详细信息，请参见“TableSchema 接口”一节第 240 页。

   - **向表中添加列**　使用 createColumn 方法来定义列名和类型：

     ```
     table_schema.createColumn("emp_number", Domain.INTEGER);
     ```

   - **将主索引指派到列**

     a. 使用 createPrimaryIndex 方法在表上创建新的主索引并将结果指派给 IndexSchema：

```
IndexSchema index_schema =
    table_schema.createPrimaryIndex("prime_keys");
```

b. 使用 addColumn 方法来指定主索引列和排序顺序：

```
index_schema.addColumn("emp_number", IndexSchema.ASCENDING);
```

- **将外键指派到列**　该过程假定数据库中已存在一个具有 access_number 整数列的 Security 表。如果该表不存在，则使用上面所列的过程进行创建。

a. 使用 createForeignKey 方法来指定创建外键时所涉及的表：

```
ForeignKeySchema foreign_key_schema = conn.createForeignKey(
    "Employee",
    "Security",
    "fk_emp_to_sec"
);
```

第一个参数引用要包含外键的表；第二个参数引用包含了外键所引用的列的表。

b. 使用 addColumnReference 方法来指定创建外键时所涉及的两个列：

```
foreign_key_schema.addColumnReference("emp_number",
"access_number");
```

第一个参数引用在第一个表中要变为外键的列名；第二个参数引用在第二个表中外键所引用的列名。

4. 使 Connection 退出 "模式创建" 模式：

```
conn.schemaCreateComplete();
```

另请参见

- “Connection 接口”一节第 122 页
- “ColumnSchema 接口”一节第 104 页
- “ForeignKeySchema 接口”一节第 171 页
- “IndexSchema 接口”一节第 173 页
- “TableSchema 接口”一节第 240 页

使用 SQL 访问和操作数据

支持的 SQL 语句

UltraLite 支持的某些 SQL 语句不受 UltraLiteJ 的支持。以下是 UltraLiteJ 所支持的 SQL 语句的完整列表：

SQL 语句	注释和限制：
ALTER TABLE	请参见"UltraLite ALTER TABLE 语句"一节 《UltraLite - 数据库管理和参考》。不支持 MAX HASH SIZE。
COMMIT	请参见"UltraLite COMMIT 语句"一节 《UltraLite - 数据库管理和参考》。
CREATE INDEX	请参见"UltraLite CREATE INDEX 语句"一节 《UltraLite - 数据库管理和参考》。不支持 MAX HASH SIZE。
CREATE TABLE	请参见"UltraLite CREATE TABLE 语句"一节 《UltraLite - 数据库管理和参考》。不支持 MAX HASH SIZE。
DELETE	请参见"UltraLite DELETE 语句"一节 《UltraLite - 数据库管理和参考》。
DROP INDEX	请参见"UltraLite DROP INDEX 语句"一节 《UltraLite - 数据库管理和参考》。
DROP TABLE	请参见"UltraLite DROP TABLE 语句"一节 《UltraLite - 数据库管理和参考》。
INSERT	请参见"UltraLite INSERT 语句"一节 《UltraLite - 数据库管理和参考》。
ROLLBACK	请参见"UltraLite ROLLBACK 语句"一节 《UltraLite - 数据库管理和参考》。
SELECT	有关 SELECT 语句的一般说明，请参见"UltraLite SELECT 语句"一节 《UltraLite - 数据库管理和参考》。 以下限制适用： ● 不支持 SQLCODE 函数。 ● 不支持数学函数 ACOS、ASIN、ATAN、ATAN2 和 POWER。
START SYNCHRONIZATION DELETE	请参见"UltraLite START SYNCHRONIZATION DELETE 语句"一节 《UltraLite - 数据库管理和参考》。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQL 语句	注释和限制：
STOP SYNCHRONIZATION DELETE	请参见“UltraLite STOP SYNCHRONIZATION DELETE 语句”一节 《UltraLite - 数据库管理和参考》。
TRUNCATE TABLE	请参见“UltraLite TRUNCATE TABLE 语句”一节 《UltraLite - 数据库管理和参考》。
UPDATE	有关 SELECT 语句的一般说明，请参见“UltraLite UPDATE 语句”一节 《UltraLite - 数据库管理和参考》。 不支持 JOIN 子句。

使用 INSERT、UPDATE 和 DELETE 进行数据操作

可使用 PreparedStatement 的 execute 方法来执行 SQL 数据操作。PreparedStatement 使用用户定义的 SQL 语句查询数据库。

将 SQL 语句应用于 PreparedStatement 时，使用 **?** 字符来表示查询参数。对于任何 INSERT、UPDATE 或 DELETE 语句，每个 **?** 参数都根据其在语句中的顺序位置来进行引用。例如，第一个 **?** 引用为参数 1，第二个引用为参数 2。

♦ **向表中插入行**

1. 将新 SQL 语句准备为一个字符串。

```
String sql_string =
    "INSERT INTO Department(dept_no, name) VALUES( ?, ? )";
```

2. 将该字符串传递给 PreparedStatement。

```
PreparedStatement inserter =
    conn.prepareStatement(sql_string);
```

3. 使用 set 方法将输入值传递给 PreparedStatement。

   本示例将 dept_no 设置为 101，并引用为参数 1，将 name 设置为 "Electronics"，并引用为参数 2。

```
inserter.set(1, 101);
inserter.set(2, "Electronics");
```

4. 执行该语句。

```
inserter.execute();
```

5. 关闭 PreparedStatement 以释放资源。

```
inserter.close()
```

6. 提交对数据库的所有更改。

```
conn.commit();
```

♦ 更新表中的行

1. 将新 SQL 语句准备为一个字符串。

```
String sql_string =
    "UPDATE Department SET dept_no = ? WHERE dept_no = ?";
```

2. 将该字符串传递给 PreparedStatement。

```
PreparedStatement updater =
    conn.prepareStatement(sql_string);
```

3. 使用 set 方法将输入值传递给 PreparedStatement。

```
updater.set(1, 102);
updater.set(2, 101);
```

   上述示例与声明以下 SQL 语句等效：

```
UPDATE Department SET dept_no = 102 WHERE dept_no = 101
```

4. 执行该语句。

```
updater.execute();
```

5. 关闭 PreparedStatement 以释放资源。

```
updater.close()
```

6. 提交对数据库的所有更改。

```
conn.commit();
```

♦ 删除表中的行

1. 将新 SQL 语句准备为一个字符串。

```
String sql_string =
    "DELETE FROM Department WHERE dept_no = ?";
```

2. 将该字符串传递给 PreparedStatement。

```
PreparedStatement deleter =
    conn.prepareStatement(sql_string);
```

3. 使用 set 方法将输入值传递给 PreparedStatement。

```
deleter.set(1, 102);
```

   上述示例与声明以下 SQL 语句等效：

```
DELETE FROM Department WHERE dept_no = 102
```

4. 执行该语句。

```
deleter.execute();
```

5. 关闭 PreparedStatement 以释放资源。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
deleter.close()
```

6. 提交对数据库的所有更改。

```
conn.commit();
```

使用 SELECT 检索数据

可以使用 PreparedStatement 的 executeQuery 方法来检索数据，该方法使用用户定义的 SQL 语句查询数据库，并将查询结果作为 ResultSet 返回。然后，可对该 ResultSet 进行遍历以读取查询到的数据。

浏览 ResultSet 对象

ResultSet 包含以下方法来浏览 SQL SELECT 语句的查询结果：

- **next**　移至下一行。
- **previous**　移至上一行。

使用 ResultSet 检索数据

♦ **从数据库选择数据**

1. 将新 SQL 语句准备为一个字符串。

```
String sql_string =
    "SELECT * FROM Department ORDER BY dept_no";
```

2. 将该字符串传递给 PreparedStatement。

```
PreparedStatement select_statement =
    conn.prepareStatement(sql_string);
```

3. 执行该语句并将查询结果指派给 ResultSet。

```
ResultSet cursor =
    select_statement.executeQuery();
```

4. 遍历该 ResultSet 并检索数据。

```
// Get the next row stored in the ResultSet.
cursor.next();

// Store the data from the first column in the table.
int dept_no = cursor.getInt(1);

// Store the data from the second column in the table.
String dept_name = cursor.getString(2);
```

5. 关闭 ResultSet 以释放资源。

```
cursor.close();
```

6. 关闭 PreparedStatement 以释放资源。

```
select_statement.close()
```

使用 COMMIT 和 ROLLBACK 管理事务

UltraLiteJ 不支持 AutoCommit 模式。必须使用 Connection 接口支持的方法显式地提交或回退事务。

要提交事务，可使用 commit 方法。

要回退事务，可使用 rollback 方法。

另请参见

- “commit 方法”一节第 128 页
- “rollback 方法”一节第 140 页

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

对数据进行加密和模糊处理

缺省情况下，存储在 UltraLiteJ 数据库中的数据不加密。可使用 API 对数据进行加密或模糊处理。加密为数据提供了非常安全的表示方式，而模糊处理是一种简化的安全保护方式，以防止随意查看数据库内容。

♦ **对数据库中数据进行加密或模糊处理**

1. 创建一个实现 EncryptionControl 接口的类。

   以下示例创建一个实现加密接口的新类 Encryptor。

```
static class Encryptor
    implements EncryptionControl
{
```

2. 在新类中实现 initialize、encrypt 和 decrypt 方法。

   该类应与以下所示代码类似：

```
static class Encryptor
    implements EncryptionControl
{
    /** Decrypt a page stored in the database.
     * @param page_no the number of the page being decrypted
     * @param src the encrypted source page which was read from the
database
     * @param tgt the decrypted page (filled in by method)
     */
    public void decrypt( int page_no, byte[] src, byte[] tgt )
        throws ULjException
    {
        // Your decryption method goes here.
    }

    /** Encrypt a page stored in the database.
     * @param page_no the number of the page being encrypted
     * @param src the unencrypted source
     * @param tgt the encrypted target page which will be written to the
database (filled in by method)
     */
    public void encrypt( int page_no, byte[] src, byte[] tgt )
        throws ULjException
    {
        // Your encryption method goes here.
    }

    /** Initialize the encryption control with a password.
     * @param password the password
     */
    public void initialize(String password)
        throws ULjException
    {
        // Your initialization method goes here.
    }
}
```

   有关 EncryptionControl 方法的详细信息，请参见“EncryptionControl 接口”一节第 169 页。

3. 将数据库进行配置以使用新类进行加密控制。

   可使用 setEncryption 方法指定加密控制。以下示例假定您已创建一个引用数据库名称的新 Configuration 对象 config：

   ```
   config.setEncryption(new Encryptor());
   ```

4. 连接到数据库。

   现在，数据库中添加或修改的所有数据均已加密。

注意
加密和模糊处理对非持久性数据库存储不可用。

另请参见

- “示例：对数据进行模糊处理”一节第 43 页
- “示例：对数据进行加密”一节第 47 页
- “EncryptionControl 接口”一节第 169 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

与 MobiLink 同步

将 UltraLiteJ 用作 MobiLink 客户端

要同步数据，您的应用程序必须执行以下步骤：

1. 实例化 syncParms 对象，该对象包含以下相关信息：统一数据库（服务器名、端口号）、要同步的数据库的名称以及要同步的表的定义。
2. 使用 syncParms 对象从连接对象中调用同步方法，以执行同步。

可在表级别定义要同步的数据。不能为表的一部分配置同步。

另请参见

- "SyncParms 类" 一节第 219 页
- "SyncResult 类" 一节第 234 页

示例

本示例介绍如何使用 UltraLiteJ 应用程序同步数据。

要将具有 SQL Anywhere 11 CustDB 的 MobiLink 服务器作为统一数据库启动，请运行 *samples-dir \UltraLiteJ* 目录下的 *start_ml.bat*。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * Sync: sample program to demonstrate Database synchronization.
 *
 * Requires starting the MobiLink Server Sample using start_ml.bat
 */
public class Sync
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Demo1.ulj" );

            Connection conn = DatabaseManager.createDatabase( config );
            conn.schemaCreateBegin();

            TableSchema table_schema = conn.createTable( "ULCustomer" );
            table_schema.createColumn( "cust_id", Domain.INTEGER );
            table_schema.createColumn( "cust_name", Domain.VARCHAR, 30 );
            IndexSchema index_schema =
table_schema.createPrimaryIndex( "prime_keys" );
            index_schema.addColumn( "cust_id", IndexSchema.ASCENDING );

            conn.schemaCreateComplete();
```

```
            //
            // Synchronization
            //

            // Version set for MobiLink 11.0.x
           SyncParms syncParms = conn.createSyncParms( SyncParms.HTTP_STREAM,
"50", "custdb 11.0" );
            syncParms.getStreamParms().setPort( 9393 );
            conn.synchronize( syncParms );
            SyncResult result = syncParms.getSyncResult();
            Demo.display(
                    "*** Synchronized *** bytes sent=" +
result.getSentByteCount()
                    + ", bytes received=" + result.getReceivedByteCount()
                    + ", rows received=" + result.getReceivedRowCount()
                );

            conn.release();

        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

UltraLiteJ 同步流的网络协议选项

与 MobiLink 服务器同步时，必须在您的应用程序中设置网络协议。每个数据库都通过网络协议进行同步。有两种网络协议可用于 UltraLiteJ—HTTP 和 HTTPS。

对于所设置的网络协议，可以从一组相应的协议选项中进行选择，以确保 UltraLiteJ 应用程序可以找到 MobiLink 服务器并与之通信。网络协议选项提供了诸如寻址信息（主机和端口）和协议特定信息之类的信息。要确定可将哪些选项用于您使用的流类型，请参见“MobiLink 客户端网络协议选项汇总”一节 《MobiLink - 客户端管理》。

设置 HTTP 网络协议

HTTP 网络协议使用 UltraLiteJ API 中的 StreamHTTPParms 接口进行设置。使用接口方法来指定 MobiLink 服务器上定义的网络协议选项。有关网络选项的完整列表，请参见“StreamHTTPParms 接口”一节第 205 页。

设置 HTTPS 网络协议

HTTPS 网络协议使用 UltraLiteJ API 中的 StreamHTTPSParms 接口进行设置。使用接口方法来指定 MobiLink 服务器上定义的网络协议选项。有关网络选项的完整列表，请参见“StreamHTTPSParms 接口”一节第 209 页。

同步 CustDB 应用程序

CustDB（客户数据库）是一个随 SQL Anywhere 一起安装的示例数据库。CustDB 数据库是一个简单的销售订单数据库。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

查找和部署应用程序

UltraLiteJ 的安装包括与 CustDB 数据库相关的示例 BlackBerry 应用程序。该应用程序名为 CustDB，其源代码和相关文件在 *sample-dir\ultralitej\CustDB* 目录中。CustDB 目录包含可使用 Research In Motion (RIM) JDE 打开的项目文件。

通过为 BlackBerry 浏览器提供下面的 URL 可将 CustDB 应用程序直接下载到 BlackBerry（以了解它的工作方式）：

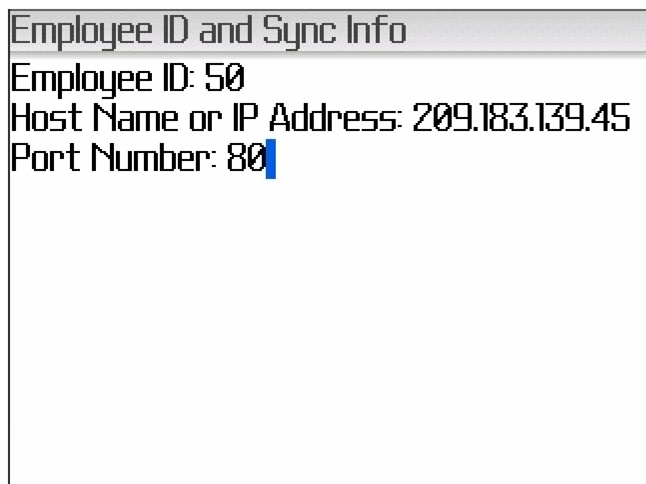
```
http://ultralitej.sybase.com/
```

与 CustDB 应用程序相关的文件

- ***CustDB.java*** 此文件包含所有基本数据库访问方法。这些方法包括创建和连接到数据库、插入、删除和更新订单。此文件包含许多对后端服务器通信的数据库调用。
- ***SchemaCreator.java*** 这些文件包含使用 UltraLiteJ 在设备上创建表的代码。

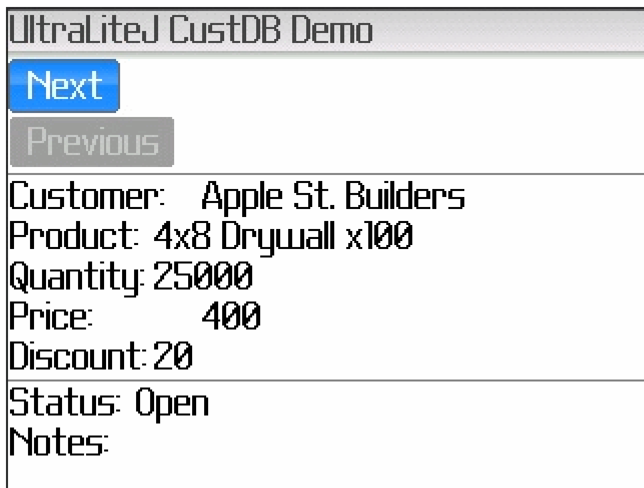
使用 CustDB 应用程序

刚开始启动时，CustDB 程序收集用来与寄存 CustDB 数据库的服务器进行交互的信息。指定用于查询的 Employee ID（建议 "50"）、寄存数据的服务器的主机名或 IP 地址以及用于连接到服务器的端口号。



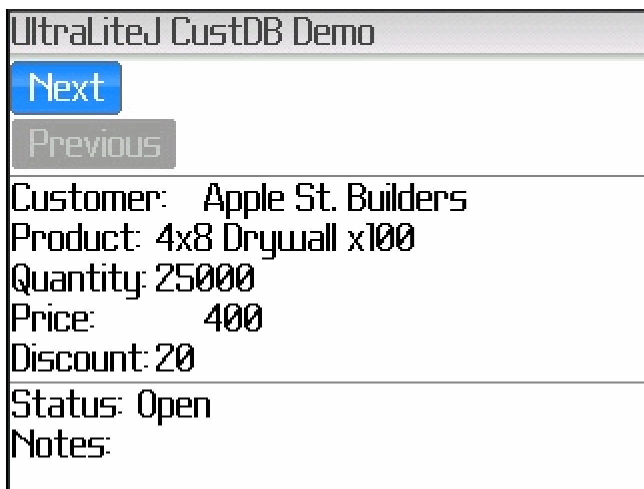
指定这些值并保存设置（[菜单] » [保存]）后，应用程序即与指定服务器同步。应用程序仅从服务器上下载与对应于指定雇员编号 (50) 的 Employee ID 相匹配的订单。仅选择仍处于 open 状态的订单（订单可采用以下三种状态之一：Open、Approved 或 Denied）。

每个订单都显示在屏幕上，其中包含以下信息：客户名、订购的产品、订购量、价格和折扣。屏幕还显示订单的当前状态以及与该订单相关的任何注意事项。



在此屏幕上您可以将注意事项添加到订单或更改订单的状态（更改为 Approved 或 Denied）。您可以使用 [**Next**] 和 [**Previous**] 按钮浏览订单。

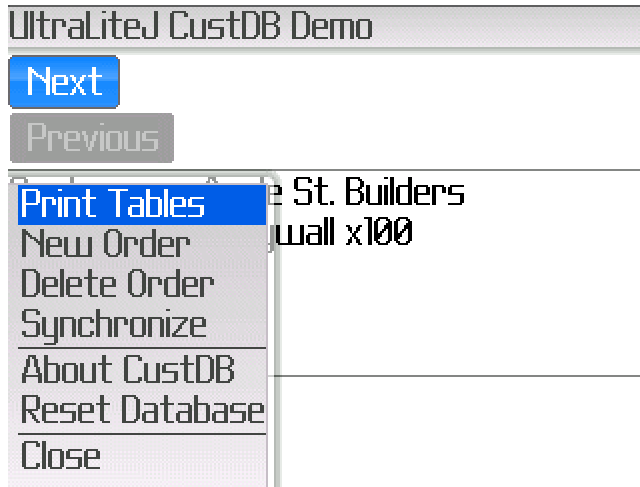
CustDB 程序也允许您将新订单添加到数据库中。要添加新订单，则单击 [**Menu**] » [**New Order**]。



您可以输入所需的数量和折扣值。

退出应用程序前，从主菜单中选择 [**Synchronize**] 以使更改和新订单与服务器保持同步。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1


UltraLiteJ CustDB Demo
Next
Previous
e St. Builders
wall x100
Print Tables
New Order
Delete Order
Synchronize
About CustDB
Reset Database
Close

部署 UltraLiteJ 应用程序

要成功运行 UltraLiteJ 应用程序，必须使用您的分发来部署 UltraLiteJ API。下表列出了各种不同的 UltraLiteJ 部署所需的文件的列表。所有文件路径均相对于您的 SQL Anywhere 安装目录的 *UltraLite\UltraLiteJ* 目录。

部署类型	所需文件
BlackBerry 智能手机	*J2meRim11\UltraLiteJ.cod* *J2meRim11\UltraLiteJ.jad*[1]
J2ME	*J2me11\UltraLiteJ.jar*
J2SE	*J2se\UltraLiteJ.jar*

[1] 仅当采用空中传输（over-the-air，简称 OTA）部署方式时才需要。或者，您也可以创建自己的 jad 文件来使用您的应用程序部署 UltraLiteJ。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

代码示例

本节包含使用 UltraLiteJ API 的 Java 代码示例。这些示例包含 demo 类，该类用于显示消息和处理 ULjException 对象，以便于调试。

所有代码示例均可在 *samples-dir/UltraLiteJ* 目录中找到。操作文件内容之前，请创建原始源代码的副本。

示例：Demo 类

此类由文档的本节中包含的所有示例所使用。

```
// ********************************************************************
// Copyright 2006-2008 iAnywhere Solutions, Inc.  All rights reserved.
// ********************************************************************
package ianywhere.ultralitej.demo;

//import java.io.*;
import ianywhere.ultralitej.*;
//import ianywhere.ultralitej.implementation.*;

/**
 * Demonstration class.
 *
 * <p>This class is not part of the Database library.  It is used
 * only by the demonstration programs.
 *
 * @author ianywhere
 * @version 1.0
 */
public class Demo
{
    /** Display a message.
     * @param msg message to be displayed
     */
    public static void display( String msg )
    {
        System.out.println( msg );
    }

    /** Display a message.
     * @param msg1 message(1) to be displayed
     * @param msg2 message(2) to be displayed
     */
    public static void display( String msg1, String msg2 )
    {
        display( msg1 + msg2 );
    }

    /** Display a message.
     * @param msg1 message(1) to be displayed
     * @param msg2 message(2) to be displayed
     * @param msg3 message(3) to be displayed
     */
    public static void display( String msg1, String msg2, String msg3 )
    {
        display( msg1 + msg2 + msg3 );
    }
```

```
    /** Display a message.
     * @param msg1 message(1) to be displayed
     * @param msg2 message(2) to be displayed
     * @param msg3 message(3) to be displayed
     * @param msg4 message(4) to be displayed
     */
    public static void display( String msg1, String msg2, String msg3, String
msg4 )
    {
        display( msg1 + msg2 + msg3 + msg4 );
    }

    /** Display message for an exception.
     * @param exc ULjException containing message
     */
    public static void displayException( ULjException exc )
    {
        display( exc.getMessage() );
    }

    /** Display message for an exception.
     * @param exc ULjException containing message
     */
    public static void displayExceptionFull( ULjException exc )
    {
        display( exc.getMessage() );
    }
}
```

示例：创建数据库

本示例演示如何在 J2SE Java 环境中创建文件系统数据库存储。Configuration 对象用于创建数据库。一旦创建完成，则返回 Connection 对象。要创建表，调用 schemaUpdateBegin 方法以启动对基础模式的更改，schemaUpdateComplete 方法完成对模式的更改。

注意：

- UltraLiteJ 中的表没有所有者且只能通过名称进行标识。
- Domain 接口定义常量以表示 UltraLiteJ 表的列中支持的各种数据类型。
- 保证主索引 (createPrimaryIndex) 是唯一的。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * CreateDb: sample program to demonstrate Database creation.
 */
public class CreateDb
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
```


Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Demo1.ulj" );

            Connection conn = DatabaseManager.createDatabase( config );
            conn.schemaCreateBegin();

            TableSchema table_schema = conn.createTable( "Employee" );
            table_schema.createColumn( "number", Domain.INTEGER );
            table_schema.createColumn( "last_name", Domain.VARCHAR, 32 );
            table_schema.createColumn( "first_name", Domain.VARCHAR, 32 );
            table_schema.createColumn( "age", Domain.INTEGER );
            table_schema.createColumn( "dept_no", Domain.INTEGER );
            IndexSchema index_schema =
table_schema.createPrimaryIndex( "prime_keys" );
            index_schema.addColumn( "number", IndexSchema.ASCENDING );

            table_schema = conn.createTable( "Department" );
            table_schema.createColumn( "dept_no", Domain.INTEGER );
            table_schema.createColumn( "name", Domain.VARCHAR, 50 );
            index_schema = table_schema.createPrimaryIndex( "prime_keys" );
            index_schema.addColumn( "dept_no", IndexSchema.ASCENDING );

            ForeignKeySchema foreign_key_schema =
conn.createForeignKey( "Employee", "Department", "fk_emp_to_dept" );
            foreign_key_schema.addColumnReference( "dept_no", "dept_no" );

            conn.schemaCreateComplete();

            conn.release();

            Demo.display( "CreateDb completed successfully" );

        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

示例：插入行

本示例演示如何在 UltraLiteJ 数据库中插入行。

注意：

- 仅当从 Connection 对象调用了 commit 方法时，插入的数据才会保留在数据库中。
- 当已插入行但尚未提交时，该行对其它连接可见。因此，其它连接可能会检索尚未实际提交的行数据。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * LoadDb -- sample program to demonstrate loading a Database.
 */
public class LoadDb
{
    /**
     * Add a Department row.
     * @param conn connection to Database
```

```
     * @param dept_no department number
     * @param dept_name department name
     */
    private static void addDepartment( PreparedStatement inserter, int
dept_no, String dept_name )
        throws ULjException
    {
        inserter.set( 1 /* "dept_no" */, dept_no );
        inserter.set( 2 /* "name" */, dept_name );
        inserter.execute();
    }

    /**
     * Add an Employee row.
     * @param conn connection to Database
     * @param emp_no employee number
     * @param last_name employee last name
     * @param first_name employee first name
     * @param age employee age
     * @param dept_no department number where employee works
     */
    private static void addEmployee( PreparedStatement inserter, int emp_no,
String last_name
                                   , String first_name, int age, int
dept_no )
        throws ULjException
    {
        inserter.set( 1 /* "number" */, emp_no );
        inserter.set( 2 /* "last_name" */, last_name );
        inserter.set( 3 /* "first_name" */, first_name );
        inserter.set( 4 /* "age" */, age );
        inserter.set( 5 /* "dept_no" */, dept_no );
        inserter.execute();
    }

    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Demo1.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement inserter;

          inserter = conn.prepareStatement( "INSERT INTO Department( dept_no,
name ) VALUES( ?, ? )" );
            addDepartment( inserter, 100, "Engineering" );
            addDepartment( inserter, 110, "Sales" );
            addDepartment( inserter, 103, "Marketing" );
            inserter.close();

            inserter = conn.prepareStatement(
               "INSERT INTO employee( \"number\", last_name, first_name, age,
dept_no ) VALUES( ?, ?, ?, ?, ? )"
            );
            addEmployee( inserter, 1000, "Welch", "James", 58, 100 );
            addEmployee( inserter, 1010, "Iverson", "Victoria", 23, 103 );
            inserter.close();
```

```
                conn.commit();
                conn.release();
                Demo.display( "LoadDb completed successfully" );
            } catch( ULjException exc ) {
                Demo.displayException( exc );
            }
        }
    }
```

示例：读取表

在本示例中，从 Connection 中获取 PreparedStatement 对象，从 PreparedStatement 中获取 ResultSet 对象。每次可获取后继行时，ResultSet 的 next 方法都会返回 true。当前行中的列值可随后从 ResultSet 对象中获取。

注意：

- 创建 ResultSet 时，它会定位在结果集的第一行之前。代码必须调用 next 方法将其移动到表中的第一行。
- UltraLiteJ 中的表名和列名不区分大小写。可以只通过列名（“age”）引用列，也可以通过用表名限定列名（“Employee.age”）来引用列。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * ReadSeq -- sample program to demonstrate reading a Database table
 * sequentially.
 */
public class ReadSeq
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Demo1.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement stmt = conn.prepareStatement( "SELECT * FROM
Employee ORDER BY number" );
            ResultSet cursor = stmt.executeQuery();
            for( ; cursor.next(); ) {
                /* Can't access columns by name because no meta data */
                int emp_no = cursor.getInt( 1 /* "number" */ );
                String last_name = cursor.getString( 2 /* "last_name" */ );
                String first_name = cursor.getString( 3 /* "first_name" */ );
                int age = cursor.getInt( 4 /* "age" */ );
                Demo.display( first_name + ' ' + last_name );
                Demo.display( "  empl. no = "
                            , Integer.toString( emp_no )
                            , "  age = "
```

```
                            , Integer.toString( age ) );
            }
            cursor.close();
            stmt.close();
            conn.release();
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

示例：内连接操作

本示例演示如何执行内连接操作。在本方案中，每个员工都具有相应的部门信息。连接操作将 employee 表中的数据与 department 表中的相应数据相关联。通过使用 employee 表的部门编号找到 department 表中的相关信息来实现关联。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * ReadInnerJoin -- sample program to demonstrate reading the Employee table
 * and joining to each row the corresponding Department row.
 */
public class ReadInnerJoin
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Demo1.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement stmt = conn.prepareStatement(
                "SELECT E.number, E.last_name, E.first_name, E.age,"
                + " E.dept_no, D.name"
                + " FROM Employee E"
                + " JOIN Department D ON E.dept_no = D.dept_no"
                + " ORDER BY E.number"
            );
            ResultSet cursor = stmt.executeQuery();
            for( ; cursor.next(); ) {
                /* Can't access columns by name because no meta data */
                int emp_no = cursor.getInt( 1 /* "E.number" */ );
                String last_name = cursor.getString( 2 /* "E.last_name" */ );
                String first_name = cursor.getString( 3 /* "E.first_name"
*/ );
                int age = cursor.getInt( 4 /* "E.age" */ );
                int dept_no = cursor.getInt( 5 /* "E.dept_no" */ );
                String dept_name = cursor.getString( 6 /* "D.name" */ );
                System.out.println( first_name + ' ' + last_name );
                System.out.print( "  empl. no = " );
                System.out.print( emp_no );
                System.out.print( "  dept = " );
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
                System.out.println( dept_no );
                System.out.print( "  age = " );
                System.out.print( age );
                System.out.println( ", " + dept_name );
            }
            cursor.close();
            stmt.close();
            conn.release();
            Demo.display( "ReadInnerJoin completed successfully" );
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

示例：创建销售数据库

在此示例中，创建面向销售的数据库。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * CreateDb: sample program to demonstrate creation of simple sales Database
and
 * load it with some data.
 * <p>The program also illustrates the use of ordinals when inserting rows
into tables.
 */
public class CreateSales
{
    static int ORDINAL_INVOICE_INV_NO;
    static int ORDINAL_INVOICE_NAME;
    static int ORDINAL_INVOICE_DATE;

    static int ORDINAL_INV_ITEM_INV_NO;
    static int ORDINAL_INV_ITEM_ITEM_NO;
    static int ORDINAL_INV_ITEM_PROD_NO;
    static int ORDINAL_INV_ITEM_QUANTITY;
    static int ORDINAL_INV_ITEM_PRICE;

    static int ORDINAL_PROD_NO;
    static int ORDINAL_PROD_NAME;
    static int ORDINAL_PROD_PRICE;

    /** Create the Database.
     * @return connection for a new Database
     */
    private static Connection createDatabase()
        throws ULjException
    {
        Configuration config =
DatabaseManager.createConfigurationFile( "Sales.ulj" );
        Connection conn = DatabaseManager.createDatabase( config );

        conn.schemaCreateBegin();

        TableSchema table_schema = conn.createTable( "Product" );
        table_schema.createColumn( "prod_no", Domain.INTEGER );
        table_schema.createColumn( "prod_name", Domain.VARCHAR, 32 );
        table_schema.createColumn( "price", Domain.NUMERIC, 9, (short)2 );
```

```
        IndexSchema index_schema =
table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "prod_no", IndexSchema.ASCENDING );

        table_schema = conn.createTable( "Invoice" );
        table_schema.createColumn( "inv_no", Domain.INTEGER );
        table_schema.createColumn( "name", Domain.VARCHAR, 50 );
        table_schema.createColumn( "date", Domain.DATE );
        index_schema = table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "inv_no", IndexSchema.ASCENDING );

        table_schema = conn.createTable( "InvoiceItem" );
        table_schema.createColumn( "inv_no", Domain.INTEGER );
        table_schema.createColumn( "item_no", Domain.INTEGER );
        table_schema.createColumn( "prod_no", Domain.INTEGER );
        table_schema.createColumn( "quantity", Domain.INTEGER );
        table_schema.createColumn( "price", Domain.NUMERIC, 9, (short)2 );
        index_schema = table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "inv_no", IndexSchema.ASCENDING );
        index_schema.addColumn( "item_no", IndexSchema.ASCENDING );

        conn.schemaCreateComplete();

        return conn;
    }

    /** Populate the Database.
     * @param conn connection to Database
     */
    private static void populateDatabase( Connection conn )
        throws ULjException
    {
        PreparedStatement ri_product = conn.prepareStatement(
            "INSERT INTO Product( prod_no, prod_name, price )
VALUES( ?, ?, ? )"
            );
        ORDINAL_PROD_NO = 1;
        ORDINAL_PROD_NAME = 2;
        ORDINAL_PROD_PRICE = 3;
        addProduct( ri_product, 2001, "blue screw", ".03" );
        addProduct( ri_product, 2002, "red screw", ".09" );
        addProduct( ri_product, 2004, "hammer", "23.99" );
        addProduct( ri_product, 2005, "vice", "39.99" );
        ri_product.close();

        PreparedStatement ri_invoice = conn.prepareStatement(
            "INSERT INTO Invoice( inv_no, name, \"date\" )"
            + " VALUES( :inv_no, :name, :inv_date )"
        );
        ORDINAL_INVOICE_INV_NO = ri_invoice.getOrdinal( "inv_no" );
        ORDINAL_INVOICE_NAME = ri_invoice.getOrdinal( "name" );
        ORDINAL_INVOICE_DATE = ri_invoice.getOrdinal( "inv_date" );

        PreparedStatement ri_item = conn.prepareStatement(
            "INSERT INTO InvoiceItem( inv_no, item_no, prod_no, quantity,
price )"
            + " VALUES( ?, ?, ?, ?, ? )"
        );
        ORDINAL_INV_ITEM_INV_NO = 1;
        ORDINAL_INV_ITEM_ITEM_NO = 2;
        ORDINAL_INV_ITEM_PROD_NO = 3;
        ORDINAL_INV_ITEM_QUANTITY = 4;
        ORDINAL_INV_ITEM_PRICE = 5;
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
        addInvoice( ri_invoice, 2006001, "Jones Mfg.", "2006/12/23" );
        addInvoiceItem( ri_item, 2006001, 1, 2001, 3000, ".02" );
        addInvoiceItem( ri_item, 2006001, 2, 2002, 5000, ".08" );

        addInvoice( ri_invoice, 2006002, "Smith Inc.", "2006/12/24" );
        addInvoiceItem( ri_item, 2006002, 1, 2004, 2, "23.99" );
        addInvoiceItem( ri_item, 2006002, 2, 2005, 3, "39.99" );

        addInvoice( ri_invoice, 2006003, "Lee Ltd.", "2006/12/24" );
        addInvoiceItem( ri_item, 2006003, 1, 2004, 5, "23.99" );
        addInvoiceItem( ri_item, 2006003, 2, 2005, 4, "39.99" );
        addInvoiceItem( ri_item, 2006003, 3, 2001, 800, ".03" );
        addInvoiceItem( ri_item, 2006003, 4, 2002, 700, ".09" );

        ri_item.close();
        ri_invoice.close();

        conn.commit();
    }

    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Connection conn = createDatabase();
            populateDatabase( conn );

            conn.release();

            Demo.display( "CreateSales completed successfully" );

        } catch( ULjException exc ) {
            Demo.displayExceptionFull( exc );
        }
    }

    /** Add an invoice row.
     * @param conn connection to Database
     * @param inv_no invoice number
     * @param name name to whom invoice was sent
     */
    private static void addInvoice( PreparedStatement ri, int inv_no, String
name
                                  , String date )
        throws ULjException
    {
        ri.set( ORDINAL_INVOICE_INV_NO, inv_no );
        ri.set( ORDINAL_INVOICE_NAME, name );
        ri.set( ORDINAL_INVOICE_DATE, date );
        ri.execute();
    }

    /** Add an invoice-item row.
     * @param conn connection to Database
     * @param inv_no invoice number
     * @param item_no line number for item
     * @param prod_no product number sold
     * @param quantity quantity sold
```

```
      * @param price price of one item
      */
    private static void addInvoiceItem( PreparedStatement ri, int inv_no, int
item_no
                                        , int prod_no, int quantity, String
price )
        throws ULjException
    {
        ri.set( ORDINAL_INV_ITEM_INV_NO, inv_no );
        ri.set( ORDINAL_INV_ITEM_ITEM_NO, item_no );
        ri.set( ORDINAL_INV_ITEM_PROD_NO, prod_no );
        ri.set( ORDINAL_INV_ITEM_QUANTITY, quantity );
        ri.set( ORDINAL_INV_ITEM_PRICE, price );
        ri.execute();
    }

    /** Add a product row.
     * @param conn connection to Database
     * @param prod_no product number
     * @param prod_name product name
     * @param price selling price
     */
    private static void addProduct( PreparedStatement ri, int prod_no, String
prod_name, String price )
        throws ULjException
    {
        ri.set( ORDINAL_PROD_NO, prod_no );
        ri.set( ORDINAL_PROD_NAME, prod_name );
        ri.set( ORDINAL_PROD_PRICE, price );
        ri.execute();
    }
}
```

示例：集合和分组

本示例演示 UltraLiteJ 对结果集合的支持。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/** Create a sales report to illustrate aggregation support.
 */
public class SalesReport
{
    /** Mainline.
     * @param args program arguments (not used)
     */
    public static void main( String[] args )
    {
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Sales.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement stmt = conn.prepareStatement(
                "SELECT inv_no, SUM( quantity * price ) AS total"
                + " FROM InvoiceItem"
                + " GROUP BY inv_no ORDER BY inv_no"
            );
            ResultSet agg_cursor = stmt.executeQuery();
            for( ; agg_cursor.next(); ) {
                int inv_no = agg_cursor.getInt( 1 /* "inv_no" */ );
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
                String total = agg_cursor.getString( 2 /* "total" */ );
                Demo.display( Integer.toString( inv_no ) + ' ' + total );
            }
            Demo.display( "SalesReport completed successfully" );
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

示例：以其它顺序检索行

本示例演示 UltraLiteJ 对以其它顺序来处理行的支持。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;

public class SortTransactions
{
    /** Mainline.
     * @param args program arguments (not used)
     */
    public static void main( String[] args )
    {
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Sales.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement stmt = conn.prepareStatement(
                "SELECT inv_no, prod_no, quantity FROM InvoiceItem"
                + " ORDER BY prod_no"
            );
            ResultSet cursor = stmt.executeQuery();
            for( ; cursor.next(); ) {
                /* Can't access columns by name because no meta data */
                int inv_no = cursor.getInt( 1 /* "inv_no" */ );
                int prod_no = cursor.getInt( 2 /* "prod_no" */ );
                int quantity = cursor.getInt( 3 /* "quantity" */ );
                Demo.display( Integer.toString( prod_no ) + ' '
                    + Integer.toString( inv_no ) + ' '
                    + Integer.toString( quantity )
                );
            }
            conn.release();
            Demo.display( "SortTransactions completed successfully" );
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

示例：修改表定义

本示例演示如何更改表定义。在本方案中，对 Invoice 表进行修改，使其中一列的长度从 50 个字符扩展到 100 个字符。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/** Reorganize the Invoice table to have a name with an increased size
 *
 * <p>This shows a possible strategy which can be used to reorganize
 * tables, since UltraLiteJ has no table-altering API.
 * <p>The (contrived) example expands the name column to 100 characters.
 *
 */
public class Reorg
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Sales.ulj" );
            Connection conn = DatabaseManager.connect( config );
            createNewInvoiceTable( conn );
            copyInvoicesToNewTable( conn );
            deleteOldInvoicesTable( conn );
            renameNewInvoicesTable( conn );
            enableSynchronizationForNewTable( conn );
            conn.release();
        } catch( ULjException exc ) {
            Demo.displayExceptionFull( exc );
        }
    }

    private static void createNewInvoiceTable( Connection conn )
        throws ULjException
    {
        conn.schemaCreateBegin();
        TableSchema table_schema = conn.createTable( "NewInvoice" );
        table_schema.createColumn( "inv_no", Domain.INTEGER );
       table_schema.createColumn( "name", Domain.VARCHAR, 100 );   // was 50
in old table
        table_schema.createColumn( "date", Domain.DATE );
        IndexSchema index_schema =
table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "inv_no", IndexSchema.ASCENDING );
       table_schema.setNoSync( true );        // we don't want to sync inserts
yet
        conn.schemaCreateComplete();
    }

    private static void copyInvoicesToNewTable( Connection conn )
        throws ULjException
    {
        PreparedStatement inserter = conn.prepareStatement(
            "INSERT INTO NewInvoice( inv_no, name, \"date\" )
VALUES( ?, ?, ? )"
        );
        int ordinal_inv_no = 1;
        int ordinal_inv_name = 2;
        int ordinal_inv_date = 3;
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
        PreparedStatement stmt = conn.prepareStatement(
            "SELECT inv_no, name, \"date\" FROM Invoice"
        );
        ResultSet cursor = stmt.executeQuery();
        for( ; cursor.next(); ) {
            inserter.set( ordinal_inv_no, cursor.getInt( ordinal_inv_no ) );
            inserter.set( ordinal_inv_name,
cursor.getString( ordinal_inv_name ) );
            inserter.set( ordinal_inv_date,
cursor.getString( ordinal_inv_date ) );
            inserter.execute();
            // in memory-low conditions, we could delete the row from the old
table (specify
            // stopSynchronizationDelete, so these deletes would not
synchronize.
        }
        inserter.close();
        cursor.close();
        stmt.close();
        conn.commit();
    }

    private static void deleteOldInvoicesTable( Connection conn )
        throws ULjException
    {
        conn.dropTable( "Invoice" );
    }

    private static void renameNewInvoicesTable( Connection conn )
        throws ULjException
    {
        conn.renameTable( "NewInvoice", "Invoice" );
    }

    private static void enableSynchronizationForNewTable( Connection conn )
        throws ULjException
    {
        conn.enableSynchronization( "Invoice" );
    }
}
```

示例：对数据进行模糊处理

本示例演示对数据库中数据进行模糊处理的方法。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * Obfuscate -- sample program to a possible obfuscation of the database.
 *
 * Obfuscation is not very good encryption.  It merely makes the data
unreadable
 * with a file dumping program.  The original data can be probably recovered
by
 * someone with knowledge of the algorithms used.
 */
public class Obfuscate
{
    /** Create the database.
     * @return connection for a new database
```

```
     */
    private static Connection createDatabase()
        throws ULjException
    {
        ConfigPersistent config =
DatabaseManager.createConfigurationFile( "Obfuscate.ulj" );
        config.setEncryption( new Obfuscator() );
        Connection conn = DatabaseManager.createDatabase( config );

        conn.schemaCreateBegin();

        TableSchema table_schema = conn.createTable( "Product" );
        table_schema.createColumn( "prod_no", Domain.INTEGER );
        table_schema.createColumn( "prod_name", Domain.VARCHAR, 32 );
        table_schema.createColumn( "price", Domain.NUMERIC, 9, (short)2 );
        IndexSchema index_schema =
table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "prod_no", IndexSchema.ASCENDING );

        conn.schemaCreateComplete();

        return conn;
    }

    /** Add a product row.
     * @param ri PreparedStatement for the Product table
     * @param prod_no product number
     * @param prod_name product name
     * @param price selling price
     */
    private static void addProduct( PreparedStatement ri, int prod_no, String
prod_name, String price )
        throws ULjException
    {
        ri.set( "prod_no", prod_no );
        ri.set( "prod_name", prod_name );
        ri.set( "price", price );
        ri.execute();
    }

    /** Populate the database.
     * @param conn connection to database
     */
    private static void populate( Connection conn )
        throws ULjException
    {
        PreparedStatement ri = conn.prepareStatement(
            "INSERT INTO Product( prod_no, prod_name, price )"
            + " VALUES( :prod_no, :prod_name, :price )"
        );
        addProduct( ri, 2001, "blue screw", ".03" );
        addProduct( ri, 2002, "red screw", ".09" );
        addProduct( ri, 2004, "hammer", "23.99" );
        addProduct( ri, 2005, "vise", "39.99" );
        ri.close();
        conn.commit();
    }

    /** Display contents of Product table.
     * @param conn connection to database
     */
    private static void displayProducts( Connection conn )
        throws ULjException
    {
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
        PreparedStatement stmt = conn.prepareStatement(
            "SELECT prod_no, prod_name, price FROM Product"
            + " ORDER BY prod_no"
        );
        ResultSet cursor = stmt.executeQuery();
        for( ; cursor.next(); ) {
            String prod_no = cursor.getString( 1 /* "prod_no" */ );
            String prod_name = cursor.getString( 2 /* "prod_name" */ );
            String price = cursor.getString( 3 /* "price" */ );
            Demo.display( prod_no + " " + prod_name + " " + price );
        }
        cursor.close();
        stmt.close();
    }

    /** mainline for program.
     * @param args command-line arguments (not used)
     */
    public static void main
        ( String[] args )
    {
        try {
            Connection conn = createDatabase();
            populate( conn );
            displayProducts( conn );
            conn.release();
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }

    /** Class to implement encryption/decryption of the database.
     */
    static class Obfuscator
        implements EncryptionControl
    {
        /** seed for obfuscator         */      private int _seed;

        /** (un)Obfuscate a page.
         * @param page_no the number of the page being encrypted
         * @param src the encrypted source page which was read from the
database
         * @param tgt the unencrypted page (filled in by method)
         */
        private void transform( int page_no, byte[] src, byte[] tgt )
        {
            int seed = ( _seed + page_no ) % 256;
            for( int i = 0; i < src.length; ++i ) {
                tgt[ i ] = (byte)( seed ^ src[ i ] );
                seed = ( seed + 93 ) % 256;
            }
        }

        /** Encrypt a page stored in the database.
         * @param page_no the number of the page being encrypted
         * @param src the encrypted source page which was read from the
database
         * @param tgt the unencrypted page (filled in by method)
         */
        public void decrypt( int page_no, byte[] src, byte[] tgt )
            throws ULjException
        {
            transform( page_no, src, tgt );
        }
```

```
        /** Encrypt a page stored in the database.
         * @param page_no the number of the page being encrypted
         * @param src the unencrypted source
         * @param tgt the encrypted target page which will be written to the
database (filled in by method)
         */
        public void encrypt( int page_no, byte[] src, byte[] tgt )
            throws ULjException
        {
            transform( page_no, src, tgt );
        }

        /** Initialize the encryption control with a password.
         * @param password the password
         */
        public void initialize( String password )
            throws ULjException
        {
            byte[] bytes = null;
            try {
                bytes = password.getBytes( "UTF8" );
            } catch( Exception e ) {
                Demo.display( "Encryption initialization failure" );
                throw new ObfuscationError();
            }
            _seed = 0;
            for( int i = bytes.length; i > 0; ) {
                _seed ^= bytes[ --i ];
            }
        }
    }

    /** Error class for encryption errors.
     */
    static class ObfuscationError
        extends ULjException
    {
        /** Constructor.
         */
        ObfuscationError()
        {
            super( "Obfuscation Error" );
        }

        /**
         * Get the error code, associated with this exception.
         * @return the error code (from the list at the top of this class)
associated
         *  with this exception
         */
        public int getErrorCode()
        {
            return ULjException.SQLE_ERROR;
        }

        /** Get exception causing this exception, if it exists.
        * @return null, if there exists no causing exception; otherwise, the
exception causing this exception
         */
        public ULjException getCausingException()
        {
            return null;
        }
```

```
        /** Get offset of error within a SQL string.
         * @return (-1) when there is no SQL string associated with the error
message; otherwise,
         * the (base 0) offset within that string where the error occurred.
         */
        public int getSqlOffset()
        {
            return -1;
        }
    }
}
```

示例：对数据进行加密

本示例演示对数据库中数据进行加密的方法。在本方案中，对数据进行解密会导致性能下降。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
/**
 * Encrypted -- sample program to demonstrate encryption of database.
 *
 * This sample requires either the Sun JDK 1.4.2 (or later) or a freeware
 * version of the Java encryption classes (JCE).
 */
public class Encrypted
{
    /** Create the database.
     * @return connection for a new database
     */
    private static Connection createDatabase()
        throws ULjException
    {
        ConfigPersistent config =
DatabaseManager.createConfigurationFile( "Encrypt.ulj" );
        config.setEncryption( new Encryptor() );
        Connection conn = DatabaseManager.createDatabase( config );

        conn.schemaCreateBegin();

        TableSchema table_schema = conn.createTable( "Product" );
        table_schema.createColumn( "prod_no", Domain.INTEGER );
        table_schema.createColumn( "prod_name", Domain.VARCHAR, 32 );
        table_schema.createColumn( "price", Domain.NUMERIC, 9, (short)2 );
        IndexSchema index_schema =
table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "prod_no", IndexSchema.ASCENDING );

        conn.schemaCreateComplete();

        return conn;
    }

    /** Add a product row.
     * @param ri PreparedStatement for the Product table
     * @param prod_no product number
     * @param prod_name product name
```

```
     * @param price selling price
     */
    private static void addProduct( PreparedStatement ri, int prod_no, String
prod_name, String price )
        throws ULjException
    {
        ri.set( "prod_no", prod_no );
        ri.set( "prod_name", prod_name );
        ri.set( "price", price );
        ri.execute();
    }

    /** Populate the database.
     * @param conn connection to database
     */
    private static void populate( Connection conn )
        throws ULjException
    {
        PreparedStatement ri = conn.prepareStatement(
            "INSERT INTO Product( prod_no, prod_name, price )"
            + " VALUES( :prod_no, :prod_name, :price )"
        );
        addProduct( ri, 2001, "blue screw", ".03" );
        addProduct( ri, 2002, "red screw", ".09" );
        addProduct( ri, 2004, "hammer", "23.99" );
        addProduct( ri, 2005, "vise", "39.99" );
        ri.close();
        conn.commit();
    }

    /** Display contents of Product table.
     * @param conn connection to database
     */
    private static void displayProducts( Connection conn )
        throws ULjException
    {
        PreparedStatement stmt = conn.prepareStatement(
            "SELECT prod_no, prod_name, price FROM Product"
            + " ORDER BY prod_no"
        );
        ResultSet cursor = stmt.executeQuery();
        for( ; cursor.next(); ) {
            /* Can't access columns by name because no meta data */
            String prod_no = cursor.getString( 1 /* "prod_no" */ );
            String prod_name = cursor.getString( 2 /* "prod_name" */ );
            String price = cursor.getString( 3 /* "price" */ );
            Demo.display( prod_no + " " + prod_name + " " + price );
        }
        cursor.close();
        stmt.close();
    }

    /** mainline for program.
     * @param args command-line arguments (not used)
     */
    public static void main
        ( String[] args )
    {
        try {
            Connection conn = createDatabase();
            populate( conn );
            displayProducts( conn );
            conn.release();
        } catch( ULjException exc ) {
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
                Demo.displayException( exc );
            }
        }

        /** Class to implement encryption/decryption of the database.
         */
        static class Encryptor
            implements EncryptionControl
        {
            private SecretKeySpec _key = null;
            private Cipher _cipher = null;

            /** Encrypt a page stored in the Database.
             * @param page_no the number of the page being encrypted
             * @param src the encrypted source page which was read from the
Database
             * @param tgt the unencrypted page (filled in by method)
             */
            public void decrypt( int page_no, byte[] src, byte[] tgt )
                throws ULjException
            {
                byte[] decrypted = null;
                try {
                    _cipher.init( Cipher.DECRYPT_MODE, _key );
                    decrypted = _cipher.doFinal( src );
                } catch( Exception e ) {
                    Demo.display( "Error: decrypting" );
                    throw new EncryptionError();
                }
                for( int i = tgt.length; i > 0; ) {
                    --i;
                    tgt[ i ] = decrypted[ i ];
                }
            }

            /** Encrypt a page stored in the Database.
             * @param page_no the number of the page being encrypted
             * @param src the unencrypted source
             * @param tgt the encrypted target page which will be written to the
Database (filled in by method)
             */
            public void encrypt( int page_no, byte[] src, byte[] tgt )
                throws ULjException
            {
                byte[] encrypted = null;
                try {
                    _cipher.init( Cipher.ENCRYPT_MODE, _key );
                    encrypted = _cipher.doFinal( src );
                } catch( Exception e ) {
                    Demo.display( "Error: encrypting" );
                    throw new EncryptionError();
                }
                for( int i = tgt.length; i > 0; ) {
                    --i;
                    tgt[ i ] = encrypted[ i ];
                }
            }

            /** Initialize the encryption control with a password.
             * @param password the password
             */
            public void initialize( String password )
                throws ULjException
            {
```

```
                try {
                    byte[] bytes = password.getBytes( "UTF8" );
                    MessageDigest md = MessageDigest.getInstance( "SHA" );
                    bytes = md.digest( bytes );
                    byte[] key_bytes = new byte[16];
                    for( int i = key_bytes.length; i > 0; ) {
                        --i;
                        key_bytes[ i ] = bytes[ i ];
                    }
                    _key = new SecretKeySpec( key_bytes, "AES" );
                    _cipher = Cipher.getInstance( "AES/ECB/NoPadding" );
                } catch( Exception e ) {
                    Demo.display( "Error: initializing encryption" );
                    throw new EncryptionError();
                }
            }
        }

        /** Error class for encryption errors.
         */
        static class EncryptionError
            extends ULjException
        {
            /** Constructor.
             */
            EncryptionError()
            {
                super( "Encryption Error" );
            }

            /**
             * Get the error code, associated with this exception.
             * @return the error code (from the list at the top of this class)
associated
             *  with this exception
             */
            public int getErrorCode()
            {
                return ULjException.SQLE_ERROR;
            }

            /** Get exception causing this exception, if it exists.
             * @return null, if there exists no causing exception; otherwise, the
exception causing this exception
             */
            public ULjException getCausingException()
            {
                return null;
            }

            /** Get offset of error within a SQL string.
             * @return (-1) when there is no SQL string associated with the error
message; otherwise,
             * the (base 0) offset within that string where the error occurred.
             */
            public int getSqlOffset()
            {
                return -1;
            }
        }
    }
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

示例：显示数据库模式信息

此示例说明如何导航 UltraLiteJ 数据库的系统表以检查模式信息。也显示表的每一行的数据。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;

/** Sample program to dump schema of a database.
 * This sample extracts schema information into a set of data structures
 * (TableArray, OptionArray) before dumping out the meta data so as to
 * provide for future schema information lookup.
 */
public class DumpSchema
{
    /** Mainline.
     * @param args program arguments (not used)
     */
    public static void main( String[] args )
    {
        try {
            Configuration config =
DatabaseManager.createConfigurationFile( "Sales.ulj" );
            Connection conn = DatabaseManager.connect( config );

            Demo.display(
                    TableSchema.SYS_TABLES
                    + " table_flags are:\nTableSchema.TABLE_IS_SYSTEM(0x"
                    + Integer.toHexString( ((int)TABLE_FLAG_SYSTEM) &
0xffff )
                    + "),\nTableSchema.TABLE_IS_NOSYNC(0x"
                    + Integer.toHexString( ((int)TABLE_FLAG_NO_SYNC) &
0xffff )
                    + ")"
                );
            getSchema( conn );
            dumpSchema( conn );

        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }

    // Some constants for metadata
    private static String SQL_SELECT_TABLE_COLS =
            "SELECT T.table_id, T.table_name, T.table_flags,"
            + " C.column_id, C.column_name, C.column_flags,"
            + " C.column_domain, C.column_length, C.column_default"
            + " FROM " + TableSchema.SYS_TABLES + " T"
            + " JOIN " + TableSchema.SYS_COLUMNS + " C"
            + " ON T.table_id = C.table_id"
            + " ORDER BY T.table_id"
            ;
    private static final int TABLE_ID = 1;
    private static final int TABLE_NAME = 2;
    private static final int TABLE_FLAGS = 3;
    private static final int COLUMN_ID = 4;
    private static final int COLUMN_NAME = 5;
    private static final int COLUMN_FLAGS = 6;
    private static final int COLUMN_DOMAIN_TYPE = 7;
    private static final int COLUMN_DOMAIN_LENGTH = 8;
    private static final int COLUMN_DEFAULT = 9;
```

```
    private static final int TABLE_FLAG_SYSTEM = TableSchema.TABLE_IS_SYSTEM;
    private static final int TABLE_FLAG_NO_SYNC =
TableSchema.TABLE_IS_NOSYNC;

    private static final int COLUMN_FLAG_IN_PRIMARY_INDEX = 0x01;
    private static final int COLUMN_FLAG_IS_NULLABLE = 0x02;

    private static String SQL_SELECT_INDEX_COLS =
            "SELECT I.table_id, I.index_id, I.index_name, I.index_flags,"
            + " X.\"order\", X.column_id, X.index_column_flags"
            + " FROM " + TableSchema.SYS_INDEX_COLUMNS + " X"
            + " JOIN " + TableSchema.SYS_INDEXES + " I"
            + " ON I.table_id = X.table_id AND I.index_id = X.index_id"
            + " ORDER BY X.table_id, X.index_id, X.\"order\""
            ;
    private static final int INDEX_TABLE_ID = 1;
    private static final int INDEX_ID = 2;
    private static final int INDEX_NAME = 3;
    private static final int INDEX_FLAGS = 4;
    private static final int INDEX_COLUMN_ORDER = 5;
    private static final int INDEX_COLUMN_COLUMN_ID = 6;
    private static final int INDEX_COLUMN_FLAGS = 7;

    private static final int INDEX_COLUMN_FLAG_FORWARD = 1;

    private static final int INDEX_FLAG_UNIQUE_KEY = 0x01;
    private static final int INDEX_FLAG_UNIQUE_INDEX = 0x02;
    private static final int INDEX_FLAG_PERSISTENT = 0x04;
    private static final int INDEX_FLAG_PRIMARY_INDEX = 0x08;

    private static String SQL_SELECT_OPTIONS =
            "SELECT name, value FROM " + TableSchema.SYS_INTERNAL
            + " ORDER BY name"
            ;
    private static final int OPTION_NAME = 1;
    private static final int OPTION_VALUE = 2;

    // Metadata:
    private static TableArray tables = new TableArray();
    private static OptionArray options = new OptionArray();

    /**
     * Extracts the schema of a database
     */
    private static void getSchema( Connection conn ) throws ULjException
    {
        PreparedStatement stmt = conn.prepareStatement(
                SQL_SELECT_TABLE_COLS
            );
        ResultSet cursor = stmt.executeQuery();
        Table table = null;
        int last_table_id = -1;
        for( ; cursor.next(); ) {
            int table_id = cursor.getInt( TABLE_ID );
            if( table_id != last_table_id ) {
                String table_name = cursor.getString( TABLE_NAME );
                int table_flags = cursor.getInt( TABLE_FLAGS );
                table = new Table( table_id, table_name, table_flags );
                tables.append( table );
                last_table_id = table_id;
            }
            int column_id = cursor.getInt( COLUMN_ID );
            String column_name = cursor.getString( COLUMN_NAME );
            int column_flags = cursor.getInt( COLUMN_FLAGS );
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
                int column_domain = cursor.getInt( COLUMN_DOMAIN_TYPE );
                int column_length = cursor.getInt( COLUMN_DOMAIN_LENGTH );
                int column_default = cursor.getInt( COLUMN_DEFAULT );
                Column column = new Column(
                        conn, column_id, column_name, column_flags,
                        column_domain, column_length, column_default
                    );
                table.addColumn( column );
            }
            cursor.close();
            stmt.close();

            // read indexes
            stmt = conn.prepareStatement( SQL_SELECT_INDEX_COLS );
            cursor = stmt.executeQuery();
            int last_index_id = -1;
            Index index = null;
            last_table_id = -1;
            for( ; cursor.next(); ) {
                int table_id = cursor.getInt( INDEX_TABLE_ID );
                int index_id = cursor.getInt( INDEX_ID );
                if( last_table_id != table_id || last_index_id != index_id ) {
                    String index_name = cursor.getString( INDEX_NAME );
                    int index_flags = cursor.getInt( INDEX_FLAGS );
                    index = new Index( index_id, index_name, index_flags );
                    table = findTable( table_id );
                    table.addIndex( index );
                    last_index_id = index_id;
                    last_table_id = table_id;
                }
                int order = cursor.getInt( INDEX_COLUMN_ORDER );
                int column_id = cursor.getInt( INDEX_COLUMN_COLUMN_ID );
                int index_column_flags = cursor.getInt( INDEX_COLUMN_FLAGS );
                IndexColumn index_column = new IndexColumn( order, column_id,
index_column_flags );
                index.addColumn( index_column );
            }
            cursor.close();
            stmt.close();

            // read database options
            stmt = conn.prepareStatement( SQL_SELECT_OPTIONS );
            cursor = stmt.executeQuery();
            for( ; cursor.next(); ) {
                String option_name = cursor.getString( OPTION_NAME );
                String option_value = cursor.getString( OPTION_VALUE );
                Option option = new Option( option_name, option_value );
                options.append( option );
            }
            cursor.close();
            stmt.close();
        }

        /** Dump the schema of a database
         */
        private static void dumpSchema( Connection conn ) throws ULjException
        {
            // Display the metadata options
            Demo.display( "\nMetadata options:\n" );
            for( int opt_no = 0; opt_no < options.count(); ++ opt_no ) {
                Option option = options.elementAt( opt_no );
                option.display();
            }
            // Display the metadata tables
```

```
        Demo.display( "\nMetadata tables:" );
        for( int table_no = 0; table_no < tables.count(); ++ table_no ) {
            Table table = tables.elementAt( table_no );
            table.display( table_no );
        }
        // Display the rows for non-system tables.
        for( int table_no = 0; table_no < tables.count(); ++ table_no ) {
            Table table = tables.elementAt( table_no );
            if( 0 == ( table.getFlags() & TABLE_FLAG_SYSTEM ) ) {
                Demo.display( "\nRows for table: ", table.getName(), "\n" );
                Index index = table.getIndex( 0 );
                PreparedStatement stmt = conn.prepareStatement(
                        "SELECT * FROM \"" + table.getName() + "\""
                    );
                ResultSet cursor = stmt.executeQuery();
                int column_count = table.getColumnCount();
                int row_count = 0;
                for( ; cursor.next(); ) {
                    StringBuffer buf = new StringBuffer();
                    buf.append( "Row[" );
                    buf.append( Integer.toString( ++row_count ) );
                    buf.append( "]:" );
                    char joiner = ' ';
                    for( int col_no = 1; col_no <= column_count; ++col_no ) {
                        String value = cursor.isNull( col_no )
                                     ? "<NULL>"
                                     : cursor.getString( col_no );
                        buf.append( joiner );
                        buf.append( value );
                        joiner = ',';
                    }
                    Demo.display( buf.toString() );
                }
                cursor.close();
                stmt.close();
            }
        }
    }

    /** Find a table.
     */
    private static Table findTable( int table_id )
    {
        Table retn = null;
        for( int i = tables.count(); i > 0; ) {
            Table table = tables.elementAt( --i );
            if( table_id == table.getId() ) {
                retn = table;
                break;
            }
        }
        return retn;
    }

    /** Representation of a column.
     */
    private static class Column
    {
        private int _column_id;
        private String _column_name;
        private int _column_flags;
        private Domain _domain;
        private int _column_default;
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
        Column( Connection conn, int column_id, String column_name, int
column_flags, int column_domain, int column_length, int column_default )
            throws ULjException
        {
            _column_id = column_id;
            _column_name = column_name;
            _column_flags = column_flags;
            _column_default = column_default;
            int scale = 0;
            switch( column_domain ) {
              case Domain.NUMERIC :
                scale = column_length >> 8;
                column_length &= 255;
                _domain = conn.createDomain( Domain.NUMERIC, column_length,
scale );
                break;
              case Domain.VARCHAR :
              case Domain.BINARY :
                _domain = conn.createDomain( column_domain, column_length );
                break;
              default :
                _domain = conn.createDomain( column_domain );
                break;
            }
        }

        String getName()
        {
            return _column_name;
        }

        void display()
        {
            StringBuffer buf = new StringBuffer();
            buf.append( "  \"" );
            buf.append( _column_name );
            buf.append( "\"\t" );
            buf.append( _domain.getName() );
            switch( _domain.getType() ) {
              case Domain.NUMERIC :
                buf.append( '(' );
                buf.append( Integer.toString( _domain.getPrecision() ) );
                buf.append( ',' );
                buf.append( Integer.toString( _domain.getScale() ) );
                buf.append( ')' );
                break;
              case Domain.VARCHAR :
              case Domain.BINARY :
                buf.append( '(' );
                buf.append( Integer.toString( _domain.getSize() ) );
                buf.append( ')' );
                break;
            }
            if( 0 != ( _column_flags & COLUMN_FLAG_IS_NULLABLE ) ) {
                buf.append( " NULL" );
            } else {
                buf.append( " NOT NULL" );
            }
            switch( _column_default ) {
                case ColumnSchema.COLUMN_DEFAULT_NONE:
                default:
                    break;
                case ColumnSchema.COLUMN_DEFAULT_AUTOINC:
                    buf.append( " DEFAULT AUTOINCREMENT" );
```

```
                        break;
                    case ColumnSchema.COLUMN_DEFAULT_GLOBAL_AUTOINC:
                        buf.append( " DEFAULT GLOBAL AUTOINCREMENT" );
                        break;
                    case ColumnSchema.COLUMN_DEFAULT_CURRENT_DATE:
                        buf.append( " DEFAULT CURRENT DATE" );
                        break;
                    case ColumnSchema.COLUMN_DEFAULT_CURRENT_TIME:
                        buf.append( " DEFAULT CURRENT TIME" );
                        break;
                    case ColumnSchema.COLUMN_DEFAULT_CURRENT_TIMESTAMP:
                        buf.append( " DEFAULT CURRENT TIMESTAMP" );
                        break;
                    case ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID:
                        buf.append( " DEFAULT NEWID()" );
                        break;
                }
                buf.append( ", /* column_id=" );
                buf.append( Integer.toString( _column_id ) );
                buf.append( " column_flags=" );
                int c = 0;
                if( 0 != ( _column_flags & COLUMN_FLAG_IN_PRIMARY_INDEX ) ) {
                    buf.append( "IN_PRIMARY_INDEX" );
                    c++;
                }
                if( 0 != ( _column_flags & COLUMN_FLAG_IS_NULLABLE ) ) {
                    if( c > 0 ) {
                        buf.append( "," );
                    }
                    buf.append( "NULLABLE" );
                }
                buf.append( " */" );
                Demo.display( buf.toString() );
            }
        }

        /** Representation of Index schema.
         */
        private static class Index
        {
            private String _index_name;
            private int _index_id;
            private int _index_flags;
            private IndexColumnArray _columns;

            Index( int index_id, String index_name, int index_flags )
            {
                _index_id = index_id;
                _index_name = index_name;
                _index_flags = index_flags;
                _columns = new IndexColumnArray();
            }

            void addColumn( IndexColumn column )
            {
                _columns.append( column );
            }

            void display( Table table, boolean constraints )
            {
                StringBuffer buf = new StringBuffer();
                String flags = "";
                String indent = "  ";
                if( 0 != ( _index_flags & INDEX_FLAG_PRIMARY_INDEX ) ) {
```

```
                if( !constraints )      return;
                buf.append( "  CONSTRAINT \"" );
                buf.append( _index_name );
                buf.append( "\" PRIMARY KEY (" );
                flags = "PRIMARY_KEY,UNIQUE_KEY";
            } else if( 0 != ( _index_flags & INDEX_FLAG_UNIQUE_KEY ) ) {
                if( !constraints )      return;
                buf.append( "  CONSTRAINT \"" );
                buf.append( _index_name );
                buf.append( "\" UNIQUE (" );
                flags = "UNIQUE_KEY";
            } else {
                if( constraints )       return;
                if( 0 != ( _index_flags & INDEX_FLAG_UNIQUE_INDEX ) ) {
                    buf.append( "UNIQUE " );
                    flags = "UNIQUE_INDEX";
                }
                indent = "";
                buf.append( "INDEX \"" );
                buf.append( _index_name );
                buf.append( "\" ON \"" );
                buf.append( table.getName() );
                buf.append( "\" (" );
            }
            buf.append( "\n" );
            buf.append( indent );
            buf.append( "  /* index_id=" );
            buf.append( Integer.toString( _index_id ) );
            buf.append( " index_flags=" );
            buf.append( flags );
            if( 0 != ( _index_flags & INDEX_FLAG_PERSISTENT ) ) {
                buf.append( ",PERSISTENT" );
            }
            buf.append( " */" );
            Demo.display( buf.toString() );
            int bounds = _columns.count();
            for( int col_no = 0; col_no < bounds; ++ col_no ) {
                IndexColumn column = _columns.elementAt( col_no );
                column.display( table, indent, col_no + 1 < bounds );
            }
            Demo.display( indent + ")" );
        }

        String getName()
        {
            return _index_name;
        }
    }

    /** Representation of IndexColumn schema.
     */
    private static class IndexColumn
    {
        private int _index_column_id;
        private int _index_column_column_id;
        private int _index_column_flags;

        IndexColumn( int index_column_id, int index_column_column_id, int
index_column_flags )
        {
            _index_column_id = index_column_id;
            _index_column_column_id = index_column_column_id;
            _index_column_flags = index_column_flags;
        }
```

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
        void display( Table table, String indent, boolean notlast )
        {
            StringBuffer buf = new StringBuffer( indent );
            buf.append( "  \"" );
            Column column = table.getColumn( _index_column_column_id );
            buf.append( column.getName() );
            if( 0 != ( _index_column_flags & INDEX_COLUMN_FLAG_FORWARD ) ) {
                buf.append( "\" ASC" );
            } else {
                buf.append( "\" DESC" );
            }
            if( notlast ) {
                buf.append( "," );
            }
            Demo.display( buf.toString() );
        }
    }

    /** Representation of a database Option.
     */
    private static class Option
    {
        private String _option_name;
        private String _option_value;

        Option( String name, String value )
        {
            _option_name = name;
            _option_value = value;
        }

        void display()
        {
            StringBuffer buf = new StringBuffer();
            buf.append( "Option[ " );
            buf.append( _option_name );
            buf.append( " ] = '" );
            buf.append( _option_value );
            buf.append( "'" );
            Demo.display( buf.toString() );
        }

    }

    /** Representation of Table schema.
     */
    private static class Table
    {
        private String _table_name;
        private int _table_id;
        private int _table_flags;
        private ColumnArray _columns;
        private IndexArray _indexes;

        Table( int table_id, String table_name, int table_flags )
        {
            _table_name = table_name;
            _table_id = table_id;
            _table_flags = table_flags;
            _columns = new ColumnArray();
            _indexes = new IndexArray();
        }
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
void addColumn( Column column )
{
    _columns.append( column );
}

void addIndex( Index index )
{
    _indexes.append( index );
}

Column getColumn( int id )
{
    return _columns.elementAt( id );
}

int getColumnCount()
{
    return _columns.count();
}

int getFlags()
{
    return _table_flags;
}

Index getIndex( int id )
{
    return _indexes.elementAt( id );
}

String getName()
{
    return _table_name;
}

void display( int logical_number )
{
    StringBuffer str = new StringBuffer();
    str.append( "\nTABLE \"" );
    str.append( _table_name );
    str.append( "\" /* table_id=" );
    str.append( Integer.toString( _table_id ) );
    str.append( " table_flags=" );
    if( 0 == _table_flags ) {
        str.append( "0" );
    } else {
        int c = 0;
        if( 0 != ( _table_flags & TABLE_FLAG_SYSTEM ) ) {
            str.append( "SYSTEM" );
            c++;
        }
        if( 0 != ( _table_flags & TABLE_FLAG_NO_SYNC ) ) {
            if( c > 0 ) {
                str.append( "," );
            }
            str.append( "NO_SYNC" );
            c++;
        }
    }
    str.append( " */ (" );
    Demo.display( str.toString() );
    int bound = _columns.count();
    for( int col_no = 0; col_no < bound; ++col_no ) {
        Column column = _columns.elementAt( col_no );
```

```
                column.display();
            }
            bound = _indexes.count();
            for( int idx_no = 0; idx_no < bound; ++idx_no ) {
                Index index = _indexes.elementAt( idx_no );
                index.display( this, true );
            }
            Demo.display( ")" );
            for( int idx_no = 0; idx_no < bound; ++idx_no ) {
                Index index = _indexes.elementAt( idx_no );
                index.display( this, false );
            }
        }

        int getId()
        {
            return _table_id;
        }
    }

    /** Simple adjustable array of objects.
     */
    private static class ObjArray
    {
        private Object[] _array = new Object[ 10 ];
        private int _used = 0;

        void append( Object str )
        {
            if( _used >= _array.length ) {
                Object[] new_array = new Object[ _used * 2 ];
                for( int i = _used; i > 0; ) {
                    --i;
                    new_array[ i ] = _array[ i ];
                }
                _array = new_array;
            }
            _array[ _used++ ] = str;
        }

        int count()
        {
            return _used;
        }

        Object getElementAt( int position )
        {
            return _array[ position ];
        }
    }

    /** Simple adjustable array of Strings.
     */
    private static class StrArray
        extends ObjArray
    {
        String elementAt( int position )
        {
            return (String)getElementAt( position );
        }
    }

    /** Simple adjustable array of Table objects.
     */
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
    private static class TableArray
        extends ObjArray
    {
        Table elementAt( int position )
        {
            return (Table)getElementAt( position );
        }
    }

    /** Simple adjustable array of Column objects.
     */
    private static class ColumnArray
        extends ObjArray
    {
        Column elementAt( int position )
        {
            return (Column)getElementAt( position );
        }
    }

    /** Simple adjustable array of Index objects.
     */
    private static class IndexArray
        extends ObjArray
    {
        Index elementAt( int position )
        {
            return (Index)getElementAt( position );
        }
    }

    /** Simple adjustable array of IndexColumn objects.
     */
    private static class IndexColumnArray
        extends ObjArray
    {
        IndexColumn elementAt( int position )
        {
            return (IndexColumn)getElementAt( position );
        }
    }

    /** Simple adjustable array of Option objects.
     */
    private static class OptionArray
        extends ObjArray
    {
        Option elementAt( int position )
        {
            return (Option)getElementAt( position );
        }
    }

}
```

应用程序的部分输出如下所示。

```
Metadata options:

Option[ date_format ] = 'YYYY-MM-DD'
Option[ date_order ] = 'YMD'
Option[ global_database_id ] = '0'
Option[ nearest_century ] = '50'
```

```
Option[ precision ] = '30'
Option[ scale ] = '6'
Option[ time_format ] = 'HH:NN:SS.SSS'
Option[ timestamp_format ] = 'YYYY-MM-DD HH:NN:SS.SSS'
Option[ timestamp_increment ] = '1'

Metadata tables:

Table[0] name = "systable"  id = 0 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "table_name" flags = 0x0 domain = VARCHAR(128)
  column[2 ]: name = "table_flags" flags = 0x0 domain = UNSIGNED-SHORT
  column[3 ]: name = "table_data" flags = 0x0 domain = INTEGER
  column[4 ]: name = "table_autoinc" flags = 0x0 domain = BIG
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD

Table[1] name = "syscolumn"  id = 1 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "column_id" flags = 0x1,IN-PRIMARY-INDEX domain =
INTEGER
  column[2 ]: name = "column_name" flags = 0x0 domain = VARCHAR(128)
  column[3 ]: name = "column_flags" flags = 0x0 domain = TINY
  column[4 ]: name = "column_domain" flags = 0x0 domain = TINY
  column[5 ]: name = "column_length" flags = 0x0 domain = INTEGER
  column[6 ]: name = "column_default" flags = 0x0 domain = TINY
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "column_id" flags = 0x1,FORWARD

Table[2] name = "sysindex"  id = 2 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "index_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[2 ]: name = "index_name" flags = 0x0 domain = VARCHAR(128)
  column[3 ]: name = "index_flags" flags = 0x0 domain = TINY
  column[4 ]: name = "index_data" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "index_id" flags = 0x1,FORWARD

Table[3] name = "sysindexcolumn"  id = 3 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "index_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[2 ]: name = "order" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[3 ]: name = "column_id" flags = 0x0 domain = INTEGER
  column[4 ]: name = "index_column_flags" flags = 0x0 domain = TINY
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "index_id" flags = 0x1,FORWARD
    key[2 ]: name = "order" flags = 0x1,FORWARD

Table[4] name = "sysinternal"  id = 4 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "name" flags = 0x1,IN-PRIMARY-INDEX domain =
VARCHAR(128)
  column[1 ]: name = "value" flags = 0x0 domain = VARCHAR(128)
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "name" flags = 0x1,FORWARD

Table[5] name = "syspublications"  id = 5 flags = 0xc000,SYSTEM,NO_SYNC
```


Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
  column[0 ]: name = "publication_id" flags = 0x1,IN-PRIMARY-INDEX domain =
INTEGER
  column[1 ]: name = "publication_name" flags = 0x0 domain = VARCHAR(128)
  column[2 ]: name = "download_timestamp" flags = 0x0 domain = TIMESTAMP
  column[3 ]: name = "last_sync_sent" flags = 0x0 domain = INTEGER
  column[4 ]: name = "last_sync_confirmed" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "publication_id" flags = 0x1,FORWARD

Table[6] name = "sysarticles"  id = 6 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "publication_id" flags = 0x1,IN-PRIMARY-INDEX domain =
INTEGER
  column[1 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "publication_id" flags = 0x1,FORWARD
    key[1 ]: name = "table_id" flags = 0x1,FORWARD

Table[7] name = "sysforeignkey"  id = 7 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "foreign_table_id" flags = 0x0 domain = INTEGER
  column[2 ]: name = "foreign_key_id" flags = 0x1,IN-PRIMARY-INDEX domain =
INTEGER
  column[3 ]: name = "name" flags = 0x0 domain = VARCHAR(128)
  column[4 ]: name = "index_name" flags = 0x0 domain = VARCHAR(128)
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "foreign_key_id" flags = 0x1,FORWARD

Table[8] name = "sysfkcol"  id = 8 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "foreign_key_id" flags = 0x1,IN-PRIMARY-INDEX domain =
INTEGER
  column[2 ]: name = "item_no" flags = 0x1,IN-PRIMARY-INDEX domain = SHORT
  column[3 ]: name = "column_id" flags = 0x0 domain = INTEGER
  column[4 ]: name = "foreign_column_id" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-
INDEX,PERSISTENT,PRIMARY-INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "foreign_key_id" flags = 0x1,FORWARD
    key[2 ]: name = "item_no" flags = 0x1,FORWARD
```

教程：构建 BlackBerry 应用程序

目录

UltraLiteJ 开发简介 66
第 1 部分：创建用于 BlackBerry 的 UltraLiteJ 应用程序 67
第 2 部分：向 BlackBerry 应用程序添加同步 76
教程的代码列表 80

UltraLiteJ 开发简介

本教程指导您使用 Research In Motion BlackBerry Java 开发环境为 BlackBerry 智能手机开发 UltraLiteJ 应用程序。在本教程中，您将在 BlackBerry 模拟器上运行该应用程序，并将应用程序部署到物理设备。代码示例贯穿于整个教程，并且在教程结尾处还提供了完整的代码列表。

教程假定您具备以下条件：

- 您熟悉 Java 编程语言。
- 您的计算机上已安装 Research In Motion BlackBerry JDE 4.0 或更高版本。
- 您的计算机上已安装 UltraLiteJ。UltraLiteJ 的缺省安装位置为 *install-dir\UltraLite\UltraLiteJ*。
- 您的计算机上已安装 Research In Motion MDS Services Simulator。第 2 部分需要使用该程序。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

第 1 部分：创建用于 BlackBerry 的 UltraLiteJ 应用程序

本部分介绍如何创建一个在 UltraLiteJ 数据库中维护简单名称列表的 BlackBerry 应用程序。第二部分介绍如何将该应用程序与 MobiLink 服务器进行同步。

第 1 课：创建 BlackBerry JDE 项目

在本课中，将创建一个新的 BlackBerry Java 开发环境（Java Development Environment，简称 JDE）项目。

1. 从 JDE [**File**] 菜单中，选择 [**New Workspace**]。
2. 选择工作区的位置，例如，*c:\tutorials*。将工作区命名为 **HelloBlackBerry**，然后单击 [**OK**]。
3. 在本教程中，该工作区只包含一个项目。从 [**Project**] 菜单中，选择 [**Create New Project**]。
4. 将项目命名为 **HelloBlackBerry**，然后单击 [**OK**]。
5. 将 UltraLiteJ JAR 文件添加到项目中。
    a. 在 [**Workspace**] 窗口中，右击项目，然后选择 [**Properties**]。
    b. 在 [**Build**] 选项卡中，单击 [**Imported Jar Files**] 字段旁的 [**Add**]。
    c. 浏览至 UltraLiteJ 安装程序中的 *UltraLiteJ\J2meRim11\UltraLiteJ.jar* 文件，然后单击 [**Open**]。
    d. 单击 [**OK**] 关闭 [**Properties**] 窗口。
6. 从 [**Project**] 菜单中，选择 [**Set Active Projects**]。选择 HelloBlackBerry 项目，然后单击 [**OK**]。
7. 保存该项目。

第 2 课：显示 BlackBerry 应用程序屏幕

在本课中，将创建带有 main 方法的用于打开 HomeScreen 的类，该 HomeScreen 包含标题和状态消息。

1. 在工作区视图中，右击项目并选择 [**Create New File In Project**]。
2. 在 [**Source File Name**] 框中，键入 **myapp\Application.java** 以创建名为 *Application.java* 的文件，该文件是 myapp 包的一部分。

    单击 [**OK**] 创建文件。Application 类即会出现在 JDE 窗口中。
3. 定义 Application 类。此类不需要导入。添加构造函数和 main 方法，因此 Application 类定义如下：

```
class Application extends net.rim.device.api.ui.UiApplication {
    public static void main( String[] args )
    {
        Application instance = new Application();
        instance.enterEventDispatcher();
    }
```

```
        Application() {
            pushScreen( new HomeScreen() );
        }
    }
```

4. 将 HomeScreen 类添加到项目。

   a. 在工作区视图中，右击项目并选择 [**Create New File In Project**]。

   b. 在 [**Source File Name**] 框中，键入 **myapp\HomeScreen.java**。

   c. 单击 [**OK**] 创建文件。

      HomeScreen 类出现在 JDE 窗口中。

5. 定义 HomeScreen 类，以使其显示标题和状态消息。

```
    package myapp;
    import net.rim.device.api.ui.*;
    import net.rim.device.api.ui.component.*;
    import net.rim.device.api.ui.container.*;
    import java.util.*;

    class HomeScreen extends MainScreen {

        HomeScreen() {

            // Set the window title
            LabelField applicationTitle = new LabelField("Hello BlackBerry");
            setTitle(applicationTitle);

            // Add a label to show application status
            _statusLabel = new LabelField( "Status: Started" );
            add( _statusLabel );
        }
        private LabelField _statusLabel;
    }
```

   将 _statusLabel 定义为类变量，这样以后便可从应用程序的其它部分对其进行访问。

6. 右击该项目然后选择 [**Build**]。确保编译过程没有出错。

7. 按下 F5 在设备模拟器中运行应用程序。

   在单独的窗口中启动 BlackBerry 模拟器。

8. 在模拟器上，导航至 [**Applications**] 窗口，然后选择 HelloBlackBerry 应用程序。

9. 启动应用程序。

   将出现一个窗口，其中显示标题栏 [**Hello BlackBerry**] 和状态行 [**Status: started**]。

10. 从 JDE [**Debug**] 菜单中选择 [**Stop Debugging**]。

    模拟器终止。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

第 3 课：创建 UltraLiteJ 数据库

在本课中，您将编写代码来创建并连接 UltraLiteJ 数据库。用于创建新数据库的代码在名为 DataAccess 的单个类中定义，并从 HomeScreen 构造函数中调用。使用单个类可确保每次仅打开一个数据库连接。而 UltraLiteJ 支持多个连接，它是使用单个连接的常见设计模式。

1. 修改 HomeScreen 构造函数以实例化 DataAccess 对象。

   下面是已完全更新的 HomeScreen 类。将 DataAccess 对象保存为类级别变量，以便从代码的其它部分对其进行访问。

```
class HomeScreen extends MainScreen {

    HomeScreen() {

        // Set the window title
        LabelField applicationTitle = new LabelField("Hello BlackBerry");
        setTitle(applicationTitle);

        // Add a label to show application status
        _statusLabel = new LabelField( "Status: Started");
        add( _statusLabel );

        // Create database and connect
        try{
            _da = DataAccess.getDataAccess(false);
            _statusLabel.setText("Status: Connected");
        }
        catch( Exception ex)
        {
            _statusLabel.setText("Exception: " + ex.toString() );
        }
    }
    private LabelField _statusLabel;
    private DataAccess _da;
}
```

2. 在 HelloBlackBerry 项目中创建名为 *myapp\DataAccess.java* 的文件。

3. 提供确保单个数据库连接的方法 getDataAccess。

```
package myapp;

import ianywhere.ultralitej.*;
import java.util.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

class DataAccess {
    DataAccess() {    }

    public static synchronized DataAccess getDataAccess(boolean reset)
                throws Exception
    {
        if( _da == null ){
            _da = new DataAccess();
            ConfigObjectStore _config =
             DatabaseManager.createConfigurationObjectStore("HelloDB");
            if(reset)
            {
```

```
                    _conn = DatabaseManager.createDatabase( _config );
                }
                else
                {
                    try{
                        _conn = DatabaseManager.connect( _config );
                    }
                    catch( ULjException uex1) {
                        if( uex1.getErrorCode() !=
                            ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
                            System.out.println( "Exception: " +
                                                uex1.toString() );
                            Dialog.alert( "Exception: " + uex1.toString() +
                                          ". Recreating database..." );
                        }
                        _conn = DatabaseManager.createDatabase( _config );
                    }
                }
                // _da.createDatabaseSchema();
            }
            return _da;
        }
        private static Connection _conn;
        private static DataAccess _da;
    }
```

此类从 *UltraLiteJ.jar* 文件导入 ianywhere.ultralitej 包。创建或连接到数据库的步骤如下：

a. 定义配置。在本示例中，配置为 ConfigObjectStore 配置对象，这意味着 UltraLiteJ 数据库保存在 BlackBerry 对象存储库中。

b. 尝试连接到数据库。

   如果连接尝试失败，则创建数据库。createDatabase 方法随后返回一个已打开的连接。

4. 按下 F5 构建应用程序并将其部署到设备模拟器。
5. 从 [**File**] 菜单中选择 [**Load Java Program**]。
6. 浏览至 UltraLiteJ 安装目录的 *J2meRim11* 文件夹，然后打开 *UltraLiteJ.cod* 文件。
7. 从模拟器运行程序。

   您应该会看到一条状态消息，它指示应用程序已成功连接到数据库。

第 4 课：创建数据库表

在本课中，您将创建一个名为 Names 的简单表，其中包含具有以下属性的两列：

列名	数据类型	是否允许空值？	缺省值	是否为主键？
ID	UUID	否	无	是
Name	Varchar(254)	否	None	否

1. 添加 DataAccess 方法以创建表。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
private void createDatabaseSchema()
{
    try{
        _conn.schemaCreateBegin();
        ColumnSchema column_schema;
        TableSchema table_schema = _conn.createTable("Names");
        column_schema = table_schema.createColumn( "ID", Domain.UUID );
        column_schema.setDefault( ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID);
        table_schema.createColumn( "Name", Domain.VARCHAR, 254 );
        IndexSchema index_schema =
                table_schema.createPrimaryIndex("prime_keys");
        index_schema.addColumn("ID", IndexSchema.ASCENDING);
        _conn.schemaCreateComplete();
    }
    catch( ULjException uex1){
        System.out.println( "ULjException: " + uex1.toString() );
    }
    catch( Exception ex1){
        System.out.println( "Exception: " + ex1.toString() );
    }
}
```

如果表已存在，则抛出异常。

2. 调用 DataAccess.getDataAccess 方法。

   取消第 1 部分第 3 课第 3 步的示例代码中调用 createDatabaseSchema 的注释。对 createDatabaseSchema 的调用应如下所示：

   ```
   _da.createDatabaseSchema()
   ```

3. 在模拟器上再次运行应用程序。

变更表模式

变更表模式（例如添加表定义）时，必须记住以下信息：

- 必须在对 schemaCreateBegin 和 schemaCreateEnd 调用结束前完成所有模式的创建。
- Connection.createTable 方法创建空表。
- TableSchema.createColumn 方法创建列。
- TableSchema.createPrimaryIndex 方法创建主键。

第 5 课： 向表中添加数据

在本课中，您将向屏幕添加以下控件：

- 可在其中输入名称的文本字段。
- 将文本字段中的名称添加到数据库的菜单项。
- 显示表中名称的列表字段。

然后，您将添加代码以在文本字段中插入名称并刷新列表。

1. 向屏幕中添加控件。

   a. 调用 getDataAccess 方法之前添加以下代码。

```
// Add an edit field for entering new names
_nameEditField = new EditField( "Name: ", "", 50,
EditField.USE_ALL_WIDTH );
add (_nameEditField );

// Add an ObjectListField for displaying a list of names
_nameListField = new ObjectListField();
add( _nameListField );

// Add a menu item
addMenuItem(_addToListMenuItem);

// Create database and connect
try{
    _da = DataAccess.getDataAccess();
```

b. 为 _nameEditField 和 _nameListField 添加类级别声明。同样，使用 run 方法定义 _addToListMenuItem MenuItem（目前为空）。这些声明位于 _statusLabel 和 _da 声明之后。

```
private EditField _nameEditField;
private ObjectListField _nameListField;

private MenuItem _addToListMenuItem = new MenuItem("Add", 1, 1){
    public void run() {
        // TODO
    }
};
```

c. 重新编译应用程序并确认它可以运行。

2. 向应用程序添加以下方法和对象：

- 将行插入表的 DataAccess 方法
- 将 Names 表的行保存为对象的对象
- 将表中的行读取到对象的矢量的 DataAccess 方法
- 刷新显示在 HomeScreen 上的列表内容的方法
- 将某项添加到 HomeScreen 上的列表的方法。

a. 添加将行插入表中的 DataAccess 方法：

```
public void insertName( String name ){
    try{
        Value nameID = _conn.createUUIDValue();
        String sql = "INSERT INTO Names( ID, Name ) VALUES
                     ( ?, ? )";
        PreparedStatement ps = _conn.prepareStatement(sql);
        ps.set(1, nameID );
        ps.set(2, name );
        ps.execute();
        _conn.commit();
    }
    catch( ULjException uex ){
        System.out.println( "ULjException: " + uex.toString() );
    }
    catch( Exception ex ){
        System.out.println( "Exception: " + ex.toString() );
    }
}
```

b. 添加保存 Names 表的行的类。toString 方法由 ObjectListField 控件使用。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
package myapp;

class NameRow {

    public NameRow( String nameID, String name ) {
        _nameID = nameID;
        _name = name;
    }

    public String getNameID(){
        return _nameID;
    }

    public String getName(){
        return _name;
    }

    public String toString(){
        return _name;
    }

    private String _nameID;
    private String _name;

}
```

c. 添加将表中的行读取到对象的矢量的 DataAccess 方法：

```
public Vector getNameVector(){
    Vector nameVector = new Vector();
    try{
        String sql = "SELECT ID, Name FROM Names";
        PreparedStatement ps = _conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        while ( rs.next() ){
            String nameID = rs.getString(1);
            String name = rs.getString(2);
            NameRow nr = new NameRow( nameID, name);
            nameVector.addElement(nr);
        }
    }
    catch( ULjException uex ){
        System.out.println( "ULjException: " + uex.toString() );
    }
    catch( Exception ex ){
        System.out.println( "Exception: " + ex.toString() );
    }
    finally{
        return nameVector;
    }
}
```

d. 将用户接口方法添加到 HomeScreen 类。以下是刷新名称列表的方法：

```
public void refreshNameList(){
    //Clear the list
    _nameListField.setSize(0);
    //Refill from the list of names
    Vector nameVector = _da.getNameVector();
    for( Enumeration e = nameVector.elements(); e.hasMoreElements(); ){
        NameRow nr = ( NameRow )e.nextElement();
        _nameListField.insert(0, nr);
```

```
        }
    }
```

e. 在 HomeScreen 构造函数结束之前调用 refreshNameList 方法，以便应用程序启动时列表已填充。

```
// Fill the ObjectListField
this.refreshNameList();
```

f. 添加用于将行添加到列表的 HomeScreen 方法：

```
private void onAddToList(){
    _da.insertName(_nameEditField.getText());
    this.refreshNameList();
    _nameEditField.setText("");
}
```

g. 从 _addToListMenuItem MenuItem（当前为 //TODO）的 run 方法内调用此方法：

```
public void run() {
    onAddToList();
}
```

3. 编译并运行应用程序。

重置模拟器
如果需要将模拟器重置为干净的状态，请从 BlackBerry JDE 的 [**File**] 菜单（而不是 [**Simulator**] 菜单）中选择 [**Erase Simulator File**]，然后消除子菜单中的项。如果用此方法重置模拟器，则必须在再次运行应用程序之前重新导入 *UltraLiteJ.cod* 文件。

第 6 课：将应用程序部署到智能手机

有几种方法可将应用程序部署到 BlackBerry 智能手机。本课程介绍如何使用 BlackBerry Desktop Manager 软件来部署应用程序。

在 BlackBerry 上运行的应用程序必须使用 BlackBerry Signature Tool 进行标记。此工具可从 Research in Motion (RIM) 中作为 BlackBerry JDE Component Package 的一部分获得。虽然 *UltraLiteJ.cod* 文件已标记，但是您必须标记 *HelloBlackBerry.cod* 文件。

注意
您必须从 RIM 获得密钥才可以使用 BlackBerry Signature Tool 签署应用程序。有关获取密钥的详细信息，请访问 BlackBerry Developer Program Web 站点，网址为 http://na.blackberry.com/eng/developers/。

签署应用程序

1. 启动 BlackBerry Signature Tool。
2. 浏览到已编译的应用程序，即 *HelloBlackBerry.cod* 文件，并将其选中。
3. 单击 [**Request To Sign The File**]。
4. 单击 [**Close**] 关闭标记工具。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

部署应用程序

这些步骤介绍如何使用 BlackBerry Desktop Manager 来将文件部署到设备。

1. 使用 USB 电缆将 BlackBerry 连接到计算机，并确保 Desktop Manager 能够看到该设备。
2. 单击 [**Application Loader**] 并按照向导中的说明进行操作。
3. 浏览至 *HelloBlackBerry.alx* 文件并将其添加到设备上。
4. 浏览至 *J2meRim11\UltraLiteJ.alx* 文件并将其添加到设备上。

现在，您应该能够使用 BlackBerry 智能手机上的应用程序。

第 2 部分：向 BlackBerry 应用程序添加同步

本部分将扩展应用程序以处理同步。其中包括创建 SQL Anywhere 数据库、运行 MobiLink 服务器和从 BlackBerry 应用程序添加同步函数。

第 1 课：创建 SQL Anywhere 数据库

数据同步需要一个可与 UltraLiteJ 保持同步的统一数据库。在本课中，您将创建 SQL Anywhere 数据库。

1. 在应用程序目录中，创建一个子目录用来保存名为 *c:\tutorial\database* 的 SQL Anywhere 数据库。
2. 从 *c:\tutorial\database* 运行以下命令，以创建空 SQL Anywhere 数据库：

   ```
   dbinit HelloBlackBerry.db
   ```

3. 创建要连接到数据库的 ODBC 数据源。
   a. 打开 ODBC 管理器。

      选择 [开始] » [程序] » [**SQL Anywhere 11**] » [**ODBC 管理器**]。
   b. 单击 [**用户 DSN**] 选项卡，为当前用户创建 ODBC 数据源。
   c. 单击 [**添加**]。
   d. 在驱动程序列表中，选择 [**SQL Anywhere 11**]，然后单击 [**完成**]。
   e. 单击 [**ODBC**] 选项卡。
   f. 在 [**数据源名**] 字段中，键入 **HelloBlackBerry**。
   g. 单击 [**登录**] 选项卡。
   h. 在 [**用户 ID**] 字段中，键入 **DBA**，然后在 [**口令**] 字段中键入 **SQL**。

      这些是所有 SQL Anywhere 数据库的缺省登录名和口令，不应在生产环境中使用。
   i. 单击 [**数据库**] 选项卡，键入 **HelloBlackBerry** 作为 [**服务器名**]，而 *c:\tutorial\database\HelloBlackBerry.db* 作为 [**数据库文件**]。单击 [**确定**]。
4. 运行以下命令以启动 Interactive SQL 并连接到 SQL Anywhere 数据库：

   ```
   dbisql -c dsn=HelloBlackBerry
   ```

5. 执行以下语句，以在数据库中创建表：

   ```
   CREATE TABLE Names (
    ID UNIQUEIDENTIFIER NOT NULL DEFAULT newID(),
    Name varchar(254),
    PRIMARY KEY (ID)
   );
   ```

6. 关闭 Interactive SQL。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

第 2 课：创建 MobiLink 脚本并启动 MobiLink 服务器

在本课中，您将使用 Sybase Central 准备统一数据库，以进行同步。

1. 从 [开始] 菜单中选择 [程序] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 在 Sybase Central [任务] 窗格中，从 MobiLink 11 任务中选择 [创建同步模型]。
   a. 键入同步模型名称 **HelloBlackBerrySyncModel**，并将该模型存储在 *c:\tutorial\database* 文件夹中。单击 [下一步]。
   b. 单击 [选择统一数据库]。
   c. 在 [连接到统一数据库] 窗口中，选择 HelloBlackBerry ODBC 数据源并单击 [确定]。
   d. 单击 [是] 创建 MobiLink 系统设置。
   e. 选择 [否，新建一个远程数据库模式]。
   f. 对于下载，选择 [基于时间戳的下载]。
   g. 单击 [完成] 完成创建同步模型并保存该项目。
3. 右击同步模型，然后选择 [部署]。选择仅部署到统一数据库。
4. 在 [部署同步模型向导] 中，选择 [**Save SQL**] 并部署到数据库。当提示选择统一数据库时，指定 ODBC 数据源 **HelloBlackBerry**。
5. 对于 MobiLink 用户和口令，选择用户名 **mluser** 和口令 **mlpassword**。
6. 单击 [完成] 将同步模型部署到统一数据库。

第 3 课：向应用程序添加同步

在本课中，您将向应用程序添加同步功能。

1. 在 HomeScreen 构造函数中，添加 [同步] 菜单项。

```
// Add a menu item
addMenuItem(_addToListMenuItem);

// Add sync menu item
addMenuItem(_syncMenuItem);

// Create database and connect
try{ ...
```

2. 在类变量声明中定义菜单项。

```
private MenuItem _addToListMenuItem = new MenuItem("Add", 1, 1){
        public void run() {
            onAddToList();
        }
};
private MenuItem _syncMenuItem = new MenuItem("Sync", 2, 1){
        public void run() {
            onSync();
```

```
        }
};
```

3. 创建 onSync 方法。

```
private void onSync(){
    try{
        if( _da.sync() ){
            _statusLabel.setText("Synchronization succeeded");
        } else {
            _statusLabel.setText("Synchronization failed");
          }
        this.refreshNameList();
    } catch ( Exception ex){
        System.out.println( ex.toString() );
    }
}
```

4. 在类级别中定义 syncParms 和 streamParms 变量。

```
private static SyncParms _syncParms;
private static StreamHTTPParms _streamParms;
```

5. 在 DataAccess 类中，添加 sync 方法。

```
public boolean sync() {
    try {
        if( _syncParms == null ){
            String host = "ultralitej.sybase.com";
            _syncParms = _conn.createSyncParms( "mluser",
"HelloBlackBerrySyncModel" );
            _syncParms.setPassword("mlpassword");
            _streamParms = _syncParms.getStreamParms();
            _streamParms.setPort( 80 ); // use your own
            _streamParms.setHost( host ); // use your own
            if(host.equals("ultralitej.sybase.com"))
            {
                _streamParms.setURLSuffix("scripts/iaredirect.dll/ml/
HelloBlackBerry/");
            }
        }
        System.out.println( "Synchronizing" );
        _conn.synchronize( _syncParms );
        return true;
    }
    catch( ULjException uex){
        System.out.println(uex.toString());
        return false;
    }
}
```

同步参数对象 SyncParms 包括部署同步模型时指定的用户名和口令。也包括所创建的同步模型的名称。现在，在 MobiLink 中，此名称是指部署到统一数据库的同步版本或一组同步逻辑。

流参数对象 StreamHTTPParms 表示 MobiLink 服务器的主机名和端口号。在下一课中启动 MobiLink 服务器时，使用您自己的计算机名并选择可用的端口。不要使用 localhost 作为计算机名。如果计算机上没有运行 Web 服务器，则您可以使用端口 80。

6. 编译应用程序。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

第 4 课：在模拟器上运行应用程序

在可以运行 BlackBerry 应用程序和同步之前，MobiLink 服务器必须正在运行。MDS 模拟器也必须在运行以提供设备模拟器和 MobiLink 之间的通信通道。

1. 通过从 *c:\tutorial\database* 运行以下命令启动 MobiLink：

   ```
   mlsrv11 -c " DSN=HelloBlackBerry" -v+ -x http(port=8081) -ot ml.txt
   ```

   -c 选项用于将 MobiLink 连接到 SQL Anywhere 数据库。v+ 选项设置高级别的详细程度，以便您可以按照服务器窗口中发生的情况进行操作。-x 选项指示用于通信的端口号。-ot 选项指定将在启动 MobiLink 服务器的目录中创建日志文件 (*ml.txt*)。

2. 选择 [开始] » [程序] » [**Research in Motion**] » [**BlackBerry Email and MDS Services Simulator 4.1.2**] » [**MDS**]。

3. 在服务器中输入名称。

   a. 启动 Interactive SQL 并连接到 HelloBlackBerry 数据源。

   b. 运行下面的 SQL 语句以添加名称：

      ```
      INSERT Names ( Name ) VALUES ( 'ServerName1' );
      INSERT Names ( Name ) VALUES ( 'ServerName2' );
      COMMIT;
      ```

4. 从 JDE 中，按下 F5 编译应用程序并在设备模拟器中运行该应用程序。

5. 浏览至主屏幕并将名称添加到列表。

6. 同步应用程序。

7. 从主屏幕中，显示菜单项并选择 [同步]。

   在服务器中输入的名称会出现在屏幕中。如果从 Interactive SQL 来查询 Names 表中的名称，则会看到在模拟器中输入的名称都已到达服务器。

教程的代码列表

本节将提供在前面教程中介绍的完整代码。在这些教程中出现了四种 Java 类。

另请参见

- “第 1 部分：创建用于 BlackBerry 的 UltraLiteJ 应用程序”一节第 67 页
- “第 2 部分：向 BlackBerry 应用程序添加同步”一节第 76 页

Application.java

```
/*
 * Application.java
 *
 * ? <your company here>, 2003-2005
 * Confidential and proprietary.
 */

package myapp;


/**
 *
 */
class Application extends net.rim.device.api.ui.UiApplication {

    public static void main( String[] args )
    {
        Application instance = new Application();
        instance.enterEventDispatcher();
    }

    Application() {
        pushScreen( new HomeScreen() );
    }
}
```

DataAccess.java

```
/*
 * DataAccess.java
 *
 * ? <your company here>, 2003-2005
 * Confidential and proprietary.
 */

package myapp;


import ianywhere.ultralitej.*;
import java.util.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

/**
 *
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
 */
class DataAccess {
    DataAccess() {    }

    public static synchronized DataAccess getDataAccess(boolean reset)
       throws Exception
    {
    try{
        if( _da == null ){
            _da = new DataAccess();
            ConfigObjectStore _config =
               DatabaseManager.createConfigurationObjectStore("HelloDB");
            if(reset)
            {
                _conn = DatabaseManager.createDatabase( _config );
            }
            else
            {
                try{
                    _conn = DatabaseManager.connect( _config );
                }
                catch( ULjException uex1) {
                    if( uex1.getErrorCode() !=
                        ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
                        System.out.println( "Exception: " +
uex1.toString() );
                        Dialog.alert( "Exception: " + uex1.toString() +
                           ". Recreating database..." );
                    }
                    _conn = DatabaseManager.createDatabase( _config );
                }
            }
            _da.createDatabaseSchema();
        }
        return _da;
    } catch ( ULjException ue)
    {
        System.out.println("Exception in getDataAccess" + ue.toString() );
        return null;
    }
    }

    /**
     *
     * Create the table in the database.
     * If the table already exists, a harmless exception is thrown
     */
    private void createDatabaseSchema()
    {
        try{
            _conn.schemaCreateBegin();
            ColumnSchema column_schema;
            TableSchema table_schema = _conn.createTable("Names");
            column_schema = table_schema.createColumn( "ID", Domain.UUID );
            column_schema.setDefault( ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID);
            table_schema.createColumn( "Name", Domain.VARCHAR, 254 );
            IndexSchema index_schema =
               table_schema.createPrimaryIndex("prime_keys");
            index_schema.addColumn("ID", IndexSchema.ASCENDING);
            _conn.schemaCreateComplete();
        }
        catch( ULjException uex1){
            System.out.println( "ULjException: " + uex1.toString() );
        }
```

```
        catch( Exception ex1){
            System.out.println( "Exception: " + ex1.toString() );
        }
    }

    public void insertName( String name ){
        try{
            Value nameID = _conn.createUUIDValue();
            String sql = "INSERT INTO Names( ID, Name ) VALUES
( ?, ? )";
            PreparedStatement ps = _conn.prepareStatement(sql);
            ps.set(1, nameID );
            ps.set(2, name );
            ps.execute();
            _conn.commit();
        }
        catch( ULjException uex ){
            System.out.println( "ULjException: " + uex.toString() );
        }
        catch( Exception ex ){
            System.out.println( "Exception: " + ex.toString() );
        }
    }

    public Vector getNameVector(){
        Vector nameVector = new Vector();
        try{
            String sql = "SELECT ID, Name FROM Names";
            PreparedStatement ps = _conn.prepareStatement(sql);
            ResultSet rs = ps.executeQuery();
            while ( rs.next() ){
                String nameID = rs.getString(1);
                String name = rs.getString(2);
                NameRow nr = new NameRow( nameID, name);
                nameVector.addElement(nr);
            }
        }
        catch( ULjException uex ){
            System.out.println( "ULjException: " + uex.toString() );
        }
        catch( Exception ex ){
            System.out.println( "Exception: " + ex.toString() );
        }
        finally{
            return nameVector;
        }
    }

    public boolean sync() {
        try {
            if( _syncParms == null ){
                String host = "ultralitej.sybase.com";
                _syncParms = _conn.createSyncParms( "mluser",
                             "HelloBlackBerrySyncModel" );
                _syncParms.setPassword("mlpassword");
                _streamParms = _syncParms.getStreamParms();
                _streamParms.setPort( 80 ); // use your own
                _streamParms.setHost( host ); // use your own
                if(host.equals("ultralitej.sybase.com"))
                {
                    _streamParms.setURLSuffix(
                       "scripts/iaredirect.dll/ml/HelloBlackBerry/");
                }
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
                }
                System.out.println( "Synchronizing" );
                _conn.synchronize( _syncParms );
                return true;
            }
            catch( ULjException uex){
                System.out.println(uex.toString());
                return false;
            }
        }

        public boolean complete() {
            try{
                _conn.checkpoint();
                _conn.release();
                _conn = null;
                _da = null;
                _config = null;
                return true;
            }
            catch(Exception e){
                return false;
            }
        }

        private static ConfigObjectStore _config;
        private static Connection _conn;
        private static DataAccess _da;
        private static SyncParms _syncParms;
        private static StreamHTTPParms _streamParms;
    }
```

HomeScreen.java

```
/*
 * HomeScreen.java
 *
 * ? <your company here>, 2003-2005
 * Confidential and proprietary.
 */

package myapp;


import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import java.util.*;
/**
 *
 */
class HomeScreen extends MainScreen {


    HomeScreen() {

        // Set the window title
        LabelField applicationTitle = new LabelField("Hello BlackBerry");
        setTitle(applicationTitle);

        // Add a label to show application status
```

```
        _statusLabel = new LabelField( "Status: Started");
        add( _statusLabel );

        // Add an edit field for entering new names
        _nameEditField = new EditField( "Name: ", "", 50,
           EditField.USE_ALL_WIDTH );
        add (_nameEditField );

        // Add an ObjectListField for displaying a list of names
        _nameListField = new ObjectListField();
        add( _nameListField );

        // Add a menu item
        addMenuItem(_addToListMenuItem);

        // Add sync menu item
        addMenuItem(_syncMenuItem);

        // Add reset menu item
        addMenuItem(_resetMenuItem);

        // Create database and connect
        try{
            _da = DataAccess.getDataAccess(false);
            _statusLabel.setText("Status: Connected");
        }
        catch( Exception ex)
        {
            System.out.println("Exception: " + ex.toString() );
            _statusLabel.setText("Exception: " + ex.toString() );
        }

        // Fill the ObjectListField
        this.refreshNameList();

    }

    public void refreshNameList(){
        try{
            //Clear the list
            _nameListField.setSize(0);
            //Refill from the list of names
            Vector nameVector = _da.getNameVector();
           for( Enumeration e = nameVector.elements(); e.hasMoreElements(); )
{
                NameRow nr = ( NameRow )e.nextElement();
                _nameListField.insert(0, nr);
            }
        } catch ( Exception ex){
            System.out.println(ex.toString());
        }
    }

    private void onAddToList(){
        String name = _nameEditField.getText();
        _da.insertName(name);
        this.refreshNameList();
        _nameEditField.setText("");
        _statusLabel.setText(name + " added to list");
    }

    private void onSync(){
        try{
            if( _da.sync() ){
```


Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
                _statusLabel.setText("Synchronization succeeded");
            } else {
                _statusLabel.setText("Synchronization failed");
            }
            this.refreshNameList();
        } catch ( Exception ex){
            System.out.println( ex.toString() );
        }
    }

    private void onReset(){
        _da.complete();
        try{
            _da = DataAccess.getDataAccess(true);
            _statusLabel.setText("Status: Connected");
            this.refreshNameList();
        }
        catch( Exception ex)
        {
            System.out.println("Exception: " + ex.toString() );
            _statusLabel.setText("Exception: " + ex.toString() );
        }
    }

    private LabelField _statusLabel;
    private DataAccess _da;
    private EditField _nameEditField;
    private ObjectListField _nameListField;
    private MenuItem _addToListMenuItem = new MenuItem("Add", 1, 1){
        public void run() {
            onAddToList();
        }
    };
    private MenuItem _syncMenuItem = new MenuItem("Sync", 2, 1){
        public void run() {
            onSync();
        }
    };
    private MenuItem _resetMenuItem = new MenuItem("Reset", 3, 1){
        public void run() {
            onReset();
        }
    };

}
```

NameRow.java

```
/*
 * NameRow.java
 *
 * ? <your company here>, 2003-2005
 * Confidential and proprietary.
 */

package myapp;



/**
```

```
 * Hold a row of the Name table as an object
 */
class NameRow {

    public NameRow( String nameID, String name ) {
        _nameID = nameID;
        _name = name;
    }

    public String getNameID(){
        return _nameID;
    }

    public String getName(){
        return _name;
    }


    /**
     * Required for use by the ObjectListField in HomeScreen
     *
     * @return The Name as a string
     */
    public String toString(){
        return _name;
    }

    private String _nameID;
    private String _name;

}
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

UltraLiteJ 参考

UltraLiteJ API 参考 89
UltraLiteJ 系统表 265
UltraLiteJ 实用程序 275

UltraLiteJ API 参考

目录

CollectionOfValueReaders 接口 91
CollectionOfValueWriters 接口 98
ColumnSchema 接口 104
ConfigFile 接口 109
ConfigNonPersistent 接口 110
ConfigObjectStore 接口（仅限 J2ME BlackBerry） 111
ConfigPersistent 接口 112
ConfigRecordStore 接口（仅限 J2ME） 119
Configuration 接口 120
Connection 接口 122
DatabaseInfo 接口 144
DatabaseManager 类 147
DecimalNumber 接口 153
Domain 接口 156
EncryptionControl 接口 169
ForeignKeySchema 接口 171
IndexSchema 接口 173
PreparedStatement 接口 176
ResultSet 接口 180
ResultSetMetadata 接口 183
SISListener 接口（仅限 J2ME BlackBerry） 184
SISRequestHandler 接口（仅限 J2ME BlackBerry） 185
SQLCode 接口 186
StreamHTTPParms 接口 205
StreamHTTPSParms 接口 209
SyncObserver 接口 213
SyncObserver.States 接口 215
SyncParms 类 219
SyncResult 类 234
SyncResult.AuthStatusCode 接口 238
TableSchema 接口 240

ULjException 类 248
Value 接口 252
ValueReader 接口 256
ValueWriter 接口 260

包

ianywhere.ultralitej

CollectionOfValueReaders 接口

提供用于返回某一给定行的列值的方法。

语法

public **CollectionOfValueReaders**

派生类

- "ResultSet 接口"一节第 180 页

注释

将根据由 PreparedStatement 声明的 SQL SELECT 语句来返回结果，并将结果保存在 ResultSet 中。

在此接口中由任何给定方法返回的值将根据整型参数 ordinal 来进行访问，该参数按照列在 SQL SELECT 语句中出现的顺序指定列顺序。ordinal 参数具有指数基 1。

值作为 java 原始类型或只读 Value 对象返回。

以下示例展示了如何使用 SQL SELECT 语句声明新的 PreparedStatement、执行语句、将查询结果存储在新的 ResultSet 中，以及使用 get 方法将 column1 值存储为 String。

```
// Define a new SQL SELECT statement.
String sql_string = "SELECT column1, column2 FROM SampleTable";

// Create a new PreparedStatement from an existing connection.
PreparedStatement ps = conn.prepareStatement(sql_string);

// Create a new ResultSet to contain the query results of the SQL statement.
ResultSet rs = ps.executeQuery();

// Check if the PreparedStatement contains a ResultSet.
if (ps.hasResultSet()) {
    // Retrieve the column1 value using getString.
    String row1_col1 = rs.getString(1);
}
```

成员

CollectionOfValueReaders 的所有成员，包括所有继承的成员。

- "getBlobInputStream 方法"一节第 92 页
- "getBoolean 方法"一节第 92 页
- "getBytes 方法"一节第 93 页
- "getClobReader 方法"一节第 93 页
- "getDate 方法"一节第 93 页
- "getDecimalNumber 方法"一节第 94 页
- "getDouble 方法"一节第 94 页
- "getFloat 方法"一节第 94 页
- "getInt 方法"一节第 95 页
- "getLong 方法"一节第 95 页
- "getOrdinal 方法"一节第 96 页
- "getString 方法"一节第 96 页
- "getValue 方法"一节第 96 页
- "isNull 方法"一节第 97 页

getBlobInputStream 方法

返回 InputStream。

语法

java.io.InputStream **CollectionOfValueReaders.getBlobInputStream(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的 InputStream 表示。

getBoolean 方法

返回布尔值。

语法

boolean **CollectionOfValueReaders.getBoolean(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

返回值

指定值的布尔表示。

getBytes 方法

返回一个字节数组。

语法

byte[] **CollectionOfValueReaders.getBytes(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的字节数组表示。

getClobReader 方法

返回读取器。

语法

java.io.Reader **CollectionOfValueReaders.getClobReader(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的读取器表示。

getDate 方法

返回 java.util.Date。

语法

java.util.Date **CollectionOfValueReaders.getDate(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的 java.util.Date 表示。

getDecimalNumber 方法

返回 DecimalNumber。

语法

```
DecimalNumber CollectionOfValueReaders.getDecimalNumber(
  int ordinal
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的 DecimalNumber 表示。

getDouble 方法

返回双精度值。

语法

```
double CollectionOfValueReaders.getDouble(
  int ordinal
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的双精度表示。

getFloat 方法

返回浮点值。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

float **CollectionOfValueReaders.getFloat(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的浮点表示。

getInt 方法

返回整数值。

语法

int **CollectionOfValueReaders.getInt(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的整数表示。

getLong 方法

返回长整数值。

语法

long **CollectionOfValueReaders.getLong(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的长整数表示。

getOrdinal 方法

返回由字符串表示的值的序数（基数为 1）。

语法

int **CollectionOfValueReaders.getOrdinal(**
  String *name*
) throws **ULjException**

参数

- **name**　表示表的列名的字符串。

返回值

序数值。

getString 方法

返回字符串值。

语法

String **CollectionOfValueReaders.getString(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的字符串表示。

getValue 方法

返回 Value 对象。

语法

Value **CollectionOfValueReaders.getValue(**
  int *ordinal*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的 Value 对象表示。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

isNull 方法

测试某一值是否为空。

语法

```
boolean CollectionOfValueReaders.isNull(
  int ordinal
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

如果值为 NULL，则返回 true，否则返回 false。

CollectionOfValueWriters 接口

提供用于设置某一给定行的列值的方法。

语法

public **CollectionOfValueWriters**

派生类

- “PreparedStatement 接口”一节第 176 页

注释

所有方法都根据在 PreparedStatement 的初始声明中定义的 SQL 语句来准备，并通过 execute 方法应用到数据库。

更新列值时，必须通过在准备的 SQL 语句中出现的列序号来引用列。该序号作为整型参数 ordinal 传递，该参数具有指数基 1。

以下示例展示了如何使用 SQL UPDATE 语句声明新的 PreparedStatement、使用 set 方法准备更改，以及将这些更改应用到 UltraLite 数据库。

```
// Define a new prepared SQL statement.
String sql_string = "UPDATE SampleTable SET column1 = ? WHERE pkey = 1";

// Create a new PreparedStatement from an existing connection.
PreparedStatement ps = conn.prepareStatement(sql_string);

// Set a String value to the first column in the SQL statement (column1).
ps.set(1, "New Value");

// Commit the changes to the database.
conn.commit();
```

成员

CollectionOfValueWriters 的所有成员，包括所有继承的成员。

- “getBlobOutputStream 方法”一节第 99 页
- “getClobWriter 方法”一节第 99 页
- “getOrdinal 方法”一节第 99 页
- “set 方法”一节第 100 页
- “set 方法”一节第 100 页
- “set 方法”一节第 100 页
- “set 方法”一节第 101 页
- “set 方法”一节第 101 页
- “set 方法”一节第 101 页
- “set 方法”一节第 102 页
- “set 方法”一节第 102 页
- “set 方法”一节第 102 页
- “set 方法”一节第 103 页
- “setNull 方法”一节第 103 页

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

getBlobOutputStream 方法

返回 OutputStream。

语法

```
java.io.OutputStream CollectionOfValueWriters.getBlobOutputStream(
  int ordinal
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的 OutputputStream。

getClobWriter 方法

返回写入器。

语法

```
java.io.Writer CollectionOfValueWriters.getClobWriter(
  int ordinal
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。

返回值

指定值的写入器。

getOrdinal 方法

返回由名称表示的值的序数（基数：1）。

语法

```
int CollectionOfValueWriters.getOrdinal(
  String name
) throws ULjException
```

参数

- **name** 表示表的列名的字符串。

返回值

由名称表示的值的序数（基数：1）。

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置布尔值。

语法

```
void CollectionOfValueWriters.set(
  int ordinal,
  boolean value
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value** 要设置的值。

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置 DecimalNumber。

语法

```
void CollectionOfValueWriters.set(
  int ordinal,
  DecimalNumber value
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value** 要设置的值。

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置整数值。

语法

```
void CollectionOfValueWriters.set(
  int ordinal,
  int value
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value** 要设置的值。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置 java.util.Date。

语法

void **CollectionOfValueWriters.set(**
 int *ordinal*,
 java.util.Date *value*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value**　要设置的值。

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置长整数值。

语法

void **CollectionOfValueWriters.set(**
 int *ordinal*,
 long *value*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value**　要设置的值。

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置浮点值。

语法

void **CollectionOfValueWriters.set(**
 int *ordinal*,
 float *value*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value**　要设置的值。

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置双精度值。

语法

void **CollectionOfValueWriters.set(**
  int *ordinal*,
  double *value*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value**　要设置的值。

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置字节数组值。

语法

void **CollectionOfValueWriters.set(**
  int *ordinal*,
  byte[] *value*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value**　要设置的值。

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置字符串值。

语法

void **CollectionOfValueWriters.set(**
  int *ordinal*,
  String *value*
) throws **ULjException**

参数

- **ordinal**　以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value**　要设置的值。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

set 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置 Value 对象。

语法

```
void CollectionOfValueWriters.set(
  int ordinal,
  Value value
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。
- **value** 要设置的值。

setNull 方法

为 SQL 语句中由 ordinal 参数定义的列编号设置空值。

语法

```
void CollectionOfValueWriters.setNull(
  int ordinal
) throws ULjException
```

参数

- **ordinal** 以 1 为基数的整数，表示 SQL 语句中排序的列编号。

ColumnSchema 接口

指定列的模式。

语法

public **ColumnSchema**

注释

支持此接口的对象由 TableSchema.createColumn(String,short)、TableSchema.createColumn(String,short,int) 和 TableSchema.createColumn(String,short,int,int) 方法返回。

以下示例演示了简单数据库模式的创建过程。使用自动递增的主键整数列创建 T1 表。

```
// Assumes a valid Connection object
TableSchema table_schema;
ColumnSchema col_schema;
IndexSchema index_schema;

table_schema = conn.createTable("T1");
col_schema = table_schema.createColumn("num", Domain.INTEGER);
col_schema.setDefault(ColumnSchema.COLUMN_DEFAULT_AUTOINC);

// BIT columns are not nullable by default.
col_schema = table_schema.createColumn("flag", Domain.BIT);
col_schema.setNullable(true);
col_schema = table_schema.createColumn(
    "cost", Domain.NUMERIC, 10, 2
    );
col_schema.setNullable(false);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("num", IndexSchema.ASCENDING);
conn.schemaCreateComplete();
```

成员

ColumnSchema 的所有成员，包括所有继承的成员。

- "COLUMN_DEFAULT_AUTOINC 变量"一节第 104 页
- "COLUMN_DEFAULT_CURRENT_DATE 变量"一节第 105 页
- "COLUMN_DEFAULT_CURRENT_TIME 变量"一节第 105 页
- "COLUMN_DEFAULT_CURRENT_TIMESTAMP 变量"一节第 106 页
- "COLUMN_DEFAULT_GLOBAL_AUTOINC 变量"一节第 106 页
- "COLUMN_DEFAULT_NONE 变量"一节第 106 页
- "COLUMN_DEFAULT_UNIQUE_ID 变量"一节第 107 页
- "setDefault 方法"一节第 107 页
- "setNullable 方法"一节第 108 页

COLUMN_DEFAULT_AUTOINC 变量

指定列自动递增。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

final byte **ColumnSchema.COLUMN_DEFAULT_AUTOINC**

注释

使用 AUTOINCREMENT 时，列必须是整型数据类型之一或是精确的数字类型。执行 INSERT 操作时，如果没有指定 AUTOINCREMENT 列的值，则生成一个比列中的任何其它值都大的唯一值。如果 INSERT 指定的列值大于列的当前最大值，则该值将用作后续插入的起点值。

在 UltraLiteJ 中，创建表时自动增量值不会设置为 0。如果该列使用带符号的数据类型，自动增量列将生成负值。因此，应将 AUTOINCREMENT 列声明为无符号整数，以防止使用负值。

可通过查询系统表 TableSchema.SYS_COLUMNS 的 column_default 列来确定现有表的缺省值。

另请参见

- “setDefault 方法”一节第 107 页

COLUMN_DEFAULT_CURRENT_DATE 变量

指定列的缺省值为当前日期（年、月、日）。

语法

final byte **ColumnSchema.COLUMN_DEFAULT_CURRENT_DATE**

注释

请参见 SQL Anywhere 文档集中 "UltraLite 中的特殊值" 下的 "CURRENT DATE 特殊值"。

可通过查询系统表 TableSchema.SYS_COLUMNS 的 column_default 列来确定现有表的缺省值。

另请参见

- “setDefault 方法”一节第 107 页

COLUMN_DEFAULT_CURRENT_TIME 变量

指定列的缺省值为当前时间。

语法

final byte **ColumnSchema.COLUMN_DEFAULT_CURRENT_TIME**

注释

请参见 SQL Anywhere 文档集中 "UltraLite 中的特殊值" 下的 "CURRENT TIME 特殊值"。

可通过查询系统表 TableSchema.SYS_COLUMNS 的 column_default 列来确定现有表的缺省值。

另请参见

- “setDefault 方法”一节第 107 页

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

COLUMN_DEFAULT_CURRENT_TIMESTAMP 变量

指定列的缺省值为当前时间戳。

语法

final byte **ColumnSchema.COLUMN_DEFAULT_CURRENT_TIMESTAMP**

注释

此常量将 CURRENT DATE 和 CURRENT TIME 相结合，从而构成包含年月日、小时、分钟、秒和秒的小数值的 TIMESTAMP 值。小数值的精度设置为 3 个小数位。此常量的准确性受系统时钟准确性的限制。请参见 SQL Anywhere 文档集中 "UltraLite 中的特殊值" 下的 "CURRENT TIMESTAMP 特殊值"。

可通过查询系统表 TableSchema.SYS_COLUMNS 的 column_default 列来确定现有表的缺省值。

另请参见

- "setDefault 方法" 一节第 107 页

COLUMN_DEFAULT_GLOBAL_AUTOINC 变量

指定列全局自动递增。

语法

final byte **ColumnSchema.COLUMN_DEFAULT_GLOBAL_AUTOINC**

注释

此常量类似于 AUTOINCREMENT，只是要对域进行分区。每个分区都包含相同数目的值。必须为每个数据库副本指定一个唯一全局数据库标识号。UltraLiteJ 从用数据库编号唯一标识的分区中提供数据库中的缺省值。

可通过查询系统表 TableSchema.SYS_COLUMNS 的 column_default 列来确定现有表的缺省值。

另请参见

- "setDefault 方法" 一节第 107 页
- "setDatabaseId 方法" 一节第 140 页

COLUMN_DEFAULT_NONE 变量

指定列没有任何特殊缺省值。

语法

final byte **ColumnSchema.COLUMN_DEFAULT_NONE**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

注释

可为空的列缺省为空值，不可为空的数字列缺省为 0，而不可为空的变长列缺省为零长度值。

可通过查询系统表 TableSchema.SYS_COLUMNS 的 column_default 列来确定现有表的缺省值。

另请参见

- “setDefault 方法”一节第 107 页
- “setNullable 方法”一节第 108 页

COLUMN_DEFAULT_UNIQUE_ID 变量

指定列缺省为新的唯一标识符。

语法

final byte **ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID**

注释

UUID 可用于唯一地标识表中的行。生成的值在所有计算机或设备上都唯一，即在同步和复制环境中可将这些值作为键来使用。

可通过查询系统表 TableSchema.SYS_COLUMNS 的 column_default 列来确定现有表的缺省值。

另请参见

- “setDefault 方法”一节第 107 页

setDefault 方法

设置列的缺省值。

语法

ColumnSchema **ColumnSchema.setDefault(**
  byte *default_code*
)

参数

- **default_code**　ColumnSchema 代码之一，并且是具有 COLUMN_DEFAULT 后缀的常量，用于指示列应具有的缺省值类型。

注释

缺省值为 COLUMN_DEFAULT_NONE。

另请参见

- "COLUMN_DEFAULT_AUTOINC 变量"一节第 104 页
- "COLUMN_DEFAULT_CURRENT_DATE 变量"一节第 105 页
- "COLUMN_DEFAULT_CURRENT_TIME 变量"一节第 105 页
- "COLUMN_DEFAULT_CURRENT_TIMESTAMP 变量"一节第 106 页
- "COLUMN_DEFAULT_GLOBAL_AUTOINC 变量"一节第 106 页
- "COLUMN_DEFAULT_NONE 变量"一节第 106 页
- "COLUMN_DEFAULT_UNIQUE_ID 变量"一节第 107 页

返回值

已定义了缺省值的 ColumnSchema。

setNullable 方法

将列设置成可为空。

语法

```
ColumnSchema ColumnSchema.setNullable(
  boolean nullable
)
```

参数

- **nullable**　如果此列可接受空值，则设置为 true；否则设置为 false。

注释

主键和唯一键中的列始终不可为空。缺省情况下，BIT 类型的列不可为空。

返回值

定义了可为空列的 ColumnSchema。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

ConfigFile 接口

建立保存在文件中的持久数据库的配置。

语法

public **ConfigFile**

基类

- “Configuration 接口”一节第 120 页
- “ConfigPersistent 接口”一节第 112 页

成员

ConfigFile 的所有成员，包括所有继承的成员。

- “getAutoCheckpoint 方法”一节第 112 页
- “getCacheSize 方法”一节第 113 页
- “getDatabaseName 方法”一节第 120 页
- “getLazyLoadIndexes 方法”一节第 113 页
- “getPageSize 方法”一节第 120 页
- “hasPersistentIndexes 方法”一节第 113 页
- “setAutocheckpoint 方法”一节第 113 页
- “setCacheSize 方法”一节第 114 页
- “setEncryption 方法”一节第 114 页
- “setIndexPersistence 方法”一节第 115 页
- “setLazyLoadIndexes 方法”一节第 115 页
- “setPageSize 方法”一节第 121 页
- “setPassword 方法”一节第 121 页
- “setRowMaximumThreshold 方法”一节第 116 页
- “setRowMinimumThreshold 方法”一节第 116 页
- “setShadowPaging 方法”一节第 117 页
- “setWriteAtEnd 方法”一节第 117 页
- “writeAtEnd 方法”一节第 118 页

ConfigNonPersistent 接口

建立非持久性数据库的配置。

语法

public **ConfigNonPersistent**

基类

● “Configuration 接口”一节第 120 页

成员

ConfigNonPersistent 的所有成员，包括所有继承的成员。

● “getDatabaseName 方法”一节第 120 页
● “getPageSize 方法”一节第 120 页
● “setPageSize 方法”一节第 121 页
● “setPassword 方法”一节第 121 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

ConfigObjectStore 接口（仅限 J2ME BlackBerry）

建立保存在对象存储库中的持久数据库的配置。

语法

public **ConfigObjectStore**

基类

- “Configuration 接口”一节第 120 页
- “ConfigPersistent 接口”一节第 112 页

成员

ConfigObjectStore 的所有成员，包括所有继承的成员。

- “getAutoCheckpoint 方法”一节第 112 页
- “getCacheSize 方法”一节第 113 页
- “getDatabaseName 方法”一节第 120 页
- “getLazyLoadIndexes 方法”一节第 113 页
- “getPageSize 方法”一节第 120 页
- “hasPersistentIndexes 方法”一节第 113 页
- “setAutocheckpoint 方法”一节第 113 页
- “setCacheSize 方法”一节第 114 页
- “setEncryption 方法”一节第 114 页
- “setIndexPersistence 方法”一节第 115 页
- “setLazyLoadIndexes 方法”一节第 115 页
- “setPageSize 方法”一节第 121 页
- “setPassword 方法”一节第 121 页
- “setRowMaximumThreshold 方法”一节第 116 页
- “setRowMinimumThreshold 方法”一节第 116 页
- “setShadowPaging 方法”一节第 117 页
- “setWriteAtEnd 方法”一节第 117 页
- “writeAtEnd 方法”一节第 118 页

ConfigPersistent 接口

建立持久数据库的配置。

语法

public **ConfigPersistent**

基类

- “Configuration 接口”一节第 120 页

派生类

- “ConfigFile 接口”一节第 109 页
- “ConfigObjectStore 接口（仅限 J2ME BlackBerry）”一节第 111 页
- “ConfigRecordStore 接口（仅限 J2ME）”一节第 119 页

成员

ConfigPersistent 的所有成员，包括所有继承的成员。

- “getAutoCheckpoint 方法”一节第 112 页
- “getCacheSize 方法”一节第 113 页
- “getDatabaseName 方法”一节第 120 页
- “getLazyLoadIndexes 方法”一节第 113 页
- “getPageSize 方法”一节第 120 页
- “hasPersistentIndexes 方法”一节第 113 页
- “setAutocheckpoint 方法”一节第 113 页
- “setCacheSize 方法”一节第 114 页
- “setEncryption 方法”一节第 114 页
- “setIndexPersistence 方法”一节第 115 页
- “setLazyLoadIndexes 方法”一节第 115 页
- “setPageSize 方法”一节第 121 页
- “setPassword 方法”一节第 121 页
- “setRowMaximumThreshold 方法”一节第 116 页
- “setRowMinimumThreshold 方法”一节第 116 页
- “setShadowPaging 方法”一节第 117 页
- “setWriteAtEnd 方法”一节第 117 页
- “writeAtEnd 方法”一节第 118 页

getAutoCheckpoint 方法

确定自动检查点是否打开。

语法

boolean **ConfigPersistent.getAutoCheckpoint()**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

返回值

如果数据库打开了自动检查点，则为 true；否则为 false。

getCacheSize 方法

返回数据库的高速缓存大小（以字节为单位）。

语法

int **ConfigPersistent.getCacheSize()**

返回值

高速缓存大小。

getLazyLoadIndexes 方法

确定是否启用了延缓加载索引功能。

语法

boolean **ConfigPersistent.getLazyLoadIndexes()**

返回值

如果启用延缓加载，则为 true；否则为 false。

hasPersistentIndexes 方法

确定索引是否是持久的。

语法

boolean **ConfigPersistent.hasPersistentIndexes()**

返回值

如果索引是持久的，则为 true；否则为 false。

setAutocheckpoint 方法

设置自动检查点打开。

语法

ConfigPersistent **ConfigPersistent.setAutocheckpoint(**
 boolean *auto_checkpoint*
) throws **ULjException**

参数

- **auto_checkpoint**　若值为 true，则打开自动检查点。

注释

在提交的更改操作应用到持久存储区中的持久性行时，数据库启用自动检查点功能。自动检查点处于活动状态时，每次提交时都会出现检查点；否则，将编写一份事务记录以记录更改，并且在应用程序调用 checkpoint 方法之前不更改持久行存储区。

禁用自动检查点可使更改和提交操作运行得更快，但如果存在大量没有检查点的事务，将使数据库启动变慢。

如果索引不是持久的或者启用了行限制，则 Autocheckpoint 始终为 true。

返回值

设置了自动检查点的 ConfigPersistent。

setCacheSize 方法

设置数据库的高速缓存大小（以字节为单位）。

语法

ConfigPersistent **ConfigPersistent.setCacheSize(**
  int *cache_size*
) throws **ULjException**

参数

- **cache_size**　高速缓存大小。所有平台上的缺省高速缓存大小均为 20480 (20KB)。

注释

高速缓存大小决定了驻留在页面高速缓存中的数据库页面数目。增加大小则意味着读取和写入数据库页面的次数减少，但要延长在高速缓存中查找页面所需的时间。

返回值

设置了高速缓存大小的 ConfigPersistent。

setEncryption 方法

设置加密。

语法

ConfigPersistent **ConfigPersistent.setEncryption(**
  EncryptionControl *control*
)

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

参数

- **control**　用于加密数据库的 EncryptionControl 对象。

返回值

设置了加密的 ConfigPersistent。

setIndexPersistence 方法

启用持久索引。

语法

```
ConfigPersistent ConfigPersistent.setIndexPersistence(
  boolean store
) throws ULjException
```

参数

- **store**　若要存储索引，设置为 true；否则设置为 false，以便在第一次使用索引之间建立索引。

注释

此设置仅在创建数据库时使用。它确定要对数据库使用哪种索引持久性策略。

打开现有数据库时，将使用创建时的设置，并更新配置以反映该值。

返回值

设置了索引持久性的 ConfigPersistent。

setLazyLoadIndexes 方法

将索引设置为在需要时装载，或设置为在启动时一次性装载全部索引。

语法

```
ConfigPersistent ConfigPersistent.setLazyLoadIndexes(
  boolean lazy_load
) throws ULjException
```

参数

- **lazy_load**　要根据需要装载索引，设置为 true；否则设置为 false，以便在启动时一次性装载全部索引。

注释

启用此选项可减少数据库的启动时间，但以后操作的执行速度可能较慢。

返回值

设置了延缓加载索引的 ConfigPersistent。

setRowMaximumThreshold 方法

设置可保留在内存中的最大行数的阈值。

语法

ConfigPersistent **ConfigPersistent.setRowMaximumThreshold(**
  int *threshold*
)

参数

- **threshold**　最大阈值。

注释

达到最大行数时，行将被截断，并保留 setRowMinimumThreshold 方法所定义的最小行数。

另请参见

- “setRowMinimumThreshold 方法”一节第 116 页

返回值

设置了最大阈值的 ConfigPersistent。

setRowMinimumThreshold 方法

设置可保留在内存中的最小行数的阈值。

语法

ConfigPersistent **ConfigPersistent.setRowMinimumThreshold(**
  int *threshold*
)

参数

- **threshold**　最小阈值。

注释

达到最大行数时，行将被截断，并保留 setRowMinimumThreshold 方法所定义的最小行数。

另请参见

- “setRowMaximumThreshold 方法”一节第 116 页

返回值

设置了最小阈值的 ConfigPersistent。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

setShadowPaging 方法

启用影子分页。

语法

ConfigPersistent **ConfigPersistent.setShadowPaging(**
  boolean *shadow*
) throws **ULjException**

参数

- **shadow** 要启用影子分页，设置为 true；否则设置为 false。

注释

影子分页是指写入持久存储区的所有内容仅出现在未使用的数据库页面，在提交操作完成之前这些页面不会永久存储。这能够确保所有已提交的更改都永久保存，即使应用程序异常终止。

如果将影子分页设置为 false，则在更改操作已经发生但尚未提交的情况下可能会损坏数据库。

而持续不使用影子分页则意味着数据库操作的处理速度更快，并且返回的数据库结果较小。

只有在数据不重要或可以通过同步恢复时，才可以在不启用影子分页的情况下处理数据库。

返回值

设置了 ShadowPaging 的 ConfigPersistent。

setWriteAtEnd 方法

启用关闭过程中的索引持久性。

语法

ConfigPersistent **ConfigPersistent.setWriteAtEnd(**
  boolean *write_at_end*
) throws **ULjException**

参数

- **write_at_end** 设置为 true 可将数据库保留在内存中，直到关闭。

注释

启用此选项可加快数据库操作的速度，但如果应用程序异常终止，对数据库的所有更改都将丢失。

只有在数据不重要或可以通过同步恢复时，才可以在启用索引持久性的情况下处理数据库。

返回值

设置了 WriteAtEnd 的 ConfigPersistent。

writeAtEnd 方法

确定是否启用了关闭过程中的索引持久性。

语法

boolean **ConfigPersistent.writeAtEnd()**

返回值

如果启用了关闭过程中的索引持久性，则为 true；否则为 false。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

ConfigRecordStore 接口（仅限 J2ME）

建立保存在 J2ME 记录存储区中的持久数据库的配置。

语法

public **ConfigRecordStore**

基类

- “Configuration 接口”一节第 120 页
- “ConfigPersistent 接口”一节第 112 页

成员

ConfigRecordStore 的所有成员，包括所有继承的成员。

- “getAutoCheckpoint 方法”一节第 112 页
- “getCacheSize 方法”一节第 113 页
- “getDatabaseName 方法”一节第 120 页
- “getLazyLoadIndexes 方法”一节第 113 页
- “getPageSize 方法”一节第 120 页
- “hasPersistentIndexes 方法”一节第 113 页
- “setAutocheckpoint 方法”一节第 113 页
- “setCacheSize 方法”一节第 114 页
- “setEncryption 方法”一节第 114 页
- “setIndexPersistence 方法”一节第 115 页
- “setLazyLoadIndexes 方法”一节第 115 页
- “setPageSize 方法”一节第 121 页
- “setPassword 方法”一节第 121 页
- “setRowMaximumThreshold 方法”一节第 116 页
- “setRowMinimumThreshold 方法”一节第 116 页
- “setShadowPaging 方法”一节第 117 页
- “setWriteAtEnd 方法”一节第 117 页
- “writeAtEnd 方法”一节第 118 页

Configuration 接口

建立数据库的配置。

语法

public **Configuration**

派生类

- “ConfigNonPersistent 接口”一节第 110 页
- “ConfigPersistent 接口”一节第 112 页

注释

有些属性仅在创建数据库时使用，而有些属性则适用于和数据库的初始连接。在创建数据库或连接到数据库之后设置的属性将被忽略。

成员

Configuration 的所有成员，包括所有继承的成员。

- “getDatabaseName 方法”一节第 120 页
- “getPageSize 方法”一节第 120 页
- “setPageSize 方法”一节第 121 页
- “setPassword 方法”一节第 121 页

getDatabaseName 方法

返回数据库名。

语法

String **Configuration.getDatabaseName()**

返回值

数据库的名称。

getPageSize 方法

返回数据库的页大小（以字节为单位）。

语法

int **Configuration.getPageSize()**

返回值

页面大小。

setPageSize 方法

设置数据库的页大小（以字节为单位）。

语法

Configuration **Configuration.setPageSize(**
  int *page_size*
) throws **ULjException**

参数

- **page_size**　页面大小。

注释

页面大小设置用于确定存储在持久数据库中的行的大小上限。它也确定了索引页面的大小，以及每个此类页面可含有的子项数目。

使用现有数据库时，该大小已经设置为数据库创建时的页面大小。无法使用此方法重置现有数据库的页面大小。

页面大小的范围是 256 到 32736 字节。缺省值为 1024 个字节。

返回值

设置了页面大小的 Configuration 对象。

setPassword 方法

设置数据库口令。

语法

Configuration **Configuration.setPassword(**
  String *password*
) throws **ULjException**

参数

- **password**　用于新数据库的口令，或用于访问现有数据库的口令。

注释

口令用于获取对数据库的访问权限，并且必须与创建数据库时指定的口令相一致。缺省值为 "dba"。

返回值

设置了口令的 Configuration 对象。

Connection 接口

描述数据库连接，启动数据库操作需要此连接。

语法

public **Connection**

注释

使用 DatabaseManager 类的 connect 或 createDatabase 方法可获得连接。如果不再需要连接，可使用 release 方法来释放连接。数据库的所有连接都被释放时，数据库也随之关闭。

连接提供以下功能：

- 创建新模式（表、索引和发布）
- 创建新值和域对象
- 将更改永久提交到数据库
- 准备要执行的 SQL 语句
- 回退对数据库进行的未提交更改
- 对数据库进行检查点操作（用已提交更改更新基础持久存储区，而不是仅仅存储更改事务）。

以下示例演示了如何创建简单数据库模式。数据库包含两个表，分别为表 T1（仅具有名为 num 的整数主键列）和表 T2（具有名为 num 的整数主键列和名为 quantity 的整数列）。T2 在 quantity 列上具有额外索引。T1 包含在名为 PubA 的发布之中。

```
table_schema = conn.createTable("T1");
 table_schema.createColumn("num", Domain.INTEGER);
```


Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

成员

Connection 的所有成员，包括所有继承的成员。

- “checkpoint 方法”一节第 127 页
- “commit 方法”一节第 128 页
- “CONNECTED 变量”一节第 124 页
- “createDecimalNumber 方法”一节第 128 页
- “createDecimalNumber 方法”一节第 128 页
- “createDomain 方法”一节第 129 页
- “createDomain 方法”一节第 129 页
- “createDomain 方法”一节第 129 页
- “createForeignKey 方法”一节第 130 页
- “createPublication 方法”一节第 130 页
- “createSyncParms 方法”一节第 131 页
- “createSyncParms 方法”一节第 131 页
- “createTable 方法”一节第 132 页
- “createUUIDValue 方法”一节第 132 页
- “createValue 方法”一节第 133 页
- “disableSynchronization 方法”一节第 133 页
- “dropDatabase 方法”一节第 133 页
- “dropForeignKey 方法”一节第 134 页
- “dropPublication 方法”一节第 134 页
- “dropTable 方法”一节第 134 页
- “emergencyShutdown 方法”一节第 135 页
- “enableSynchronization 方法”一节第 135 页
- “getDatabaseId 方法”一节第 135 页
- “getDatabaseInfo 方法”一节第 136 页
- “getDatabasePartitionSize 方法”一节第 136 页
- “getDatabaseProperty 方法”一节第 136 页
- “getLastDownloadTime 方法”一节第 137 页
- “getOption 方法”一节第 137 页
- “getState 方法”一节第 138 页
- “NOT_CONNECTED 变量”一节第 124 页
- “OPTION_DATABASE_ID 变量”一节第 124 页
- “OPTION_DATE_FORMAT 变量”一节第 124 页
- “OPTION_DATE_ORDER 变量”一节第 125 页
- “OPTION_ML_REMOTE_ID 变量”一节第 125 页
- “OPTION_NEAREST_CENTURY 变量”一节第 125 页
- “OPTION_PRECISION 变量”一节第 125 页
- “OPTION_SCALE 变量”一节第 125 页
- “OPTION_TIME_FORMAT 变量”一节第 126 页
- “OPTION_TIMESTAMP_FORMAT 变量”一节第 126 页
- “OPTION_TIMESTAMP_INCREMENT 变量”一节第 126 页
- “prepareStatement 方法”一节第 138 页
- “PROPERTY_DATABASE_NAME 变量”一节第 126 页
- “PROPERTY_PAGE_SIZE 变量”一节第 126 页
- “release 方法”一节第 139 页

- “renameTable 方法”一节第 139 页
- “resetLastDownloadTime 方法”一节第 139 页
- “rollback 方法”一节第 140 页
- “schemaCreateBegin 方法”一节第 140 页
- “schemaCreateComplete 方法”一节第 140 页
- “setDatabaseId 方法”一节第 140 页
- “setOption 方法”一节第 141 页
- “startSynchronizationDelete 方法”一节第 142 页
- “stopSynchronizationDelete 方法”一节第 142 页
- “SYNC_ALL 变量”一节第 126 页
- “SYNC_ALL_DB_PUB_NAME 变量”一节第 127 页
- “SYNC_ALL_PUBS 变量”一节第 127 页
- “synchronize 方法”一节第 142 页
- “truncateTable 方法”一节第 142 页

CONNECTED 变量

已连接状态。

语法

final byte **Connection.CONNECTED**

NOT_CONNECTED 变量

未连接状态。

语法

final byte **Connection.NOT_CONNECTED**

OPTION_DATABASE_ID 变量

数据库选项：数据库 ID。

语法

final String **Connection.OPTION_DATABASE_ID**

OPTION_DATE_FORMAT 变量

数据库选项：日期格式。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

final String **Connection.OPTION_DATE_FORMAT**

OPTION_DATE_ORDER 变量

数据库选项：日期顺序。

语法

final String **Connection.OPTION_DATE_ORDER**

OPTION_ML_REMOTE_ID 变量

数据库选项：ML 远程 ID。

语法

final String **Connection.OPTION_ML_REMOTE_ID**

注释

OPTION_NEAREST_CENTURY 变量

数据库选项：最接近的世纪值。

语法

final String **Connection.OPTION_NEAREST_CENTURY**

OPTION_PRECISION 变量

数据库选项：精度。

语法

final String **Connection.OPTION_PRECISION**

OPTION_SCALE 变量

数据库选项：小数位数。

语法

final String **Connection.OPTION_SCALE**

OPTION_TIMESTAMP_FORMAT 变量

数据库选项：时间戳格式。

语法

final String **Connection.OPTION_TIMESTAMP_FORMAT**

OPTION_TIMESTAMP_INCREMENT 变量

表示 timestamp_increment 数据库选项的常量。该选项限制时间戳值的精度。向数据库中插入时间戳时，UltraLiteJ 会截断时间戳，使之与此增量匹配。允许的值是 1 到 60000000 微秒。缺省值为 1（注意 1000000 微秒等于 1 秒）。

语法

final String **Connection.OPTION_TIMESTAMP_INCREMENT**

OPTION_TIME_FORMAT 变量

数据库选项：时间格式。

语法

final String **Connection.OPTION_TIME_FORMAT**

PROPERTY_DATABASE_NAME 变量

数据库属性：数据库名称。

语法

final String **Connection.PROPERTY_DATABASE_NAME**

PROPERTY_PAGE_SIZE 变量

数据库属性：页面大小。

语法

final String **Connection.PROPERTY_PAGE_SIZE**

SYNC_ALL 变量

用于请求同步数据库中所有表（包括不在任何发布中的表）的发布列表。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

final String **Connection.SYNC_ALL**

注释

绝不会同步标记为 NoSync 的表。

此常量等同于空值引用或空字符串。

SYNC_ALL_DB_PUB_NAME 变量

SYNC_ALL_DB 发布的保留名称。

语法

final String **Connection.SYNC_ALL_DB_PUB_NAME**

另请参见

- "getLastDownloadTime 方法"一节第 137 页
- "resetLastDownloadTime 方法"一节第 139 页

SYNC_ALL_PUBS 变量

用于请求同步数据库中所有发布的发布列表。

语法

final String **Connection.SYNC_ALL_PUBS**

注释

绝不会同步标记为 NoSync 的表。

checkpoint 方法

对数据库更改执行检查点操作。

语法

void **Connection.checkpoint()** throws **ULjException**

注释

调用此函数会将所有已提交事务应用于数据库的持久版本。

commit 方法

提交数据库更改。

语法

void **Connection.commit()** throws **ULjException**

注释

调用此方法可使自上次提交或回退以来的所有数据库更改都成为永久更改。

createDecimalNumber 方法

创建 DecimalNumber。

语法

DecimalNumber **Connection.createDecimalNumber(**
  int *precision*,
  int *scale*
) throws **ULjException**

参数

- **precision** 数字中的位数。
- **scale** 数字中的小数位数。

返回值

具有指定类型的 DecimalNumber。

createDecimalNumber 方法

创建 DecimalNumber。

语法

DecimalNumber **Connection.createDecimalNumber(**
  int *precision*,
  int *scale*,
  String *value*
) throws **ULjException**

参数

- **precision** 数字中的位数。
- **scale** 数字中的小数位数。
- **value** 要设置的值。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

返回值

具有指定类型的 DecimalNumber。

createDomain 方法

创建固定大小的域。

语法

Domain **Connection.createDomain(**
  int *type*
) throws **ULjException**

参数

- **type**　域类型。

返回值

具有给定类型的域。

createDomain 方法

创建可变大小的域。

语法

Domain **Connection.createDomain(**
  int *type*,
  int *size*
) throws **ULjException**

参数

- **type**　域类型。
- **size**　域大小。

返回值

具有给定类型的域。

createDomain 方法

创建可变大小的域。

语法

Domain **Connection.createDomain(**
  int *type*,
  int *size*,

```
  int scale
) throws ULjException
```

参数

- **type**　域类型。
- **size**　域大小。
- **scale**　域的小数位数。

返回值

具有给定类型的域。

createForeignKey 方法

创建新的外键。

语法

```
ForeignKeySchema Connection.createForeignKey(
  String table_name,
  String primary_table_name,
  String name
) throws ULjException
```

参数

- **table_name**　包含外键的表的名称。该表被限制为必须包括对主表的有效引用。
- **primary_table_name**　包含被引用列的表的名称。
- **name**　外键的名称。所指定的名称必须是有效的 SQL 标识符。

注释

> **注意**
> UltraLiteJ 不强制对表进行外键约束。外键用于确定对多个表进行同步时要遵循的正确顺序。客户端数据库上的外键应与客户端要与之同步的统一数据库中的关系相匹配。

返回值

定义了名称和相关表的 ForeignKey。

createPublication 方法

在数据库中创建新发布。

语法

```
void Connection.createPublication(
  String pub_name,
```


Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
  String[] tables
) throws ULjException
```

参数

- **pub_name**　要创建的发布的名称。
- **tables**　表名称数组。

注释

使用特殊的 Connection.SYNC_ALL 发布列表同步整个数据库。

另请参见

- "dropPublication 方法" 一节第 134 页
- "setPublications 方法" 一节第 230 页
- "setTableOrder 方法" 一节第 231 页

createSyncParms 方法

为 HTTP 同步创建同步参数集。

语法

```
SyncParms Connection.createSyncParms(
  String userName,
  String version
) throws ULjException
```

参数

- **userName**　此客户端数据库的唯一 MobiLink 用户名。
- **version**　MobiLink 脚本版本。

另请参见

- "createSyncParms 方法" 一节第 131 页
- "setUserName 方法" 一节第 232 页

返回值

SyncParms 对象。

createSyncParms 方法

创建同步参数集。

语法

```
SyncParms Connection.createSyncParms(
  int streamType,
```

```
  String userName,
  String version
) throws ULjException
```

参数

- **streamType**　在 SyncParms 类中定义的一个常量，用于标识同步流的类型。
- **userName**　MobiLink 用户名。
- **version**　MobiLink 脚本版本。

另请参见

- “createSyncParms 方法”一节第 131 页
- “HTTP_STREAM 变量”一节第 221 页
- “HTTPS_STREAM 变量”一节第 220 页

返回值

SyncParms 对象

createTable 方法

在数据库中创建新表。

语法

```
TableSchema Connection.createTable(
  String table_name
) throws ULjException
```

参数

- **table_name**　要创建的表的名称。

注释

仅当连接的数据库处于 "模式创建" 模式时才能执行此方法。

返回值

新表的 TableSchema 对象。

另请参见

- “schemaCreateBegin 方法”一节第 140 页
- “schemaCreateComplete 方法”一节第 140 页

createUUIDValue 方法

创建 UUID 值。

语法

Value **Connection.createUUIDValue()** throws **ULjException**

返回值

域值。

createValue 方法

创建域中的值。

语法

Value **Connection.createValue(**
  Domain *dom*
) throws **ULjException**

参数

- **dom**　给定的域。

返回值

域值。

disableSynchronization 方法

禁用表同步。

语法

void **Connection.disableSynchronization(**
  String *table_name*
) throws **ULjException**

参数

- **table_name**　表的名称。

dropDatabase 方法

删除数据库。

语法

void **Connection.dropDatabase()** throws **ULjException**

注释

连接所引用的数据库被消除，连接被释放。对于正在删除的数据库，只有此连接可以处于活动状态。

dropForeignKey 方法

删除外键。

语法

void **Connection.dropForeignKey(**
  String *table_name*,
  String *fkey_name*
) throws **ULjException**

参数

- **table_name**　包含外键的表的名称。
- **fkey_name**　要删除的外键的名称。

dropPublication 方法

从数据库中删除发布。

语法

void **Connection.dropPublication(**
  String *pub_name*
) throws **ULjException**

参数

- **pub_name**　要删除的发布的名称。

注释

不能删除特殊 Connection.SYNC_ALL_DB_PUB_NAME 发布。

另请参见

- “createPublication 方法”一节第 130 页

dropTable 方法

从数据库中删除表。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

void **Connection.dropTable(**
String *table_name*
) throws **ULjException**

参数

- **table_name** 要删除的表的名称。

注释

只有在当前连接没有任何未完成的未提交事务且该表未出现在任何发布中时，才能删除该表。删除的表中的所有行都将丢失。所有未同步的操作也会丢失。

emergencyShutdown 方法

紧急关闭连接的数据库。

语法

void **Connection.emergencyShutdown()** throws **ULjException**

注释

只有在出现严重错误的情况下才能调用此方法。仅应在物理硬件或数据损坏时使用。

此方法将关闭所有打开的连接，然后关闭连接的数据库。

enableSynchronization 方法

启用表同步。

语法

void **Connection.enableSynchronization(**
String *table_name*
) throws **ULjException**

参数

- **table_name** 表名。

getDatabaseId 方法

返回数据库 ID 值。

语法

int **Connection.getDatabaseId()** throws **ULjException**

返回值

数据库 ID。

getDatabaseInfo 方法

返回一个包含有关数据库属性信息的 DataInfo 对象。

语法

DatabaseInfo **Connection.getDatabaseInfo()** throws **ULjException**

返回值

DatabaseInfo 对象。

getDatabasePartitionSize 方法

返回数据库分区大小。

语法

int **Connection.getDatabasePartitionSize()** throws **ULjException**

返回值

数据库分区的大小。

另请参见

- “getDatabaseId 方法”一节第 135 页

getDatabaseProperty 方法

返回数据库的属性。

语法

String **Connection.getDatabaseProperty(**
  String *name*
) throws **ULjException**

参数

- **name** 数据库属性的名称。

返回值

与给定名称对应的属性的值。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

getLastDownloadTime 方法

返回指定发布的最近一次下载的时间。

语法

Date **Connection.getLastDownloadTime(**
String *pub_name*
) throws **ULjException**

参数

- **pub_name**　要检查的发布的名称。

注释

要返回全部数据库的上次下载时间，参数 pub_name 必须引用单个发布，或必须是特殊发布 Connection.SYNC_ALL_DB_PUB_NAME。

只有在执行 `schemaCreateComplete()` 之后才能执行此方法。

另请参见

- “createPublication 方法”一节第 130 页
- “resetLastDownloadTime 方法”一节第 139 页
- “schemaCreateBegin 方法”一节第 140 页
- “schemaCreateComplete 方法”一节第 140 页

返回值

上次下载的时间戳。

getOption 方法

返回数据库选项。

语法

String **Connection.getOption(**
String *option_name*
) throws **ULjException**

参数

- **option_name**　任意配置选项值，要检索其值的带有 OPTION 后缀的变量。

注释

数据库选项存储在数据库中（设置该选项后，可在随后某时连接数据库时获取这些选项）。

创建数据库时，也将创建一组必需的选项。

返回值

数据库选项的值。

另请参见

- “setOption 方法”一节第 141 页
- “OPTION_DATABASE_ID 变量”一节第 124 页
- “OPTION_DATE_FORMAT 变量”一节第 124 页
- “OPTION_DATE_ORDER 变量”一节第 125 页
- “OPTION_ML_REMOTE_ID 变量”一节第 125 页
- “OPTION_NEAREST_CENTURY 变量”一节第 125 页
- “OPTION_PRECISION 变量”一节第 125 页
- “OPTION_SCALE 变量”一节第 125 页
- “OPTION_TIMESTAMP_FORMAT 变量”一节第 126 页
- “OPTION_TIMESTAMP_INCREMENT 变量”一节第 126 页
- “OPTION_TIME_FORMAT 变量”一节第 126 页

getState 方法

返回连接的状态。

语法

byte **Connection.getState()** throws **ULjException**

注释

只有 Configuration 接口支持的返回语句才有效。

另请参见

- “CONNECTED 变量”一节第 124 页
- “NOT_CONNECTED 变量”一节第 124 页

prepareStatement 方法

准备将要执行的语句。

语法

PreparedStatement **Connection.prepareStatement(**
  String *sql*
) throws **ULjException**

参数

- **sql**　要准备的 SQL 语句。

另请参见

- “PreparedStatement 接口”一节第 176 页

返回值

PreparedStatement 对象。

release 方法

释放连接。

语法

void **Connection.release()** throws **ULjException**

注释

连接被释放后，便不能再用于访问数据库。

如果试图释放存在未提交事务的连接，将会出错。

renameTable 方法

为表重命名。

语法

void **Connection.renameTable(**
  String *old_table_name*,
  String *new_table_name*
) throws **ULjException**

参数

- **old_table_name**　现有表的名称。
- **new_table_name**　表的新名称。

resetLastDownloadTime 方法

重置指定发布的下载时间。

语法

void **Connection.resetLastDownloadTime(**
  String *pub_name*
) throws **ULjException**

参数

- **pub_name**　要检查的发布的名称。

注释

要重置同步整个数据库时的下载时间，请使用特殊的 Connection.SYNC_ALL_DB_PUB_NAME 发布。

此方法要求当前连接没有任何未提交事务。

另请参见

- “createPublication 方法”一节第 130 页

rollback 方法

提交回退操作，以撤销对数据库的更改。

语法

void **Connection.rollback()** throws **ULjException**

注释

调用此方法将撤销提交或回退操作后此连接上的所有数据库更改。

schemaCreateBegin 方法

使连接的数据库进入 "模式创建" 模式。

语法

void **Connection.schemaCreateBegin()** throws **ULjException**

注释

数据库处于 "模式创建" 模式时，无法进行数据和同步操作。进来的数据库连接也将被拒绝。

schemaCreateComplete 方法

使连接的数据库退出 "模式创建" 模式。

语法

void **Connection.schemaCreateComplete()** throws **ULjException**

setDatabaseId 方法

为全局自动增量设置数据库 ID 和分区大小。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

```
void Connection.setDatabaseId(
  int id,
  int size
) throws ULjException
```

参数

- **id**　数据库 ID。
- **size**　分区的大小。

setOption 方法

设置数据库选项。

语法

```
void Connection.setOption(
  String option_name,
  String option_value
) throws ULjException
```

参数

- **option_name**　任意配置选项值，要设置的带有 OPTION 后缀的变量。
- **option_value**　选项的新值。

注释

如果该选项目前未存储在数据库中，则创建它。

调用该方法时，此连接不能有任何未提交事务。

另请参见

- “getOption 方法”一节第 137 页
- “OPTION_DATABASE_ID 变量”一节第 124 页
- “OPTION_DATE_FORMAT 变量”一节第 124 页
- “OPTION_DATE_ORDER 变量”一节第 125 页
- “OPTION_ML_REMOTE_ID 变量”一节第 125 页
- “OPTION_NEAREST_CENTURY 变量”一节第 125 页
- “OPTION_PRECISION 变量”一节第 125 页
- “OPTION_SCALE 变量”一节第 125 页
- “OPTION_TIMESTAMP_FORMAT 变量”一节第 126 页
- “OPTION_TIMESTAMP_INCREMENT 变量”一节第 126 页
- “OPTION_TIME_FORMAT 变量”一节第 126 页

startSynchronizationDelete 方法

启动同步删除。

语法

void **Connection.startSynchronizationDelete()** throws **ULjException**

注释

此函数允许对任何将来的删除进行同步。

stopSynchronizationDelete 方法

停止同步删除。

语法

void **Connection.stopSynchronizationDelete()** throws **ULjException**

注释

这将使将来的所有删除操作都不能同步，除非随后执行 `startSynchronizationDelete`。

synchronize 方法

将数据库与 MobiLink 服务器同步。

语法

void **Connection.synchronize(**
  SyncParms *config*
) throws **ULjException**

参数

- **config**　用于同步的参数。

注释

将下载内容应用到数据库时，对数据库执行检查点操作。

另请参见

- “checkpoint 方法”一节第 127 页

truncateTable 方法

删除表中的所有行。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

```
void Connection.truncateTable(
  String table_name
) throws ULjException
```

参数

- **table_name**　要截断的表的名称。

注释

行不会被同步。

DatabaseInfo 接口

与 Connection 对象相关联，可提供用于显示数据库信息的方法。

语法

public **DatabaseInfo**

注释

可使用 Connection 对象的 getDatabaseInfo 方法调用此接口。

成员

DatabaseInfo 的所有成员，包括所有继承的成员。

- “getCommitCount 方法”一节第 144 页
- “getDbFormat 方法”一节第 144 页
- “getLogSize 方法”一节第 145 页
- “getNumberRowsToUpload 方法”一节第 145 页
- “getPageReads 方法”一节第 145 页
- “getPageSize 方法”一节第 145 页
- “getPageWrites 方法”一节第 146 页
- “getRelease 方法”一节第 146 页

getCommitCount 方法

返回对数据库执行的提交操作总数。

语法

int **DatabaseInfo.getCommitCount()**

返回值

提交操作的总数。

getDbFormat 方法

返回数据库版本号。

语法

int **DatabaseInfo.getDbFormat()**

返回值

版本号。

getLogSize 方法

返回事务日志的总体大小（以字节为单位）。

语法

int **DatabaseInfo.getLogSize()**

返回值

事务日志大小。

getNumberRowsToUpload 方法

返回等待上载的行数。

语法

int **DatabaseInfo.getNumberRowsToUpload()**

返回值

行数。

getPageReads 方法

返回创建 DatabaseInfo 对象时读取的页数。

语法

int **DatabaseInfo.getPageReads()**

返回值

读取的页数。

getPageSize 方法

返回数据库的页大小（以字节为单位）。

语法

int **DatabaseInfo.getPageSize()**

返回值

页面大小。

getRelease 方法

返回软件的发布版本号，以字符串表示。

语法

String **DatabaseInfo.getRelease()**

注释

如果软件的发布版本号为 7.1.3，则返回 "7.1.3"。

返回值

发布版本号。

getPageWrites 方法

返回创建 DatabaseInfo 对象时写入的页数。

语法

int **DatabaseInfo.getPageWrites()**

返回值

写入的页数。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

DatabaseManager 类

提供静态方法来获取基本配置、创建新数据库和连接到现有数据库。

语法

public **DatabaseManager**

注释

以下示例演示了如何打开现有数据库，以及在不存在现有数据库时如何创建新数据库。

此示例适用于 J2ME 设备：

```
Connection conn;
ConfigRecordStore config = DatabaseManager.createConfigurationRecordStore(
    "test.ulj"
    );
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

此示例适用于 J2ME BlackBerry 设备：

```
Connection conn;
ConfigObjectStore config = DatabaseManager.createConfigurationObjectStore(
    "test.ulj"
);
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

此示例适用于 J2SE 设备：

```
Connection conn;
ConfigFile config = DatabaseManager.createConfigurationFile(
    "test.ulj"
);
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

成员

DatabaseManager 的所有成员，包括所有继承的成员。

- “connect 方法”一节第 148 页
- “createConfigurationFile 方法”一节第 148 页
- “createConfigurationNonPersistent 方法”一节第 149 页
- “createConfigurationObjectStore 函数（仅限 J2ME BlackBerry）”一节第 149 页
- “createConfigurationRecordStore 函数（仅限 J2ME）”一节第 150 页
- “createDatabase 方法”一节第 150 页
- “createSISHTTPListener 函数（仅限 J2ME BlackBerry）”一节第 150 页
- “release 方法”一节第 151 页
- “setErrorLanguage 方法”一节第 151 页

connect 方法

根据某一配置连接到现有数据库，并返回连接。

语法

Connection **DatabaseManager.connect(**
  Configuration *config*
) throws **ULjException**

参数

- **config**　现有数据库的配置。

另请参见

- “Configuration 接口”一节第 120 页

返回值

与现有数据库的连接。

createConfigurationFile 方法

创建将文件用作物理存储区的配置，并返回 ConfigFile。

语法

ConfigFile **DatabaseManager.createConfigurationFile(**
  String *file_name*
) throws **ULjException**

参数

- **file_name**　要使用或创建的文件的名称。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见

- “ConfigFile 接口”一节第 109 页

返回值

用于配置数据库的 ConfigFile。

createConfigurationNonPersistent 方法

创建不带有持久存储区的配置，并返回 ConfigNonPersist。

语法

ConfigNonPersistent **DatabaseManager.createConfigurationNonPersistent(**
  String *db_name*
) throws **ULjException**

参数

- **db_name**　数据库的名称。

另请参见

- “ConfigNonPersistent 接口”一节第 110 页

返回值

用于配置数据库的 ConfigNonPersistent。

createConfigurationObjectStore 函数（仅限 J2ME BlackBerry）

创建将 RIM 对象存储区用作物理存储区的配置，并返回 ConfigObjectStore。

语法

ConfigObjectStore **createConfigurationObjectStore(**
String*db_name*
)

参数

- **db_name**　数据库的名称。

另请参见

- “ConfigObjectStore 接口（仅限 J2ME BlackBerry）”一节第 111 页

返回值

用于配置数据库的 ConfigObjectStore。

createConfigurationRecordStore 函数（仅限 J2ME）

创建将记录存储区用作物理存储区的配置，并返回 ConfigRecordStore。

语法

```
ConfigRecordStore DatabaseManager.createConfigurationRecordStore(
  String db_name
)
```

参数

- **db_name**　数据库的名称。

另请参见

- “ConfigRecordStore 接口（仅限 J2ME）”一节第 119 页

返回值

用于配置数据库的 ConfigRecordStore。

createDatabase 方法

根据某一配置创建新数据库，并返回连接。

语法

```
Connection DatabaseManager.createDatabase(
  Configuration config
) throws ULjException
```

参数

- **config**　新数据库的配置。

注释

此方法将替换任何具有相同名称的数据库。

另请参见

- “Configuration 接口”一节第 120 页

返回值

与新数据库的连接。

createSISHTTPListener 函数（仅限 J2ME BlackBerry）

为服务器启动的同步创建 HTTP SISListener。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

SISListener **DatabaseManager.createSISHTTPListener(**
 SISRequestHandler *handler, int port*
 String *httpOptions*
)

参数

- **handler** 指定的 SISRequestHandler，用于处理 SIS 请求。
- **port** 用于监听服务器消息的 HTTP 端口。
- **httpOptions** 用于连接到服务器的 HTTP 选项。

注释

建议的端口设置为 4400。

对于 BlackBerry 模拟器，建议的 HTTP 选项是 "deviceside=false"。

另请参见

- "SISListener 接口（仅限 J2ME BlackBerry）" 一节第 184 页

返回值

用于服务器启动的同步的 SISListener。

release 方法

关闭 DatabaseManager，以释放所有连接并关闭所有数据库。

语法

void **DatabaseManager.release()** throws **ULjException**

注释

回退任何未提交的事务。

setErrorLanguage 方法

设置错误消息所使用的语言。

语法

void **DatabaseManager.setErrorLanguage(**
 String *lang*
)

参数

- **lang** 以两个字符表示的语言代码。

注释

可识别的语言是 EN、DE、FR、JA、ZH。如果指定了无法识别的语言，系统将恢复为缺省设置。

在 J2SE 和 BlackBerry J2ME 环境下，当前区域设置用于确定缺省语言。在其它 J2ME 环境下，此为指定语言的唯一方法。缺省语言是 "EN"。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

DecimalNumber 接口

描述精确的十进制值，并为 java.math.BigDecimal 不可用的 Java 平台提供十进制算术支持。

语法

public **DecimalNumber**

成员

DecimalNumber 的所有成员，包括所有继承的成员。

- “add 方法”一节第 153 页
- “divide 方法”一节第 153 页
- “getString 方法”一节第 154 页
- “isNull 方法”一节第 154 页
- “multiply 方法”一节第 154 页
- “set 方法”一节第 155 页
- “setNull 方法”一节第 155 页
- “subtract 方法”一节第 155 页

add 方法

将两个 DecimalNumber 相加并返回得出的和。

语法

```
DecimalNumber DecimalNumber.add(
  DecimalNumber num1,
  DecimalNumber num2
) throws ULjException
```

参数

- **num1**　第一个数。
- **num2**　第二个数。

返回值

num1 和 num2 之和。

divide 方法

将第一个 DecimalNumber 除以第二个 DecimalNumber，并返回商。

语法

```
DecimalNumber DecimalNumber.divide(
  DecimalNumber num1,
```

DecimalNumber *num2*
) throws **ULjException**

参数

- **num1**　被除数。
- **num2**　除数。

返回值

num1 除以 num2 所得的商。

getString 方法

返回 DecimalNumber 的字符串表示形式。

语法

String **DecimalNumber.getString()** throws **ULjException**

返回值

字符串值。

isNull 方法

确定 DecimalNumber 是否为空。

语法

boolean **DecimalNumber.isNull()**

返回值

如果对象为空，则返回 true，否则返回 false。

multiply 方法

将两个 DecimalNumber 相乘并返回得出的乘积。

语法

DecimalNumber **DecimalNumber.multiply(**
DecimalNumber *num1*,
DecimalNumber *num2*
) throws **ULjException**

参数

- **num1**　被乘数。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

- **num2** 乘数。

返回值

num1 和 num2 之积。

set 方法

用字符串值设置 DecimalNumber。

语法

void **DecimalNumber.set(**
 String *value*
) throws **ULjException**

参数

- **value** 字符串形式的数字值。

setNull 方法

将 DecimalNumber 设置为空。

语法

void **DecimalNumber.setNull()** throws **ULjException**

subtract 方法

从第一个 DecimalNumber 中减去第二个 DecimalNumber，并返回差值。

语法

DecimalNumber **DecimalNumber.subtract(**
 DecimalNumber *num1*,
 DecimalNumber *num2*
) throws **ULjException**

参数

- **num1** 被减数。
- **num2** 减数。

返回值

num1 和 num2 之差。

Domain 接口

描述表中某一列的域类型信息。

语法

public **Domain**

注释

此接口包含若干用于表示各种域的常量，以及用于从 Domain 对象中抽取信息的方法。

以下示例演示了如何创建简单数据库模式。所创建的 T2 表具有一个整数列，以及一个最大长度为 32 字节的可变长度字符串列。

```
// Assumes a valid Connection object conn
TableSchema table_schema;
IndexSchema index_schema;

table_schema = conn.createTable("T2");
table_schema.createColumn("num", Domain.INTEGER);
table_schema.createColumn("name", Domain.VARCHAR, 32);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("num", IndexSchema.ASCENDING);
```

整数类型

域常量	SQL 类型	值范围
BIT	BIT	0 或 1
TINY	TINYINT	0 到 255（占用 1 个字节存储空间的无符号整数）
SHORT	SMALLINT	-32768 到 32767（占用 2 个字节存储空间的有符号整数）
UNSIGNED_SHORT	UNSIGNED SMALLINT	0 到 65535（占用 2 个字节存储空间的无符号整数）
INTEGER	INTEGER	-231 到 231 - 1，或 -2147483648 到 2147483647（占用 4 个字节存储空间的有符号整数）
UNSIGNED_INTEGER	UNSIGNED INTEGER	0 到 232 - 1，或 0 到 4294967295（占用 4 个字节存储空间的无符号整数）
BIG	BIGINT	-263 到 263 - 1，或 -9223372036854775808 到 9223372036854775807（占用 8 个字节存储空间的有符号整数）
UNSIGNED_BIG	UNSIGNED BIGINT	0 到 264 - 1，或 0 到 18446744073709551615（占用 8 个字节存储空间的无符号整数）

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

非整数数字类型

域常量	SQL 类型	值范围
REAL	REAL	-3.402823e+38 到 3.402823e+38，最趋近于零的数为 1.175495e-38（占用 4 个字节存储空间的单精度浮点数，在第六位之后可能产生舍入误差。）
DOUBLE	DOUBLE	-1.79769313486231e+308 到 1.79769313486231e+308，最趋近于零的数为 2.22507385850721e-308（占用 8 个字节存储空间的双精度浮点数，在第十五位之后可能产生舍入误差。）
NUMERIC	NUMERIC (precision, scale)	任何十进制数字，同时标示出精度（大小）总位数和小数点后的小数位数（在精度范围内无舍入）

字符和二进制类型

域常量	SQL 类型	大小范围
VARCHAR	VARCHAR(size)	1 到 32767 个字符（存储为 1 - 3 字节的 UTF-8 字符）。计算表达式时，临时字符值的最大长度为 2048 个字符。
LONGVARCHAR	LONG VARCHAR	（内存允许范围之内的）任意长度。允许对 LONG VARCHAR 列执行的操作只有插入、更新或删除，或将其包含在查询的选择列表中。
BINARY	BINARY(size)	1 到 32767 个字节。计算表达式时，临时字符值的最大长度为 2048 字节。
LONGBINARY	LONG BINARY	（内存允许范围之内的）任意长度。允许对 LONG BINARY 列执行的操作只有插入、更新或删除，或将其包含在查询的选择列表中。
UUID	UNIQUEIDENTIFIER	始终为具有特殊解释的 16 字节二进制数。

日期和时间类型

域常量	SQL 类型	值
DATE	DATE	年、月、日。
TIME	TIME	小时、分钟、秒和秒的小数值。
TIMESTAMP	TIMESTAMP	DATE 和 TIME。

缺省情况下，BIT 列不可为空。而所有其它类型在缺省情况下都可为空。

成员

Domain 的所有成员，包括所有继承的成员。

- “BIG 变量”一节第 159 页
- “BINARY 变量”一节第 159 页
- “BINARY_DEFAULT 变量”一节第 159 页
- “BINARY_MAX 变量”一节第 159 页
- “BINARY_MIN 变量”一节第 159 页
- “BIT 变量”一节第 160 页
- “CHARACTER_MAX 变量”一节第 160 页
- “DATE 变量”一节第 160 页
- “DOMAIN_MAX 变量”一节第 160 页
- “DOUBLE 变量”一节第 160 页
- “getName 方法”一节第 167 页
- “getPrecision 方法”一节第 167 页
- “getScale 方法”一节第 167 页
- “getSize 方法”一节第 167 页
- “getType 方法”一节第 168 页
- “INTEGER 变量”一节第 161 页
- “LONGBINARY 变量”一节第 161 页
- “LONGBINARY_DEFAULT 变量”一节第 161 页
- “LONGBINARY_MIN 变量”一节第 161 页
- “LONGVARCHAR 变量”一节第 162 页
- “LONGVARCHAR_DEFAULT 变量”一节第 162 页
- “LONGVARCHAR_MIN 变量”一节第 162 页
- “NUMERIC 变量”一节第 162 页
- “PRECISION_DEFAULT 变量”一节第 162 页
- “PRECISION_MAX 变量”一节第 163 页
- “PRECISION_MIN 变量”一节第 163 页
- “REAL 变量”一节第 163 页
- “SCALE_DEFAULT 变量”一节第 163 页
- “SCALE_MAX 变量”一节第 163 页
- “SCALE_MIN 变量”一节第 164 页
- “SHORT 变量”一节第 164 页
- “TIME 变量”一节第 164 页
- “TIMESTAMP 变量”一节第 164 页
- “TINY 变量”一节第 164 页
- “UINT16_MAX 变量”一节第 165 页
- “UNSIGNED_BIG 变量”一节第 165 页
- “UNSIGNED_INTEGER 变量”一节第 165 页
- “UNSIGNED_SHORT 变量”一节第 165 页
- “UUID 变量”一节第 166 页
- “VARCHAR 变量”一节第 166 页
- “VARCHAR_DEFAULT 变量”一节第 166 页
- “VARCHAR_MIN 变量”一节第 166 页

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

BIG 变量

64 位整数（SQL 类型 BIGINT）的域 ID 常量。

语法

final short **Domain.BIG**

另请参见

- “Domain 接口”一节第 156 页

BINARY 变量

最大字节数为 *size* 的可变长度二进制对象（SQL 类型 BINARY(*size*)）的域 ID 常量。

语法

final short **Domain.BINARY**

另请参见

- “Domain 接口”一节第 156 页

BINARY_DEFAULT 变量

二进制类型的缺省大小。

语法

final short **Domain.BINARY_DEFAULT**

BINARY_MAX 变量

二进制类型的大小上限。

语法

final short **Domain.BINARY_MAX**

BINARY_MIN 变量

二进制类型的大小下限。

语法

final short **Domain.BINARY_MIN**

BIT 变量

位（SQL 类型 BIT）的域 ID 常量。

语法

final short **Domain.BIT**

注释

缺省情况下，BIT 列不可为空。

另请参见

- “Domain 接口”一节第 156 页

CHARACTER_MAX 变量

字符类型的大小上限。

语法

final short **Domain.CHARACTER_MAX**

DATE 变量

日期（SQL 类型 DATE）的域 ID 常量。

语法

final short **Domain.DATE**

另请参见

- “Domain 接口”一节第 156 页

DOMAIN_MAX 变量

域类型的最多种类数。

语法

final short **Domain.DOMAIN_MAX**

DOUBLE 变量

8 字节浮点数（SQL 类型 DOUBLE）的域 ID 常量。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

final short **Domain.DOUBLE**

另请参见

- “Domain 接口”一节第 156 页

INTEGER 变量

32 位整数（SQL 类型 INTEGER）的域 ID 常量。

语法

final short **Domain.INTEGER**

另请参见

- “Domain 接口”一节第 156 页

LONGBINARY 变量

任意长度的二进制数据块 (BLOB)（SQL 类型 LONG BINARY）的域 ID 常量。

语法

final short **Domain.LONGBINARY**

另请参见

- “Domain 接口”一节第 156 页

LONGBINARY_DEFAULT 变量

BLOB 类型的缺省大小。

语法

final short **Domain.LONGBINARY_DEFAULT**

LONGBINARY_MIN 变量

BLOB 类型的大小下限。

语法

final short **Domain.LONGBINARY_MIN**

LONGVARCHAR 变量

任意长度的字符数据块 (CLOB)（SQL 类型 LONG VARCHAR）的域 ID 常量。

语法

final short **Domain.LONGVARCHAR**

另请参见

- “Domain 接口”一节第 156 页

LONGVARCHAR_DEFAULT 变量

CLOB 类型的缺省大小。

语法

final short **Domain.LONGVARCHAR_DEFAULT**

LONGVARCHAR_MIN 变量

CLOB 类型的大小下限。

语法

final short **Domain.LONGVARCHAR_MIN**

NUMERIC 变量

具有固定精度（大小）总位数，并且小数点后有 *scale* 位小数的数字值（SQL 类型 NUMERIC(*precision,scale*)）的域 ID 常量。

语法

final short **Domain.NUMERIC**

另请参见

- “Domain 接口”一节第 156 页

PRECISION_DEFAULT 变量

数字精度的缺省大小。

语法

final short **Domain.PRECISION_DEFAULT**

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

PRECISION_MAX 变量

数字精度的大小上限。

语法

final short **Domain.PRECISION_MAX**

PRECISION_MIN 变量

数字精度的大小下限。

语法

final short **Domain.PRECISION_MIN**

REAL 变量

4 字节浮点数（SQL 类型 REAL）的域 ID 常量。

语法

final short **Domain.REAL**

另请参见

- “Domain 接口”一节第 156 页

SCALE_DEFAULT 变量

数字小数位数的缺省大小。

语法

final short **Domain.SCALE_DEFAULT**

SCALE_MAX 变量

数字小数位数的大小上限。

语法

final short **Domain.SCALE_MAX**

SCALE_MIN 变量

数字小数位数的大小下限。

语法

final short **Domain.SCALE_MIN**

SHORT 变量

16 位整数（SQL 类型 SMALLINT）的域 ID 常量。

语法

final short **Domain.SHORT**

另请参见

- “Domain 接口”一节第 156 页

TIME 变量

时间（SQL 类型 TIME）的域 ID 常量。

语法

final short **Domain.TIME**

另请参见

- “Domain 接口”一节第 156 页

TIMESTAMP 变量

时间戳（SQL 类型 TIMESTAMP）的域 ID 常量。

语法

final short **Domain.TIMESTAMP**

另请参见

- “Domain 接口”一节第 156 页

TINY 变量

无符号 8 位整数（SQL 类型 TINYINT）的域 ID 常量。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

final short **Domain.TINY**

另请参见

- “Domain 接口”一节第 156 页

UINT16_MAX 变量

无符号 16 位整数的大小上限。

语法

final int **Domain.UINT16_MAX**

UNSIGNED_BIG 变量

无符号 64 位整数（SQL 类型 UNSIGNED BIGINT）的域 ID 常量。

语法

final short **Domain.UNSIGNED_BIG**

另请参见

- “Domain 接口”一节第 156 页

UNSIGNED_INTEGER 变量

无符号 32 位整数（SQL 类型 UNSIGNED INTEGER）的域 ID 常量。

语法

final short **Domain.UNSIGNED_INTEGER**

另请参见

- “Domain 接口”一节第 156 页

UNSIGNED_SHORT 变量

无符号 16 位整数（SQL 类型 UNSIGNED SMALLINT）的域 ID 常量。

语法

final short **Domain.UNSIGNED_SHORT**

另请参见

- “Domain 接口”一节第 156 页

UUID 变量

UniqueIdentifier（SQL 类型 UNIQUEIDENTIFIER）的域 ID 常量。

语法

final short **Domain.UUID**

另请参见

- “Domain 接口”一节第 156 页

VARCHAR 变量

最大字节数为 *size* 的可变长度字符串对象（SQL 类型 VARCHAR(*size*)）的域 ID 常量。

语法

final short **Domain.VARCHAR**

另请参见

- “Domain 接口”一节第 156 页

VARCHAR_DEFAULT 变量

字符类型的缺省大小。

语法

final short **Domain.VARCHAR_DEFAULT**

VARCHAR_MIN 变量

字符类型的大小下限。

语法

final short **Domain.VARCHAR_MIN**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

getName 方法

返回域的名称。

语法

String **Domain.getName()**

返回值

域名。

getPrecision 方法

返回域值的精度。

语法

int **Domain.getPrecision()**

返回值

值的精度。

getScale 方法

返回域值的小数位数。

语法

int **Domain.getScale()**

返回值

值的小数位数。

getSize 方法

返回域值的大小。

语法

short **Domain.getSize()**

返回值

值的大小。

getType 方法

返回域的类型。

语法

```
short Domain.getType()
```

返回值

以整数形式表示的域类型。

EncryptionControl 接口

为数据库提供加密控制。

语法

public **EncryptionControl**

注释

此接口用于实现自己的加密或模糊处理方法。要加密数据库，应先创建一个用于实现 EncryptionControl 的新类，并为类提供您自己的加密方法，然后再使用 ConfigPersistent 接口中的 setEncryption 方法启动 EncryptionControl 类的一个新实例。

另请参见

- "setEncryption 方法"一节第 114 页

成员

EncryptionControl 的所有成员，包括所有继承的成员。

- "decrypt 方法"一节第 169 页
- "encrypt 方法"一节第 170 页
- "initialize 方法"一节第 170 页

decrypt 方法

解密数据库中的字节数组。

语法

void **EncryptionControl.decrypt(**
  int *page_no*,
  byte[] *src*,
  byte[] *tgt*
) throws **ULjException**

参数

- **page_no**　数组数据的页编号。
- **src**　已加密的源页。
- **tgt**　使用此方法解密的结果页。

注释

为此方法提供加密的字节数组 src 和相关的页编号。您的方法必须解密 src 并将结果存储在 tgt 字节数组中。随后将在您的应用程序中使用 tgt 来进行数据操作。

encrypt 方法

加密数据库中的字节数组。

语法

void **EncryptionControl.encrypt(**
  int *page_no*,
  byte[] *src*,
  byte[] *tgt*
) throws **ULjException**

参数

- **page_no**　数组数据的页编号。
- **src**　已解密的源页。
- **tgt**　使用此方法加密的结果页。

注释

为此方法提供未加密的字节数组 src 和相关的页编号。您的方法必须加密或模糊处理 src 并将结果存储在 tgt 字节数组中。tgt 随后将被存储到数据库中。

initialize 方法

初始化带口令的加密控制。

语法

void **EncryptionControl.initialize(**
  String *password*
) throws **ULjException**

参数

- **password**　用于加密和解密的口令。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

ForeignKeySchema 接口

指定外键的模式。

语法

public **ForeignKeySchema**

注释

支持此接口的对象由 Connection.createForeignKey(String) 方法返回。

所有外键都必须至少具有一个列引用。所引用的列的集合必须是主表中的列，并且该集合必须受主表上主键或唯一键约束的限制。

以下示例演示了简单数据库模式的创建过程。Invoices 表具有到 Products 表的外键，这指定所有的发票都应引用有效的产品 ID。

```
TableSchema table_schema;
IndexSchema index_schema;
ForeignKeySchema fkey_schema;

table_schema = conn.createTable("Invoices");
table_schema.createColumn("inv_id", Domain.INTEGER);
table_schema.createColumn("quantity", Domain.INTEGER);
table_schema.createColumn("sold_prod_id", Domain.INTEGER);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("inv_id", IndexSchema.ASCENDING);

table_schema = conn.createTable("Products");
table_schema.createColumn("prod_id", Domain.INTEGER);
table_schema.createColumn("prod_name", Domain.VARCHAR, 40);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("prod_id", IndexSchema.ASCENDING);

fkey_schema = conn.createForeignKey(
    "Invoices", "Products", "InvoiceToProduct" );
fkey_schema.addColumnReference("sold_prod_is", "prod_id");

conn.schemaCreateComplete();
```

成员

ForeignKeySchema 的所有成员，包括所有继承的成员。

- “addColumnReference 方法”一节第 171 页

addColumnReference 方法

向外键中添加列引用。

语法

```
ForeignKeySchema ForeignKeySchema.addColumnReference(
  String foreign_column,
  String primary_column
) throws ULjException
```

参数

- **foreign_column**　包含外键的列的名称。此列中的值用于引用主表的 primary_column_name 列中的值。
- **primary_column**　被引用的表中列的名称。所有主列（作为一个集合）必须受主表的主键或唯一键约束的限制。

返回值

外键已指派到外列的 ForeignKeySchema。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

IndexSchema 接口

指定索引的模式，并提供用于查询系统表的常量。

语法

public **IndexSchema**

注释

支持此接口的对象由 TableSchema.createIndex(String)、TableSchema.createPrimaryIndex(String)、TableSchema.createUniqueIndex(String) 和 TableSchema.createUniqueKey(String) 方法返回。有关各种索引类型的说明，请参见“TableSchema 接口”一节第 240 页。

所有索引都必须至少具有一列。

索引先按照添加到索引的第一列排序，随后按第二列（如果指定）排序，依此类推。

索引不应含有 LONGBINARY 或 LONGVARCHAR 类型的列。

以下示例演示了一个两列索引的创建过程。

```
// Assumes a valid TableSchema object table_schema on
// a table with columns A and B.
IndexSchema index_schema;
index_schema = table_schema.createIndex("AthenBreversed");
index_schema.addColumn("A", IndexSchema.ASCENDING);
index_schema.addColumn("B", IndexSchema.DESCENDING);
```

成员

IndexSchema 的所有成员，包括所有继承的成员。

- “addColumn 方法”一节第 175 页
- “ASCENDING 变量”一节第 173 页
- “DESCENDING 变量”一节第 174 页
- “PERSISTENT 变量”一节第 174 页
- “PRIMARY_INDEX 变量”一节第 174 页
- “UNIQUE_INDEX 变量”一节第 174 页
- “UNIQUE_KEY 变量”一节第 174 页

ASCENDING 变量

对于某一列，索引按升序排序。

语法

final byte **IndexSchema.ASCENDING**

DESCENDING 变量

对于某一列，索引按降序排序。

语法

final byte **IndexSchema.DESCENDING**

PERSISTENT 变量

位标志，用于表明某一索引为持久性索引。

语法

final byte **IndexSchema.PERSISTENT**

注释

可以在逻辑上将该值与 SYS_INDEXES 表的 index_flags 列中的其它标志加以组合。

PRIMARY_INDEX 变量

位标志，用于表明某一索引为主键。

语法

final byte **IndexSchema.PRIMARY_INDEX**

注释

可以在逻辑上将该值与 SYS_INDEXES 表的 index_flags 列中的其它标志加以组合。

UNIQUE_INDEX 变量

位标志，用于表明某一索引是唯一索引。

语法

final byte **IndexSchema.UNIQUE_INDEX**

注释

可以在逻辑上将该值与 SYS_INDEXES 表的 index_flags 列中的其它标志加以组合。

UNIQUE_KEY 变量

位标志，用于表明某一索引是唯一键。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

final byte **IndexSchema.UNIQUE_KEY**

注释

可以在逻辑上将该值与 SYS_INDEXES 表的 index_flags 列中的其它标志加以组合。

addColumn 方法

将列添加到索引。

语法

IndexSchema **IndexSchema.addColumn(**
 String *column_name*,
 byte *sort_order*
) throws **ULjException**

参数

- **column_name**　要添加的列的名称。所指定的列必须是创建此索引的表中的列。
- **sort_order**　确定排序顺序的常量。必须是 IndexSchema.ASCENDING 或 IndexSchema.DESCENDING。

注释

列添加到索引的顺序决定了排序优先级。第一列的优先级最高。

返回值

添加了列的 IndexSchema。

PreparedStatement 接口

提供若干方法，用于执行 SQL 查询以生成 ResultSet，或用于对 UltraLite 数据库执行准备的 SQL 语句。

语法

public **PreparedStatement**

基类

- “CollectionOfValueWriters 接口”一节第 98 页

注释

以下示例演示了如何执行 PreparedStatement、检查执行过程是否创建了 ResultSet、将 ResultSet 保存到局部变量，以及如何关闭 PreparedStatement：

```
// Create a new PreparedStatement object from an existing connection.
String sql_string = "SELECT * FROM SampleTable";
PreparedStatement ps = conn.prepareStatement(sql_string);

// result returns true if the execute statement runs successfully.
boolean result = ps.execute();

// Check if the PreparedStatement contains a ResultSet.
if (ps.hasResultSet()) {
    // Store the ResultSet in the rs variable.
    ResultSet rs = ps.getResultSet();
}

// Close the PreparedStatement to release resources.
ps.close();
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

成员

PreparedStatement 的所有成员，包括所有继承的成员。

- “close 方法”一节第 177 页
- “execute 方法”一节第 177 页
- “executeQuery 方法”一节第 178 页
- “getBlobOutputStream 方法”一节第 99 页
- “getClobWriter 方法”一节第 99 页
- “getOrdinal 方法”一节第 99 页
- “getPlan 方法”一节第 178 页
- “getResultSet 方法”一节第 178 页
- “getUpdateCount 方法”一节第 179 页
- “hasResultSet 方法”一节第 179 页
- “set 方法”一节第 100 页
- “set 方法”一节第 100 页
- “set 方法”一节第 100 页
- “set 方法”一节第 101 页
- “set 方法”一节第 101 页
- “set 方法”一节第 101 页
- “set 方法”一节第 102 页
- “set 方法”一节第 102 页
- “set 方法”一节第 102 页
- “set 方法”一节第 103 页
- “setNull 方法”一节第 103 页

close 方法

关闭 PreparedStatement，以释放与其相关的内存资源。

语法

void **PreparedStatement.close()** throws **ULjException**

注释

无法对此对象使用其它方法。如果 PreparedStatement 包含 ResultSet，则 ResultSet 也随之关闭。

execute 方法

执行准备的 SQL 语句。

语法

boolean **PreparedStatement.execute()** throws **ULjException**

另请参见

- “ResultSet 接口”一节第 180 页

返回值

如果执行的语句成功运行，则为 true；否则为 false。

executeQuery 方法

执行准备的 SQL SELECT 语句并返回 ResultSet。

语法

ResultSet **PreparedStatement.executeQuery()** throws **ULjException**

另请参见

- “ResultSet 接口”一节第 180 页

返回值

包含准备的 SQL SELECT 语句的查询结果的 ResultSet。

getPlan 方法

返回对 SQL 查询执行计划的基于文本的说明。

语法

String **PreparedStatement.getPlan()** throws **ULjException**

注释

如果没有任何计划，则返回空字符串。

返回值

计划的字符串表示。

getResultSet 方法

返回准备的 SQL 语句的 ResultSet。

语法

ResultSet **PreparedStatement.getResultSet()** throws **ULjException**

另请参见

- “ResultSet 接口”一节第 180 页

返回值

包含准备的 SQL 语句的查询结果的 ResultSet。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

getUpdateCount 方法

返回自上一提交语句之后插入、更新或删除的行数。

语法

int **PreparedStatement.getUpdateCount()** throws **ULjException**

返回值

如果未进行更改，则为 -1；否则返回更新的行数。

hasResultSet 方法

确定 PreparedStatement 是否包含 ResultSet。

语法

boolean **PreparedStatement.hasResultSet()** throws **ULjException**

另请参见

- "ResultSet 接口"一节第 180 页

返回值

如果找到 ResultSet，则为 true；否则为 false。

ResultSet 接口

提供多种方法，用于按行遍历表并访问列数据。

语法

public **ResultSet**

基类

- “CollectionOfValueReaders 接口”一节第 91 页

注释

可以通过使用 execute 或 executeQuery 方法执行带有 SQL SELECT 语句的 PreparedStatement 生成 ResultSet。

以下示例演示了如何执行新 PreparedStatement、通过 ResultSet 读取行，以及如何访问指定列中的数据。

```
// Define a new SQL SELECT statement.
String sql_string = "SELECT column1, column2 FROM SampleTable";

// Create a new PreparedStatement from an existing connection.
PreparedStatement ps = conn.prepareStatement(sql_string);

// Create a new ResultSet to contain the query results of the SQL statement.
ResultSet rs = ps.executeQuery();

// Check if the PreparedStatement contains a ResultSet.
if (ps.hasResultSet()) {
    // Retrieve the column values from the first row using getString.
    while (rs.next()) {
        c1 = rs.getString(1);
        c2 = rs.getString(2);
        ...
    }
}
```


Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

成员

ResultSet 的所有成员，包括所有继承的成员。

- “close 方法”一节第 181 页
- “getBlobInputStream 方法”一节第 92 页
- “getBoolean 方法”一节第 92 页
- “getBytes 方法”一节第 93 页
- “getClobReader 方法”一节第 93 页
- “getDate 方法”一节第 93 页
- “getDecimalNumber 方法”一节第 94 页
- “getDouble 方法”一节第 94 页
- “getFloat 方法”一节第 94 页
- “getInt 方法”一节第 95 页
- “getLong 方法”一节第 95 页
- “getOrdinal 方法”一节第 96 页
- “getResultSetMetadata 方法”一节第 181 页
- “getString 方法”一节第 96 页
- “getValue 方法”一节第 96 页
- “isNull 方法”一节第 97 页
- “next 方法”一节第 182 页
- “previous 方法”一节第 182 页

close 方法

关闭 ResultSet，以释放与其相关的内存资源。

语法

void **ResultSet.close()** throws **ULjException**

注释

如果进行后续尝试以从该 ResultSet 中读取行，则会抛出错误。

getResultSetMetadata 方法

返回包含 ResultSet 的元数据的 ResultSetMetadata。

语法

ResultSetMetadata **ResultSet.getResultSetMetadata()** throws **ULjException**

返回值

ResultSetMetadata 对象。

next 方法

读取 ResultSet 中的下一行数据。

语法

boolean **ResultSet.next()** throws **ULjException**

另请参见

- “ResultSetMetadata 接口”一节第 183 页

返回值

成功读取下一行时，返回 true；否则，返回 false。

previous 方法

读取 ResultSet 中的前一行数据。

语法

boolean **ResultSet.previous()** throws **ULjException**

返回值

成功读取前一行时，返回 true；否则，返回 false。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

ResultSetMetadata 接口

与 ResultSet 对象相关联，包含用于提供列信息的方法。

语法

public **ResultSetMetadata**

注释

可使用 ResultSet 对象的 getResultSetMetadata 方法调用此接口。

成员

ResultSetMetadata 的所有成员，包括所有继承的成员。

- “getColumnCount 方法”一节第 183 页

getColumnCount 方法

返回 ResultSet 中的总列数。

语法

int **ResultSetMetadata.getColumnCount()** throws **ULjException**

返回值

列数。

SISListener 接口（仅限 J2ME BlackBerry）

监听服务器启动的同步的消息。

语法

public **SISListener**

注释

应用程序使用 DatabaseManager 接口中的适当 createSISHTTPListener 方法来创建 SISListener 的实例。

成员

SISListener 的所有成员，包括所有继承的成员。

- “startListening 方法”一节第 184 页
- “stopListening 方法”一节第 184 页
- “createSISHTTPListener 函数（仅限 J2ME BlackBerry）”一节第 150 页

startListening 方法

创建并启动监听线程。

语法

void **SISListener.startListening()**

stopListening 方法

停止监听线程。

语法

void **SISListener.stopListening()**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SISRequestHandler 接口（仅限 J2ME BlackBerry）

处理服务器启动的同步请求。

语法

public **SISRequestHandler**

成员

SISRequestHandler 的所有成员，包括所有继承的成员。

- "onError 方法"一节第 185 页
- "onRequest 方法"一节第 185 页

onError 方法

处理 SIS 监听过程中发生的 SIS 相关错误。

语法

```
void SISListener.onError(
  String text
)
```

参数

- **text**　异常的字符串表示。

onRequest 方法

处理工作线程上的 SIS 请求。

语法

```
void SISListener.onRequest(
  String text
)
```

参数

- **text**　请求所发送的字符串。

注释

要停止监听，请显式调用 SISListener 接口中的 stopListening 方法。

SQLCode 接口

枚举 UltraLiteJ 可能会报告的 SQL 代码。

语法

public **SQLCode**

派生类

- “ULjException 类”一节第 248 页

注释

有关各种错误的详细说明，请参见 SQL Anywhere 文档集中的 "按 SQLCODE 分类的 SQL Anywhere 错误消息" 主题。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

成员

SQLCode 的所有成员，包括所有继承的成员。

- “SQLE_AGGREGATES_NOT_ALLOWED 变量”一节第 189 页
- “SQLE_ALIAS_NOT_UNIQUE 变量”一节第 189 页
- “SQLE_ALIAS_NOT_YET_DEFINED 变量”一节第 189 页
- “SQLE_AUTHENTICATION_FAILED 变量”一节第 189 页
- “SQLE_CANNOT_EXECUTE_STMT 变量”一节第 189 页
- “SQLE_CLIENT_OUT_OF_MEMORY 变量”一节第 189 页
- “SQLE_COLUMN_AMBIGUOUS 变量”一节第 190 页
- “SQLE_COLUMN_CANNOT_BE_NULL 变量”一节第 190 页
- “SQLE_COLUMN_NOT_FOUND 变量”一节第 190 页
- “SQLE_COLUMN_NOT_STREAMABLE 变量”一节第 190 页
- “SQLE_COMMUNICATIONS_ERROR 变量”一节第 190 页
- “SQLE_CONFIG_IN_USE 变量”一节第 191 页
- “SQLE_CONVERSION_ERROR 变量”一节第 191 页
- “SQLE_CURSOR_ALREADY_OPEN 变量”一节第 191 页
- “SQLE_DATABASE_ACTIVE 变量”一节第 191 页
- “SQLE_DEVICE_IO_FAILED 变量”一节第 191 页
- “SQLE_DIV_ZERO_ERROR 变量”一节第 191 页
- “SQLE_DOWNLOAD_CONFLICT 变量”一节第 192 页
- “SQLE_ERROR 变量”一节第 192 页
- “SQLE_EXISTING_PRIMARY_KEY 变量”一节第 192 页
- “SQLE_EXPRESSION_ERROR 变量”一节第 192 页
- “SQLE_FILE_BAD_DB 变量”一节第 192 页
- “SQLE_FILE_WRONG_VERSION 变量”一节第 193 页
- “SQLE_FOREIGN_KEY_NAME_NOT_FOUND 变量”一节第 193 页
- “SQLE_IDENTIFIER_TOO_LONG 变量”一节第 193 页
- “SQLE_INCOMPLETE_SYNCHRONIZATION 变量”一节第 193 页
- “SQLE_INDEX_HAS_NO_COLUMNS 变量”一节第 193 页
- “SQLE_INDEX_NOT_FOUND 变量”一节第 193 页
- “SQLE_INDEX_NOT_UNIQUE 变量”一节第 194 页
- “SQLE_INTERRUPTED 变量”一节第 194 页
- “SQLE_INVALID_COMPARISON 变量”一节第 194 页
- “SQLE_INVALID_DISTINCT_AGGREGATE 变量”一节第 194 页
- “SQLE_INVALID_DOMAIN 变量”一节第 194 页
- “SQLE_INVALID_FOREIGN_KEY_DEF 变量”一节第 195 页
- “SQLE_INVALID_GROUP_SELECT 变量”一节第 195 页
- “SQLE_INVALID_INDEX_TYPE 变量”一节第 195 页
- “SQLE_INVALID_LOGON 变量”一节第 195 页
- “SQLE_INVALID_OPTION 变量”一节第 195 页
- “SQLE_INVALID_OPTION_SETTING 变量”一节第 195 页
- “SQLE_INVALID_ORDER 变量”一节第 196 页
- “SQLE_INVALID_PARAMETER 变量”一节第 196 页
- “SQLE_INVALID_UNION 变量”一节第 196 页
- “SQLE_LOCKED 变量”一节第 196 页
- “SQLE_MAX_ROW_SIZE_EXCEEDED 变量”一节第 196 页

- “SQLE_MUST_BE_ONLY_CONNECTION 变量”一节第 197 页
- “SQLE_NAME_NOT_UNIQUE 变量”一节第 197 页
- “SQLE_NO_COLUMN_NAME 变量”一节第 197 页
- “SQLE_NO_CURRENT_ROW 变量”一节第 197 页
- “SQLE_NO_MATCHING_SELECT_ITEM 变量”一节第 198 页
- “SQLE_NO_PRIMARY_KEY 变量”一节第 198 页
- “SQLE_NOERROR 变量”一节第 197 页
- “SQLE_NOT_IMPLEMENTED 变量”一节第 197 页
- “SQLE_OVERFLOW_ERROR 变量”一节第 198 页
- “SQLE_PAGE_SIZE_TOO_BIG 变量”一节第 198 页
- “SQLE_PAGE_SIZE_TOO_SMALL 变量”一节第 198 页
- “SQLE_PARAMETER_CANNOT_BE_NULL 变量”一节第 199 页
- “SQLE_PERMISSION_DENIED 变量”一节第 199 页
- “SQLE_PRIMARY_KEY_NOT_UNIQUE 变量”一节第 199 页
- “SQLE_PUBLICATION_NOT_FOUND 变量”一节第 199 页
- “SQLE_RESOURCE_GOVERNOR_EXCEEDED 变量”一节第 199 页
- “SQLE_ROW_LOCKED 变量”一节第 199 页
- “SQLE_ROW_UPDATED_SINCE_READ 变量”一节第 200 页
- “SQLE_SCHEMA_UPGRADE_NOT_ALLOWED 变量”一节第 200 页
- “SQLE_SERVER_SYNCHRONIZATION_ERROR 变量”一节第 200 页
- “SQLE_SUBQUERY_RESULT_NOT_UNIQUE 变量”一节第 200 页
- “SQLE_SUBQUERY_SELECT_LIST 变量”一节第 200 页
- “SQLE_SYNC_INFO_INVALID 变量”一节第 201 页
- “SQLE_SYNCHRONIZATION_IN_PROGRESS 变量”一节第 201 页
- “SQLE_SYNTAX_ERROR 变量”一节第 201 页
- “SQLE_TABLE_HAS_NO_COLUMNS 变量”一节第 201 页
- “SQLE_TABLE_IN_USE 变量”一节第 201 页
- “SQLE_TABLE_NOT_FOUND 变量”一节第 201 页
- “SQLE_TOO_MANY_PUBLICATIONS 变量”一节第 202 页
- “SQLE_ULTRALITE_DATABASE_NOT_FOUND 变量”一节第 202 页
- “SQLE_ULTRALITE_OBJ_CLOSED 变量”一节第 202 页
- “SQLE_ULTRALITEJ_OPERATION_FAILED 变量”一节第 202 页
- “SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED 变量”一节第 202 页
- “SQLE_UNABLE_TO_CONNECT 变量”一节第 203 页
- “SQLE_UNCOMMITTED_TRANSACTIONS 变量”一节第 203 页
- “SQLE_UNDERFLOW 变量”一节第 203 页
- “SQLE_UNKNOWN_FUNC 变量”一节第 203 页
- “SQLE_UPLOAD_FAILED_AT_SERVER 变量”一节第 203 页
- “SQLE_VALUE_IS_NULL 变量”一节第 203 页
- “SQLE_VARIABLE_INVALID 变量”一节第 204 页
- “SQLE_WRONG_NUM_OF_INSERT_COLS 变量”一节第 204 页
- “SQLE_WRONG_PARAMETER_COUNT 变量”一节第 204 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_AGGREGATES_NOT_ALLOWED 变量

SQLE_AGGREGATES_NOT_ALLOWED(-150)。

语法

final int **SQLCode.SQLE_AGGREGATES_NOT_ALLOWED**

SQLE_ALIAS_NOT_UNIQUE 变量

SQLE_ALIAS_NOT_UNIQUE(-830)。

语法

final int **SQLCode.SQLE_ALIAS_NOT_UNIQUE**

SQLE_ALIAS_NOT_YET_DEFINED 变量

SQLE_ALIAS_NOT_YET_DEFINED(-831)。

语法

final int **SQLCode.SQLE_ALIAS_NOT_YET_DEFINED**

SQLE_AUTHENTICATION_FAILED 变量

SQLE_AUTHENTICATION_FAILED(-218)。

语法

final int **SQLCode.SQLE_AUTHENTICATION_FAILED**

SQLE_CANNOT_EXECUTE_STMT 变量

SQLE_CANNOT_EXECUTE_STMT(111)。

语法

final int **SQLCode.SQLE_CANNOT_EXECUTE_STMT**

SQLE_CLIENT_OUT_OF_MEMORY 变量

SQLE_CLIENT_OUT_OF_MEMORY(-876)。

语法

final int **SQLCode.SQLE_CLIENT_OUT_OF_MEMORY**

SQLE_COLUMN_AMBIGUOUS 变量

SQLE_COLUMN_AMBIGUOUS(-144)。

语法

final int **SQLCode.SQLE_COLUMN_AMBIGUOUS**

SQLE_COLUMN_CANNOT_BE_NULL 变量

SQLE_COLUMN_CANNOT_BE_NULL(-195)。

语法

final int **SQLCode.SQLE_COLUMN_CANNOT_BE_NULL**

SQLE_COLUMN_NOT_FOUND 变量

SQLE_COLUMN_NOT_FOUND(-143)。

语法

final int **SQLCode.SQLE_COLUMN_NOT_FOUND**

SQLE_COLUMN_NOT_STREAMABLE 变量

SQLE_COLUMN_NOT_STREAMABLE(-1100)。

语法

final int **SQLCode.SQLE_COLUMN_NOT_STREAMABLE**

SQLE_COMMUNICATIONS_ERROR 变量

SQLE_COMMUNICATIONS_ERROR(-85)。

语法

final int **SQLCode.SQLE_COMMUNICATIONS_ERROR**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_CONFIG_IN_USE 变量

SQLE_CONFIG_IN_USE(-1276)。

语法

final int **SQLCode.SQLE_CONFIG_IN_USE**

SQLE_CONVERSION_ERROR 变量

SQLE_CONVERSION_ERROR(-157)。

语法

final int **SQLCode.SQLE_CONVERSION_ERROR**

SQLE_CURSOR_ALREADY_OPEN 变量

SQLE_CURSOR_ALREADY_OPEN(-172)。

语法

final int **SQLCode.SQLE_CURSOR_ALREADY_OPEN**

SQLE_DATABASE_ACTIVE 变量

SQLE_DATABASE_ACTIVE(-664)。

语法

final int **SQLCode.SQLE_DATABASE_ACTIVE**

SQLE_DEVICE_IO_FAILED 变量

SQLE_DEVICE_IO_FAILED(-974)。

语法

final int **SQLCode.SQLE_DEVICE_IO_FAILED**

SQLE_DIV_ZERO_ERROR 变量

SQLE_DIV_ZERO_ERROR(-628)。

语法

final int **SQLCode.SQLE_DIV_ZERO_ERROR**

SQLE_DOWNLOAD_CONFLICT 变量

SQLE_DOWNLOAD_CONFLICT(-839)。

语法

final int **SQLCode.SQLE_DOWNLOAD_CONFLICT**

SQLE_ERROR 变量

SQLE_ERROR(-300)。

语法

final int **SQLCode.SQLE_ERROR**

SQLE_EXISTING_PRIMARY_KEY 变量

SQLE_EXISTING_PRIMARY_KEY(-112)。

语法

final int **SQLCode.SQLE_EXISTING_PRIMARY_KEY**

SQLE_EXPRESSION_ERROR 变量

SQLE_EXPRESSION_ERROR(-156)。

语法

final int **SQLCode.SQLE_EXPRESSION_ERROR**

SQLE_FILE_BAD_DB 变量

SQLE_FILE_BAD_DB(-1006)。

语法

final int **SQLCode.SQLE_FILE_BAD_DB**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_FILE_WRONG_VERSION 变量

SQLE_FILE_WRONG_VERSION(-1005)。

语法

final int **SQLCode.SQLE_FILE_WRONG_VERSION**

SQLE_FOREIGN_KEY_NAME_NOT_FOUND 变量

SQLE_FOREIGN_KEY_NAME_NOT_FOUND(-145)。

语法

final int **SQLCode.SQLE_FOREIGN_KEY_NAME_NOT_FOUND**

SQLE_IDENTIFIER_TOO_LONG 变量

SQLE_IDENTIFIER_TOO_LONG(-250)。

语法

final int **SQLCode.SQLE_IDENTIFIER_TOO_LONG**

SQLE_INCOMPLETE_SYNCHRONIZATION 变量

SQLE_INCOMPLETE_SYNCHRONIZATION(-1271)。

语法

final int **SQLCode.SQLE_INCOMPLETE_SYNCHRONIZATION**

SQLE_INDEX_HAS_NO_COLUMNS 变量

SQLE_INDEX_HAS_NO_COLUMNS(-1274)。

语法

final int **SQLCode.SQLE_INDEX_HAS_NO_COLUMNS**

SQLE_INDEX_NOT_FOUND 变量

SQLE_INDEX_NOT_FOUND(-183)。

语法

final int **SQLCode.SQLE_INDEX_NOT_FOUND**

SQLE_INDEX_NOT_UNIQUE 变量

SQLE_INDEX_NOT_UNIQUE(-196)。

语法

final int **SQLCode.SQLE_INDEX_NOT_UNIQUE**

SQLE_INTERRUPTED 变量

SQLE_INTERRUPTED(-299)。

语法

final int **SQLCode.SQLE_INTERRUPTED**

SQLE_INVALID_COMPARISON 变量

SQLE_INVALID_COMPARISON(-710)。

语法

final int **SQLCode.SQLE_INVALID_COMPARISON**

SQLE_INVALID_DISTINCT_AGGREGATE 变量

SQLE_INVALID_DISTINCT_AGGREGATE(-863)。

语法

final int **SQLCode.SQLE_INVALID_DISTINCT_AGGREGATE**

SQLE_INVALID_DOMAIN 变量

SQLE_INVALID_DOMAIN(-1275)。

语法

final int **SQLCode.SQLE_INVALID_DOMAIN**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_INVALID_FOREIGN_KEY_DEF 变量

SQLE_INVALID_FOREIGN_KEY_DEF(-113)。

语法

final int **SQLCode.SQLE_INVALID_FOREIGN_KEY_DEF**

SQLE_INVALID_GROUP_SELECT 变量

SQLE_INVALID_GROUP_SELECT(-149)。

语法

final int **SQLCode.SQLE_INVALID_GROUP_SELECT**

SQLE_INVALID_INDEX_TYPE 变量

SQLE_INVALID_INDEX_TYPE(-650)。

语法

final int **SQLCode.SQLE_INVALID_INDEX_TYPE**

SQLE_INVALID_LOGON 变量

SQLE_INVALID_LOGON(-103)。

语法

final int **SQLCode.SQLE_INVALID_LOGON**

SQLE_INVALID_OPTION 变量

SQLE_INVALID_OPTION(-200)。

语法

final int **SQLCode.SQLE_INVALID_OPTION**

SQLE_INVALID_OPTION_SETTING 变量

SQLE_INVALID_OPTION_SETTING(-201)。

语法

final int **SQLCode.SQLE_INVALID_OPTION_SETTING**

SQLE_INVALID_ORDER 变量

SQLE_INVALID_ORDER(-152)。

语法

final int **SQLCode.SQLE_INVALID_ORDER**

SQLE_INVALID_PARAMETER 变量

SQLE_INVALID_PARAMETER(-735)。

语法

final int **SQLCode.SQLE_INVALID_PARAMETER**

SQLE_INVALID_UNION 变量

SQLE_INVALID_UNION(-153)。

语法

final int **SQLCode.SQLE_INVALID_UNION**

SQLE_LOCKED 变量

SQLE_LOCKED(-210)。

语法

final int **SQLCode.SQLE_LOCKED**

SQLE_MAX_ROW_SIZE_EXCEEDED 变量

SQLE_MAX_ROW_SIZE_EXCEEDED(-1132)。

语法

final int **SQLCode.SQLE_MAX_ROW_SIZE_EXCEEDED**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_MUST_BE_ONLY_CONNECTION 变量

SQLE_MUST_BE_ONLY_CONNECTION(-211)。

语法

final int **SQLCode.SQLE_MUST_BE_ONLY_CONNECTION**

SQLE_NAME_NOT_UNIQUE 变量

SQLE_NAME_NOT_UNIQUE(-110)。

语法

final int **SQLCode.SQLE_NAME_NOT_UNIQUE**

SQLE_NOERROR 变量

SQLE_NOERROR(0)。

语法

final int **SQLCode.SQLE_NOERROR**

SQLE_NOT_IMPLEMENTED 变量

SQLE_NOT_IMPLEMENTED(-134)。

语法

final int **SQLCode.SQLE_NOT_IMPLEMENTED**

SQLE_NO_COLUMN_NAME 变量

SQLE_NO_COLUMN_NAME(-163)。

语法

final int **SQLCode.SQLE_NO_COLUMN_NAME**

SQLE_NO_CURRENT_ROW 变量

SQLE_NO_CURRENT_ROW(-197)。

语法

final int **SQLCode.SQLE_NO_CURRENT_ROW**

SQLE_NO_MATCHING_SELECT_ITEM 变量

SQLE_NO_MATCHING_SELECT_ITEM(-812)。

语法

final int **SQLCode.SQLE_NO_MATCHING_SELECT_ITEM**

SQLE_NO_PRIMARY_KEY 变量

SQLE_NO_PRIMARY_KEY(-118)。

语法

final int **SQLCode.SQLE_NO_PRIMARY_KEY**

SQLE_OVERFLOW_ERROR 变量

SQLE_OVERFLOW_ERROR(-158)。

语法

final int **SQLCode.SQLE_OVERFLOW_ERROR**

SQLE_PAGE_SIZE_TOO_BIG 变量

SQLE_PAGE_SIZE_TOO_BIG(-97)。

语法

final int **SQLCode.SQLE_PAGE_SIZE_TOO_BIG**

SQLE_PAGE_SIZE_TOO_SMALL 变量

SQLE_PAGE_SIZE_TOO_SMALL(-972)。

语法

final int **SQLCode.SQLE_PAGE_SIZE_TOO_SMALL**

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_PARAMETER_CANNOT_BE_NULL 变量

SQLE_PARAMETER_CANNOT_BE_NULL(-1277)。

语法

final int **SQLCode.SQLE_PARAMETER_CANNOT_BE_NULL**

SQLE_PERMISSION_DENIED 变量

SQLE_PERMISSION_DENIED(-121)。

语法

final int **SQLCode.SQLE_PERMISSION_DENIED**

SQLE_PRIMARY_KEY_NOT_UNIQUE 变量

SQLE_PRIMARY_KEY_NOT_UNIQUE(-193)。

语法

final int **SQLCode.SQLE_PRIMARY_KEY_NOT_UNIQUE**

SQLE_PUBLICATION_NOT_FOUND 变量

SQLE_PUBLICATION_NOT_FOUND(-280)。

语法

final int **SQLCode.SQLE_PUBLICATION_NOT_FOUND**

SQLE_RESOURCE_GOVERNOR_EXCEEDED 变量

SQLE_RESOURCE_GOVERNOR_EXCEEDED(-685)。

语法

final int **SQLCode.SQLE_RESOURCE_GOVERNOR_EXCEEDED**

SQLE_ROW_LOCKED 变量

SQLE_ROW_LOCKED(-1281)。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

final int **SQLCode.SQLE_ROW_LOCKED**

SQLE_ROW_UPDATED_SINCE_READ 变量

SQLE_ROW_UPDATED_SINCE_READ(-208)。

语法

final int **SQLCode.SQLE_ROW_UPDATED_SINCE_READ**

SQLE_SCHEMA_UPGRADE_NOT_ALLOWED 变量

SQLE_SCHEMA_UPGRADE_NOT_ALLOWED(-953)。

语法

final int **SQLCode.SQLE_SCHEMA_UPGRADE_NOT_ALLOWED**

SQLE_SERVER_SYNCHRONIZATION_ERROR 变量

SQLE_SERVER_SYNCHRONIZATION_ERROR(-857)。

语法

final int **SQLCode.SQLE_SERVER_SYNCHRONIZATION_ERROR**

SQLE_SUBQUERY_RESULT_NOT_UNIQUE 变量

SQLE_SUBQUERY_RESULT_NOT_UNIQUE(-186)。

语法

final int **SQLCode.SQLE_SUBQUERY_RESULT_NOT_UNIQUE**

SQLE_SUBQUERY_SELECT_LIST 变量

SQLE_SUBQUERY_SELECT_LIST(-151)。

语法

final int **SQLCode.SQLE_SUBQUERY_SELECT_LIST**

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_SYNCHRONIZATION_IN_PROGRESS 变量

SQLE_SYNCHRONIZATION_IN_PROGRESS(-1272)。

语法

final int **SQLCode.SQLE_SYNCHRONIZATION_IN_PROGRESS**

SQLE_SYNC_INFO_INVALID 变量

SQLE_SYNC_INFO_INVALID(-956)。

语法

final int **SQLCode.SQLE_SYNC_INFO_INVALID**

SQLE_SYNTAX_ERROR 变量

SQLE_SYNTAX_ERROR(-131)。

语法

final int **SQLCode.SQLE_SYNTAX_ERROR**

SQLE_TABLE_HAS_NO_COLUMNS 变量

SQLE_TABLE_HAS_NO_COLUMNS(-1273)。

语法

final int **SQLCode.SQLE_TABLE_HAS_NO_COLUMNS**

SQLE_TABLE_IN_USE 变量

SQLE_TABLE_IN_USE(-214)。

语法

final int **SQLCode.SQLE_TABLE_IN_USE**

SQLE_TABLE_NOT_FOUND 变量

SQLE_TABLE_NOT_FOUND(-141)。

语法

final int **SQLCode.SQLE_TABLE_NOT_FOUND**

SQLE_TOO_MANY_PUBLICATIONS 变量

SQLE_TOO_MANY_PUBLICATIONS(-1106)。

语法

final int **SQLCode.SQLE_TOO_MANY_PUBLICATIONS**

SQLE_ULTRALITEJ_OPERATION_FAILED 变量

SQLE_ULTRALITEJ_OPERATION_FAILED(-1279)。

语法

final int **SQLCode.SQLE_ULTRALITEJ_OPERATION_FAILED**

SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED 变量

SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED(-1278)。

语法

final int **SQLCode.SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED**

SQLE_ULTRALITE_DATABASE_NOT_FOUND 变量

SQLE_ULTRALITE_DATABASE_NOT_FOUND(-954)。

语法

final int **SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND**

SQLE_ULTRALITE_OBJ_CLOSED 变量

SQLE_ULTRALITE_OBJ_CLOSED(-908)。

语法

final int **SQLCode.SQLE_ULTRALITE_OBJ_CLOSED**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_UNABLE_TO_CONNECT 变量

SQLE_UNABLE_TO_CONNECT(-105)。

语法

final int **SQLCode.SQLE_UNABLE_TO_CONNECT**

SQLE_UNCOMMITTED_TRANSACTIONS 变量

SQLE_UNCOMMITTED_TRANSACTIONS(-755)。

语法

final int **SQLCode.SQLE_UNCOMMITTED_TRANSACTIONS**

SQLE_UNDERFLOW 变量

SQLE_UNDERFLOW(-1280)。

语法

final int **SQLCode.SQLE_UNDERFLOW**

SQLE_UNKNOWN_FUNC 变量

SQLE_UNKNOWN_FUNC(-148)。

语法

final int **SQLCode.SQLE_UNKNOWN_FUNC**

SQLE_UPLOAD_FAILED_AT_SERVER 变量

SQLE_UPLOAD_FAILED_AT_SERVER(-794)。

语法

final int **SQLCode.SQLE_UPLOAD_FAILED_AT_SERVER**

SQLE_VALUE_IS_NULL 变量

SQLE_VALUE_IS_NULL(-1050)。

语法

final int **SQLCode.SQLE_VALUE_IS_NULL**

SQLE_VARIABLE_INVALID 变量

SQLE_VARIABLE_INVALID(-155)。

语法

final int **SQLCode.SQLE_VARIABLE_INVALID**

SQLE_WRONG_NUM_OF_INSERT_COLS 变量

SQLE_WRONG_NUM_OF_INSERT_COLS(-207)。

语法

final int **SQLCode.SQLE_WRONG_NUM_OF_INSERT_COLS**

SQLE_WRONG_PARAMETER_COUNT 变量

SQLE_WRONG_PARAMETER_COUNT(-154)。

语法

final int **SQLCode.SQLE_WRONG_PARAMETER_COUNT**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

StreamHTTPParms 接口

表示用于定义如何使用 HTTP 与 MobiLink 服务器通信的 HTTP 流参数。

语法

public **StreamHTTPParms**

派生类

- “StreamHTTPSParms 接口”一节第 209 页

注释

下面的示例将流参数设置为与主机名为 "MyMLHost" 的 MobiLink 11 服务器通信。使用以下参数启动该服务器："-xo http(port=1234)":

```
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTP_STREAM,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
StreamHTTPParms httpParms = syncParms.getStreamParms();
httpParms.setHost("MyMLHost");
httpParms.setPort(1234);
```

实现此接口的实例由 getStreamParms 函数方法返回。

成员

StreamHTTPParms 的所有成员，包括所有继承的成员。

- “getHost 方法”一节第 205 页
- “getOutputBufferSize 方法”一节第 206 页
- “getPort 方法”一节第 206 页
- “getURLSuffix 方法”一节第 206 页
- “setHost 方法”一节第 207 页
- “setOutputBufferSize 方法”一节第 207 页
- “setPort 方法”一节第 208 页
- “setURLSuffix 方法”一节第 208 页

getHost 方法

返回 MobiLink 服务器的主机名。

语法

String **StreamHTTPParms.getHost()**

另请参见

- “setHost 方法”一节第 207 页
- “getPort 方法”一节第 206 页
- “setPort 方法”一节第 208 页

返回值

主机名。

getOutputBufferSize 方法

返回在将数据发送到 MobiLink 服务器之前用于存储数据的输出缓冲区的大小（以字节为单位）。

语法

int **StreamHTTPParms.getOutputBufferSize()**

注释

增加此值可减少发送大量上载数据所需的网络刷新次数，但需要增加内存占用量。在 HTTP 中，每次刷新都会发送一个大的（大约 250 个字节）HTTP 标头；减少刷新次数可减少带宽占用。

另请参见

- “setOutputBufferSize 方法”一节第 207 页

返回值

包含缓冲区大小的整数。

getPort 方法

返回用于连接 MobiLink 服务器的端口号。

语法

int **StreamHTTPParms.getPort()**

另请参见

- “setPort 方法”一节第 208 页

返回值

MobiLink 服务器的端口号。

getURLSuffix 方法

返回包含 URL 后缀的字符串。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

String **StreamHTTPParms.getURLSuffix()**

另请参见

- "setURLSuffix 方法"一节第 208 页

返回值

包含 URL 后缀的字符串。

setHost 方法

设置 MobiLink 服务器的主机名。

语法

```
void StreamHTTPParms.setHost(
  String v
)
```

参数

- **v** 主机名。

注释

缺省设值为空，表示本地主机。

另请参见

- "getHost 方法"一节第 205 页
- "getPort 方法"一节第 206 页
- "setPort 方法"一节第 208 页

setOutputBufferSize 方法

设置在将数据发送到 MobiLink 服务器之前用于存储数据的输出缓冲区的大小（以字节为单位）。

语法

```
void StreamHTTPParms.setOutputBufferSize(
  int size
)
```

参数

- **size** 新缓冲区大小。

注释

非 Blackberry J2ME 的缺省值为 512；否则为 4096。有效值范围为 512 到 32768。调高该值可能会导致 Java 运行时发送 MobiLink 服务器无法处理的分块 HTTP。如果 MobiLink 服务器输出一个 "未知的传输编码" 错误，则请尝试调低该值。

另请参见

- “getOutputBufferSize 方法”一节第 206 页

setPort 方法

设置用于连接 MobiLink 服务器的端口号。

语法

```
void StreamHTTPParms.setPort(
  int v
)
```

参数

- **v** 范围在 1 到 65535 之间的端口号。若超出此范围该值将还原为缺省值。

注释

对于 HTTP 同步，缺省端口为 80；对于 HTTPS 同步，缺省端口为 443。

另请参见

- “getPort 方法”一节第 206 页

setURLSuffix 方法

设置 MobiLink 服务器的 URL 后缀。

语法

```
void StreamHTTPParms.setURLSuffix(
  String v
)
```

参数

- **v** URL 后缀字符串。

注释

缺省值为空，表示 "Mobilink/"。

另请参见

- “getURLSuffix 方法”一节第 206 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

StreamHTTPSParms 接口

表示用于定义如何使用安全 HTTPS 与 MobiLink 服务器通信的 HTTPS 流参数。

语法

public **StreamHTTPSParms**

基类

- "StreamHTTPParms 接口"一节第 205 页

注释

下面的示例将流参数设置为与主机名为 "MyMLHost" 的 MobiLink 11 服务器通信。使用以下参数启动该服务器："-xo https(port=1234;certificate=RSAServer.crt;certificate_password=x)"

```
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTPS,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
StreamHTTPSParms httpsParms =
    (StreamHTTPSParms) syncParms.getStreamParms();
httpsParms.setHost("MyMLHost");
httpsParms.setPort(1234);
```

上面的示例假设 RSAServer.crt 中的证书链接到已安装在客户端主机或设备上的受信任根证书。

对于 J2SE，可使用以下任一方法部署必需的受信任根证书：

1. 将受信任的根证书安装在 JRE 的 lib/security/cacerts 密钥存储区中。

2. 使用 Java keytool 实用程序构建自己的密钥存储区，并将 Java 系统属性 javax.net.ssl.trustStore 设置为该位置（将 javax.net.ssl.trustStorePassword 设置为适当值）。

3. 使用 setTrustedCertificates 函数参数指向部署的证书文件。

为增强安全性，应使用 setCertificateName、setCertificateCompany 或 setCertificateUnit 方法启用对 MobiLink 服务器证书的验证。

为 HTTPS 同步创建 SyncParms 类对象后，由 getStreamParms 函数返回实现此接口的实例。

成员

StreamHTTPSParms 的所有成员，包括所有继承的成员。

- “getCertificateCompany 方法”一节第 210 页
- “getCertificateName 方法”一节第 210 页
- “getCertificateUnit 方法”一节第 211 页
- “getHost 方法”一节第 205 页
- “getOutputBufferSize 方法”一节第 206 页
- “getPort 方法”一节第 206 页
- “getTrustedCertificates 方法”一节第 211 页
- “getURLSuffix 方法”一节第 206 页
- “setCertificateCompany 方法”一节第 211 页
- “setCertificateName 方法”一节第 211 页
- “setCertificateUnit 方法”一节第 212 页
- “setHost 方法”一节第 207 页
- “setOutputBufferSize 方法”一节第 207 页
- “setPort 方法”一节第 208 页
- “setTrustedCertificates 方法”一节第 212 页
- “setURLSuffix 方法”一节第 208 页

getCertificateCompany 方法

返回证书公司名以对安全连接进行验证。

语法

String **StreamHTTPSParms.getCertificateCompany()**

返回值

证书公司名。

getCertificateName 方法

返回证书公用名以对安全连接进行验证。

语法

String **StreamHTTPSParms.getCertificateName()**

返回值

证书名。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

getCertificateUnit 方法

返回证书单位名以对安全连接进行验证。

语法

String **StreamHTTPSParms.getCertificateUnit()**

返回值

组织单位的名称。

getTrustedCertificates 方法

返回用于安全同步的受信任根证书的列表所在文件的名称。

语法

String **StreamHTTPSParms.getTrustedCertificates()**

返回值

受信任根证书文件的文件名。

setCertificateCompany 方法

设置证书公司名以对安全连接进行验证。

语法

void **StreamHTTPSParms.setCertificateCompany(**
String *val*
)

参数

- **val** 公司名称。

注释

缺省值为空，表示公司名不在证书中进行验证。

setCertificateName 方法

设置证书公用名以对安全连接进行验证。

语法

void **StreamHTTPSParms.setCertificateName(**
 String *val*
)

参数

- **val**　证书公用名。

注释

缺省值为空，表示公用名不在证书中进行验证。

setCertificateUnit 方法

设置证书单位名以对安全连接进行验证。

语法

void **StreamHTTPSParms.setCertificateUnit(**
 String *val*
)

参数

- **val**　公司单位名。

注释

缺省值为空，表示组织单位名不证书中进行验证。

setTrustedCertificates 方法

设置其中包含用于安全同步的受信任根证书列表的文件。

语法

void **StreamHTTPSParms.setTrustedCertificates(**
 String *filename*
) throws **ULjException**

参数

- **filename**　受信任根证书的文件名。

注释

此参数仅在 J2SE 系统中使用。

缺省值为空，表示使用系统缺省证书存储区来验证来自 MobiLink 服务器的证书链。

SyncObserver 接口

接收同步进度信息。

语法

public **SyncObserver**

注释

要在同步期间接收进度报告，您必须创建一个执行该任务的新类，并使用 setSyncObserver 函数实现此类。

下面的示例说明了一个简单的 SyncObserver 接口：

```
class MyObserver implements SyncObserver {
    public boolean syncProgress(int state, SyncResult result) {
        System.out.println(
            "sync progress state = " + state
            + " bytes sent = " + result.getSentByteCount()
            + " bytes received = " + result.getReceivedByteCount()
        );
        return false;   // Always continue synchronization.
    }
    public MyObserver() {} // The default constructor.
}
```

按如下方法启用上面的观察器类：

```
// J2ME Sample
Connection conn;
ConfigRecordStore config = DatabaseManager.createConfigurationRecordStore(
    "test.ulj"
    );
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
SyncParms.setSyncObserver(new MyObserver());
```

成员

SyncObserver 的所有成员，包括所有继承的成员。

- “syncProgress 方法”一节第 213 页

syncProgress 方法

向用户通知进度，在同步期间被调用。

语法

boolean **SyncObserver.syncProgress(**
  int *state*,

```
    SyncResult data
)
```

参数

- **state**　一个 SyncOberver.States 常量，表示同步的当前状态。
- **data**　包含最新同步结果的 SyncResult。

注释

接收和发送各种状态，这些状态采用数据包形式。由于在单个数据包中可以上载或下载多个表，因此，针对任何给定同步的 syncProgress 调用均可以跳过许多状态。

> **注意**
> 除了 SyncResult 类方法，任何其它 UltraLiteJ API 方法都不应当在 syncProgress 调用期间被调用。

另请参见

- “SyncObserver.States 接口”一节第 215 页
- “setSyncObserver 方法”一节第 231 页

返回值

返回值为真会取消同步；返回值为假则将继续同步。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SyncObserver.States 接口

定义可以用信号通知给观察器的同步状态。

语法

public **SyncObserver.States**

另请参见

- “setSyncObserver 方法”一节第 231 页
- “SyncObserver 接口”一节第 213 页

成员

SyncObserver.States 的所有成员，包括所有继承的成员。

- “CHECKING_LAST_UPLOAD 变量”一节第 215 页
- “COMMITTING_DOWNLOAD 变量”一节第 215 页
- “DISCONNECTING 变量”一节第 216 页
- “DONE 变量”一节第 216 页
- “ERROR 变量”一节第 216 页
- “FINISHING_UPLOAD 变量”一节第 216 页
- “RECEIVING_TABLE 变量”一节第 216 页
- “RECEIVING_UPLOAD_ACK 变量”一节第 217 页
- “ROLLING_BACK_DOWNLOAD 变量”一节第 217 页
- “SENDING_DOWNLOAD_ACK 变量”一节第 217 页
- “SENDING_HEADER 变量”一节第 217 页
- “SENDING_TABLE 变量”一节第 217 页
- “STARTING 变量”一节第 217 页

CHECKING_LAST_UPLOAD 变量

正在检查上次上载的状态。

语法

final int **SyncObserver.States.CHECKING_LAST_UPLOAD**

COMMITTING_DOWNLOAD 变量

语法

final int **SyncObserver.States.COMMITTING_DOWNLOAD**

注释

下载的行正在提交到数据库。

DISCONNECTING 变量

同步流正在断开连接。

语法

final int **SyncObserver.States.DISCONNECTING**

DONE 变量

同步已完成。

语法

final int **SyncObserver.States.DONE**

注释

未报告其它状态。

ERROR 变量

同步已完成但发生了错误。

语法

final int **SyncObserver.States.ERROR**

FINISHING_UPLOAD 变量

即将完成上载。

语法

final int **SyncObserver.States.FINISHING_UPLOAD**

RECEIVING_TABLE 变量

正下载一个新表。

语法

final int **SyncObserver.States.RECEIVING_TABLE**

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

RECEIVING_UPLOAD_ACK 变量

正在下载经确认的上载。

语法

final int **SyncObserver.States.RECEIVING_UPLOAD_ACK**

ROLLING_BACK_DOWNLOAD 变量

同步正在回退下载，因为在下载过程中遇到错误。

语法

final int **SyncObserver.States.ROLLING_BACK_DOWNLOAD**

SENDING_DOWNLOAD_ACK 变量

正在发送关于下载已完成的确认。

语法

final int **SyncObserver.States.SENDING_DOWNLOAD_ACK**

SENDING_HEADER 变量

已打开同步流，即将发送标头。

语法

final int **SyncObserver.States.SENDING_HEADER**

SENDING_TABLE 变量

正上载一个新表。

语法

final int **SyncObserver.States.SENDING_TABLE**

STARTING 变量

正在启动同步。未进行任何操作。

语法

final int **SyncObserver.States.STARTING**

SyncParms 类

维护数据库同步进行期间使用的参数。

语法

public **SyncParms**

注释

使用 Connection 对象的 createSyncParms 方法调用此接口。

每次只能设置一个同步命令。使用 setDownloadOnly、setPingOnly 和 setUploadOnly 方法指定这些命令。如果将这些方法中的一个设置为真，也即意味着其它方法被设置为假。

必须设置 UserName 和 Version 参数。对于每个客户端数据库，UserName 必须唯一。

根据 SyncParms 类对象的类型，使用 getStreamParms 方法配置通信流。例如，下面的代码准备并执行 HTTP 同步：

```
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTP_STREAM,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
syncParms.setPassword("ThePWDforMyUniqueMLUserID");
syncParms.getStreamParms().setHost("MyMLHost");
myConnection.synchronize(syncParms);
```

Comma Separated Lists AuthenticationParms、Publications 和 TableOrder 参数均使用包含逗号分隔的值列表的字符串值指定。可使用单引号或双引号括上列表中的各值，但没有转义字符。值中的前导空格和结尾空格会被忽略，除非用引号括上。例如：

```
syncParms.setTableOrder("'Table A',\"Table B,D\",Table C" );
```

指定“Table A”，然后是“Table B,D”，然后是“Table C”。

成员

SyncParms 的所有成员，包括所有继承的成员。

- “getAcknowledgeDownload 方法”一节第 221 页
- “getAuthenticationParms 方法”一节第 221 页
- “getLivenessTimeout 方法”一节第 222 页
- “getNewPassword 方法”一节第 222 页
- “getPassword 方法”一节第 222 页
- “getPublications 方法”一节第 223 页
- “getSendColumnNames 方法”一节第 223 页
- “getStreamParms 方法”一节第 223 页
- “getSyncObserver 方法”一节第 224 页
- “getSyncResult 方法”一节第 224 页
- “getTableOrder 方法”一节第 224 页
- “getUserName 方法”一节第 225 页
- “getVersion 方法”一节第 225 页
- “HTTP_STREAM 变量”一节第 221 页
- “HTTPS_STREAM 变量”一节第 220 页
- “isDownloadOnly 方法”一节第 225 页
- “isPingOnly 方法”一节第 226 页
- “isUploadOnly 方法”一节第 226 页
- “setAcknowledgeDownload 方法”一节第 226 页
- “setAuthenticationParms 方法”一节第 227 页
- “setDownloadOnly 方法”一节第 227 页
- “setLivenessTimeout 方法”一节第 228 页
- “setNewPassword 方法”一节第 228 页
- “setPassword 方法”一节第 229 页
- “setPingOnly 方法”一节第 229 页
- “setPublications 方法”一节第 230 页
- “setSendColumnNames 方法”一节第 230 页
- “setSyncObserver 方法”一节第 231 页
- “setTableOrder 方法”一节第 231 页
- “setUploadOnly 方法”一节第 232 页
- “setUserName 方法”一节第 232 页
- “setVersion 方法”一节第 233 页
- “SyncParms 方法”一节第 221 页

HTTPS_STREAM 变量

为安全 HTTPS 同步创建一个 SyncParms 类对象。

语法

final int **SyncParms.HTTPS_STREAM**

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见

- “createSyncParms 方法”一节第 131 页

HTTP_STREAM 变量

为 HTTP 同步创建一个 SyncParms 类对象。

语法

final int **SyncParms.HTTP_STREAM**

另请参见

- “createSyncParms 方法”一节第 131 页

SyncParms 方法

要创建 SyncParms 类对象，请使用 createSyncParms 函数。

语法

SyncParms.SyncParms()

另请参见

- “createSyncParms 方法”一节第 131 页

getAcknowledgeDownload 方法

确定远程端是否发送下载确认。

语法

abstract boolean **SyncParms.getAcknowledgeDownload()**

另请参见

- “setAcknowledgeDownload 方法”一节第 226 页

返回值

如果远程端发送下载确认，则返回 true；否则返回 false。

getAuthenticationParms 方法

返回提供给自定义用户验证脚本的参数。

语法

abstract String **SyncParms.getAuthenticationParms()**

另请参见

- “setAuthenticationParms 方法”一节第 227 页

返回值

返回验证参数的列表；如果未指定参数，则返回空值。

getLivenessTimeout 方法

返回活动超时长度（以秒为单位）。

语法

abstract int **SyncParms.getLivenessTimeout()**

另请参见

- “setLivenessTimeout 方法”一节第 228 页

返回值

超时时间。

getNewPassword 方法

返回用 setUserName 指定的用户的新 MobiLink 口令。

语法

abstract String **SyncParms.getNewPassword()**

另请参见

- “setUserName 方法”一节第 232 页
- “setNewPassword 方法”一节第 228 页

返回值

下次同步后设置的新口令。

getPassword 方法

返回用 setUserName 指定的用户的 MobiLink 口令。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

abstract String **SyncParms.getPassword()**

另请参见

- “setPassword 方法”一节第 229 页

返回值

MobiLink 用户的口令。

getPublications 方法

返回要同步的发布。

语法

abstract String **SyncParms.getPublications()**

另请参见

- “setPublications 方法”一节第 230 页

返回值

要同步的发布集。

getSendColumnNames 方法

如果将列名发送到 MobiLink 服务器，则返回 true。

语法

abstract boolean **SyncParms.getSendColumnNames()**

另请参见

- “setSendColumnNames 方法”一节第 230 页

返回值

如果发送列名，则返回 true。

getStreamParms 方法

返回用于配置同步流的参数。

语法

abstract StreamHTTPParms **SyncParms.getStreamParms()**

注释

同步流类型在创建 SyncParms 类对象时指定。

另请参见

- “createSyncParms 方法”一节第 131 页
- “StreamHTTPParms 接口”一节第 205 页
- “StreamHTTPSParms 接口”一节第 209 页

返回值

指定 HTTP 或 HTTPS 同步流参数的 StreamHTTPParms 接口或 StreamHTTPSParms 对象。该对象按引用返回。

getSyncObserver 方法

返回当前的 SyncObserver 接口。

语法

abstract SyncObserver **SyncParms.getSyncObserver()**

返回值

返回 SyncObserver 接口；如果不存在观察器，则返回空值。

getSyncResult 方法

返回包含同步状态的 SyncResult 类对象。

语法

abstract SyncResult **SyncParms.getSyncResult()**

另请参见

- “SyncResult 类”一节第 234 页

返回值

SyncResult 类对象。

getTableOrder 方法

返回将各表上载到统一数据库应遵循的顺序。

语法

abstract String **SyncParms.getTableOrder()**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见

- “setTableOrder 方法”一节第 231 页

返回值

返回一个以逗号分隔的表名列表；如果未指定表顺序，则返回空值。有关逗号分隔列表的更多信息，请参见类描述。

getUserName 方法

返回向 MobiLink 服务器唯一标识客户端的 MobiLink 用户名。

语法

abstract String **SyncParms.getUserName()**

另请参见

- “setUserName 方法”一节第 232 页

返回值

MobiLink 用户名。

getVersion 方法

返回要使用的同步脚本。

语法

abstract String **SyncParms.getVersion()**

另请参见

- “setVersion 方法”一节第 233 页

返回值

脚本版本。

isDownloadOnly 方法

确定同步是否为仅下载。

语法

abstract boolean **SyncParms.isDownloadOnly()**

另请参见

- “setDownloadOnly 方法”一节第 227 页

返回值

如果禁用了上载，则返回 true；否则返回 false。

isPingOnly 方法

确定同步是否强制 MobiLink 服务器回应，而不执行同步。

语法

abstract boolean **SyncParms.isPingOnly()**

另请参见

- “setPingOnly 方法”一节第 229 页

返回值

如果客户端仅强制服务器回应，则返回 true；否则返回 false。

isUploadOnly 方法

确定同步是否为仅上载。

语法

abstract boolean **SyncParms.isUploadOnly()**

另请参见

- “setUploadOnly 方法”一节第 232 页

返回值

如果禁用了下载，则返回 true；否则返回 false。

setAcknowledgeDownload 方法

指示远程端是否应发送下载确认。

语法

abstract void **SyncParms.setAcknowledgeDownload(**
  boolean *ack*
)

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

参数

- **ack** 如果要使客户端确认下载，则设置为 true；否则设置为 false。

注释

缺省值为 false。

另请参见

- “getAcknowledgeDownload 方法”一节第 221 页

setAuthenticationParms 方法

为自定义用户验证脚本（MobiLink authenticate_parameters 连接事件）指定参数。

语法

```
abstract void SyncParms.setAuthenticationParms(
  String v
) throws ULjException
```

参数

- **v** 以逗号分隔的验证参数列表，或空值引用。有关逗号分隔的列表的详细信息，请参见类说明。

注释

只使用前 255 个字符串并且每个字符串的长度不应超过 128 个字符（多余的字符串在发送到 MobiLink 时会被截断）。

另请参见

- “getAuthenticationParms 方法”一节第 221 页

setDownloadOnly 方法

将同步设置为仅下载。

语法

```
abstract void SyncParms.setDownloadOnly(
  boolean v
)
```

参数

- **v** 如果要禁用上载，则设置为 true；如果要启用上载，则设置为 false。

注释

缺省值为 false。指定 true 会使 setPingOnly 和 setUploadOnly 变为 false。

另请参见

- “isDownloadOnly 方法”一节第 225 页
- “setPingOnly 方法”一节第 229 页
- “setUploadOnly 方法”一节第 232 页

setLivenessTimeout 方法

设置活动超时长度（以秒为单位）。缺省值为 100 秒。

语法

abstract void **SyncParms.setLivenessTimeout(**
  int *l*
) throws **ULjException**

参数

- **l** 新活动超时值。

注释

活动超时是服务器允许远程空闲的时间长度。如果远程端在 l 秒内未与服务器通信，则服务器会认为远程端已丢失连接并终止同步。远程端将定期向服务器自动发送消息，以保持连接处于活动状态。

如果设置负值，则会抛出异常。该值可由 MobiLink 服务器更改，而不进行任何通知。如果该值设置得过低或过高，更改就会发生。

另请参见

- “getLivenessTimeout 方法”一节第 222 页

setNewPassword 方法

为用 setUserName 指定的用户设置一个新的 MobiLink 口令。

语法

abstract void **SyncParms.setNewPassword(**
  String *v*
)

参数

- **v** MobiLink 用户的新口令。

注释

新口令将在下次同步之后生效。

缺省值为空值，表示不替换口令。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见

- “getNewPassword 方法”一节第 222 页
- “setPassword 方法”一节第 229 页
- “setUserName 方法”一节第 232 页

setPassword 方法

设置用 setUserName 指定的用户的 MobiLink 口令。

语法

```
abstract void SyncParms.setPassword(
  String v
) throws ULjException
```

参数

- **v**　MobiLink 用户的口令。

注释

该用户名和口令独立于任何数据库用户 ID 和口令。该方法用于针对 MobiLink 服务器对应用程序进行验证。

缺省值为空字符串，表示没有口令。

另请参见

- “getPassword 方法”一节第 222 页
- “setNewPassword 方法”一节第 228 页
- “setUserName 方法”一节第 232 页

setPingOnly 方法

设置同步强制 MobiLink 服务器回应，而不执行同步。

语法

```
abstract void SyncParms.setPingOnly(
  boolean v
)
```

参数

- **v**　设置 true 则仅强制服务器回应；设置 false 则执行同步。

注释

缺省值为 false。指定 true 会使 setDownloadOnly 和 setUploadOnly 变为 false。

另请参见

- “isPingOnly 方法”一节第 226 页
- “setDownloadOnly 方法”一节第 227 页
- “setUploadOnly 方法”一节第 232 页

setPublications 方法

设置要同步的发布。

语法

abstract void **SyncParms.setPublications(**
String *pubs*
) throws **ULjException**

参数

- **pubs** 以逗号分隔的发布名的列表。有关逗号分隔列表的更多信息，请参见类描述。

注释

缺省设置为 Connection.SYNC_ALL，用于表示同步所有的表。要同步所有发布，请使用 Connection.SYNC_ALL_PUBS。

另请参见

- “getPublications 方法”一节第 223 页
- “SYNC_ALL 变量”一节第 126 页
- “SYNC_ALL_PUBS 变量”一节第 127 页
- “createPublication 方法”一节第 130 页

setSendColumnNames 方法

设置在同步过程中是否向 MobiLink 服务器发送列名。缺省值为 false。

语法

abstract void **SyncParms.setSendColumnNames(**
boolean *c*
)

参数

- **c** 如果应发送列名，则设置为 true

注释

仅当使用直接行 API 时，服务器才会使用列名。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见

- "getSendColumnNames 方法"一节第 223 页

setSyncObserver 方法

设置一个 SyncObserver 对象来监控同步进度。

语法

```
abstract void SyncParms.setSyncObserver(
  SyncObserver so
)
```

参数

- **so** 一个 SyncObserver。

注释

缺省值为空值，表示没有观察器。

另请参见

- "SyncObserver 接口"一节第 213 页

setTableOrder 方法

设置将各表上载到统一数据库应遵循的顺序。

语法

```
abstract void SyncParms.setTableOrder(
  String v
) throws ULjException
```

参数

- **v** 一个以逗号分隔的表名列表（按表应被同步的顺序排列）；或为空值，表示无表顺序。有关逗号分隔列表的更多信息，请参见类描述。

注释

主表应排在首位，然后列出统一数据库中含有外键关系的所有表。

由发布选择进行同步的所有表不管是否在 TableOrder 参数中加以指定都将被同步。未指定的表将按客户端数据库中外键关系的顺序进行同步。将在指定的表之后对这些表进行同步。

缺省值为空值引用，它不会替换表的缺省排序顺序。

另请参见

- “getTableOrder 方法”一节第 224 页
- “setPublications 方法”一节第 230 页

setUploadOnly 方法

将同步设置为仅上载。

语法

```
abstract void SyncParms.setUploadOnly(
  boolean v
)
```

参数

- **v**　如果要禁用下载，则设置为 true；如果要启用下载，则设置为 false。

注释

缺省值为 false。指定 true 会使 setDownloadOnly 和 setPingOnly 变为 false。

另请参见

- “isUploadOnly 方法”一节第 226 页
- “setDownloadOnly 方法”一节第 227 页
- “setPingOnly 方法”一节第 229 页

setUserName 方法

设置向 MobiLink 服务器唯一标识客户端的 MobiLink 用户名。

语法

```
abstract void SyncParms.setUserName(
  String v
) throws ULjException
```

参数

- **v**　MobiLink 用户名。

注释

该值用于确定下载内容、记录同步状态以及在同步期间发生中断时进行恢复。

该用户名和口令独立于任何数据库用户 ID 和口令。该方法用于针对 MobiLink 服务器对应用程序进行验证。

该参数在 SyncParms 类对象创建时初始化。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见

- “getUserName 方法”一节第 225 页
- “setPassword 方法”一节第 229 页
- “setNewPassword 方法”一节第 228 页
- “createSyncParms 方法”一节第 131 页

setVersion 方法

设置要使用的同步脚本。

语法

```
abstract void SyncParms.setVersion(
  String v
) throws ULjException
```

参数

- **v**　脚本版本。

注释

统一数据库中的每个同步脚本都带有版本字符串标记。例如，可以有两个不同的 download_cursor 脚本，分别用不同的版本字符串来标识。版本字符串使应用程序可以从一组同步脚本中进行选择。

该参数在 SyncParms 类对象创建时初始化。

另请参见

- “getVersion 方法”一节第 225 页
- “createSyncParms 方法”一节第 131 页

SyncResult 类

报告有关指定数据库同步的状态方面的信息。

语法

public **SyncResult**

成员

SyncResult 的所有成员，包括所有继承的成员。

- "getAuthStatus 方法"一节第 234 页
- "getAuthValue 方法"一节第 234 页
- "getCurrentTableName 方法"一节第 235 页
- "getIgnoredRows 方法"一节第 235 页
- "getReceivedByteCount 方法"一节第 235 页
- "getReceivedRowCount 方法"一节第 235 页
- "getSentByteCount 方法"一节第 236 页
- "getSentRowCount 方法"一节第 236 页
- "getStreamErrorCode 方法"一节第 236 页
- "getStreamErrorMessage 方法"一节第 236 页
- "getSyncedTableCount 方法"一节第 237 页
- "getTotalTableCount 方法"一节第 237 页
- "isUploadOK 方法"一节第 237 页

getAuthStatus 方法

返回上次同步尝试的授权状态码。

语法

abstract int **SyncResult.getAuthStatus()**

返回值

一个 AuthStatusCode 值。

getAuthValue 方法

返回自定义用户验证同步脚本中指定的值。

语法

abstract int **SyncResult.getAuthValue()**

返回值

从自定义用户验证同步脚本返回的整数。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

getCurrentTableName 方法

返回当前正在同步的表的名称。

语法

abstract String **SyncResult.getCurrentTableName()**

返回值

表名。

getIgnoredRows 方法

确定在上次同步过程中是否忽略了任何上载行。

语法

abstract boolean **SyncResult.getIgnoredRows()**

返回值

如果上次同步过程中忽略了任何已上载的行，则返回 true；否则，如果未忽略行，则返回 false。

getReceivedByteCount 方法

返回数据同步期间所收到的字节数。

语法

abstract long **SyncResult.getReceivedByteCount()**

返回值

字节数。

getReceivedRowCount 方法

返回收到的行数。

语法

abstract int **SyncResult.getReceivedRowCount()**

返回值

行数。

getSentByteCount 方法

返回数据同步期间所发送的字节数。

语法

abstract long **SyncResult.getSentByteCount()**

返回值

发送的字节数。

getSentRowCount 方法

返回发送的行数。

语法

abstract int **SyncResult.getSentRowCount()**

返回值

行数。

getStreamErrorCode 方法

返回流报告的错误代码。

语法

abstract int **SyncResult.getStreamErrorCode()**

注释

错误代码为 HTTP 响应码。

返回值

如果没有通信流错误，则返回零；否则返回来自服务器的响应码。

getStreamErrorMessage 方法

返回流报告的错误消息。

语法

abstract String **SyncResult.getStreamErrorMessage()**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

注释

错误代码为 HTTP 响应消息。

返回值

如果没有可用消息，则返回空值；否则返回响应消息。

getSyncedTableCount 方法

返回到目前为止已同步的表的数量。

语法

abstract int **SyncResult.getSyncedTableCount()**

返回值

表的数量。

getTotalTableCount 方法

返回要同步的表的数量。

语法

abstract int **SyncResult.getTotalTableCount()**

返回值

表的数量。

isUploadOK 方法

确定上次上载同步是否成功。

语法

abstract boolean **SyncResult.isUploadOK()**

返回值

如果上次上载同步成功，则返回 true；否则返回 false。

SyncResult.AuthStatusCode 接口

枚举 MobiLink 服务器返回的授权码。

语法

public **SyncResult.AuthStatusCode**

另请参见

- "getAuthStatus 方法"一节第 234 页

成员

SyncResult.AuthStatusCode 的所有成员，包括所有继承的成员。

- "EXPIRED 变量"一节第 238 页
- "IN_USE 变量"一节第 238 页
- "INVALID 变量"一节第 238 页
- "UNKNOWN 变量"一节第 239 页
- "VALID 变量"一节第 239 页
- "VALID_BUT_EXPIRES_SOON 变量"一节第 239 页

EXPIRED 变量

用户 ID 或口令已到期。授权失败。

语法

final int **SyncResult.AuthStatusCode.EXPIRED**

INVALID 变量

用户 ID 或口令不正确。授权失败。

语法

final int **SyncResult.AuthStatusCode.INVALID**

IN_USE 变量

用户 ID 已在使用。授权失败。

语法

final int **SyncResult.AuthStatusCode.IN_USE**

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

UNKNOWN 变量

授权状态为未知。

语法

final int **SyncResult.AuthStatusCode.UNKNOWN**

注释

该代码表示同步尚未执行。

VALID 变量

用户 ID 和口令在同步时有效。

语法

final int **SyncResult.AuthStatusCode.VALID**

VALID_BUT_EXPIRES_SOON 变量

用户 ID 和口令在同步时有效，但很快就要到期。

语法

final int **SyncResult.AuthStatusCode.VALID_BUT_EXPIRES_SOON**

TableSchema 接口

指定表的模式，并提供用于定义系统表名的常量。

语法

public **TableSchema**

注释

由 createTable 函数返回一个支持该接口的对象。

所有表都必须至少具有一列和一个主键。

以下示例演示了简单数据库模式的创建过程。创建的 T2 表具有两列、一个主键和一个索引。

```
// Assumes a valid Connection object conn
TableSchema table_schema;
IndexSchema index_schema;

table_schema = conn.createTable("T2");
table_schema.createColumn("num", Domain.INTEGER);
table_schema.createColumn("quantity", Domain.INTEGER);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("num", IndexSchema.ASCENDING);
index_schema = table_schema.createIndex("index1");
index_schema.addColumn("quantity", IndexSchema.ASCENDING);

conn.schemaCreateComplete();
```

主键用于唯一标识表中的每一行。主键包括的列不能为空值。主键使用 createPrimaryIndex 函数进行创建。

唯一键是一个约束，用于标识一个或多个唯一标识表中每行的列。表中任何两行的值在所有指定的列中不能相同。表可以有多个唯一约束。主键属于唯一键。唯一键使用 createUniqueKey 函数进行创建。

唯一索引用于确保表的所有索引列中不存在具有相同值的两行。每个索引键都必须是唯一的，或者至少在一列中包含空值。唯一索引使用 createUniqueIndex 函数进行创建。

不受限制的索引允许存在重复的索引条目和空值列。普通索引使用 createIndex 函数进行创建。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

成员

TableSchema 的所有成员，包括所有继承的成员。

- “createColumn 方法”一节第 243 页
- “createColumn 方法”一节第 244 页
- “createColumn 方法”一节第 244 页
- “createIndex 方法”一节第 245 页
- “createPrimaryIndex 方法”一节第 245 页
- “createUniqueIndex 方法”一节第 246 页
- “createUniqueKey 方法”一节第 246 页
- “setNoSync 方法”一节第 247 页
- “SYS_ARTICLES 变量”一节第 241 页
- “SYS_COLUMNS 变量”一节第 241 页
- “SYS_FKEY_COLUMNS 变量”一节第 241 页
- “SYS_FOREIGN_KEYS 变量”一节第 242 页
- “SYS_INDEX_COLUMNS 变量”一节第 242 页
- “SYS_INDEXES 变量”一节第 242 页
- “SYS_INTERNAL 变量”一节第 242 页
- “SYS_PRIMARY_INDEX 变量”一节第 242 页
- “SYS_PUBLICATIONS 变量”一节第 243 页
- “SYS_TABLES 变量”一节第 243 页
- “TABLE_IS_NOSYNC 变量”一节第 243 页
- “TABLE_IS_SYSTEM 变量”一节第 243 页

SYS_ARTICLES 变量

包含发布项目相关信息的系统表的名称。

语法

final String **TableSchema.SYS_ARTICLES**

SYS_COLUMNS 变量

包含数据库中表列相关信息的系统表的名称。

语法

final String **TableSchema.SYS_COLUMNS**

SYS_FKEY_COLUMNS 变量

包含外键列相关信息的系统表的名称。

语法

final String **TableSchema.SYS_FKEY_COLUMNS**

SYS_FOREIGN_KEYS 变量

包含数据库中外键相关信息的系统表的名称。

语法

final String **TableSchema.SYS_FOREIGN_KEYS**

SYS_INDEXES 变量

包含数据库中表索引相关信息的系统表的名称。

语法

final String **TableSchema.SYS_INDEXES**

SYS_INDEX_COLUMNS 变量

包含数据库中索引列相关信息的系统表的名称。

语法

final String **TableSchema.SYS_INDEX_COLUMNS**

SYS_INTERNAL 变量

包含数据库选项和内部数据库数据相关信息的系统表的名称。

语法

final String **TableSchema.SYS_INTERNAL**

SYS_PRIMARY_INDEX 变量

系统表的主键索引的名称。

语法

final String **TableSchema.SYS_PRIMARY_INDEX**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SYS_PUBLICATIONS 变量

包含数据库发布相关信息的系统表的名称。

语法

final String **TableSchema.SYS_PUBLICATIONS**

SYS_TABLES 变量

包含数据库中表相关信息的系统表的名称。

语法

final String **TableSchema.SYS_TABLES**

TABLE_IS_NOSYNC 变量

位标志，用于表明某个表是非同步表（从不进行同步的表）。

语法

final short **TableSchema.TABLE_IS_NOSYNC**

注释

可以在逻辑上将该值与 SYS_TABLES 表的 table_flags 列中的其它标志加以组合。

TABLE_IS_SYSTEM 变量

位标志，用于表明某个表是系统表。

语法

final short **TableSchema.TABLE_IS_SYSTEM**

注释

可以在逻辑上将该值与 SYS_TABLES 表的 table_flags 列中的其它标志加以组合。

createColumn 方法

创建具有固定大小类型的新列。

语法

ColumnSchema **TableSchema.createColumn(**
String *column_name*,

```
  short column_type
) throws ULjException
```

参数

- **column_name**　新列的名称。所指定的名称必须是有效的 SQL 标识符。
- **column_type**　表示固定大小列类型的一个域类型常量。

另请参见

"Domain 接口"一节第 156 页

返回值

用于使用指定名称和类型创建列的 ColumnSchema。

createColumn 方法

创建具有可变大小类型的新列。

语法

```
ColumnSchema TableSchema.createColumn(
  String column_name,
  short column_type,
  int column_size
) throws ULjException
```

参数

- **column_name**　新列的名称。所指定的名称必须是有效的 SQL 标识符。
- **column_type**　表示具有可变大小列类型（BINARY、NUMERIC、VARCHAR）的一个域类型常量。
- **column_size**　列的大小。

注释

如果列类型是固定大小的，则将忽略大小。

另请参见

"Domain 接口"一节第 156 页

返回值

用于使用指定名称和类型创建列的 ColumnSchema。

createColumn 方法

创建具有可变大小和精度类型的新列。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

```
ColumnSchema TableSchema.createColumn(
  String column_name,
  short column_type,
  int column_size,
  int column_scale
) throws ULjException
```

参数

- **column_name** 新列的名称。所指定的名称必须是有效的 SQL 标识符。
- **column_type** 表示具有可变大小和精度列类型 (NUMERIC) 的一个域类型常量。
- **column_size** 列的大小。
- **column_scale** 列的精度。

注释

如果列类型是固定的，则将忽略大小或精度。

另请参见

"Domain 接口"一节第 156 页

返回值

用于使用指定名称和类型创建列的 ColumnSchema。

createIndex 方法

创建新索引。

语法

```
IndexSchema TableSchema.createIndex(
  String index_name
) throws ULjException
```

参数

- **index_name** 索引的名称。所指定的名称必须是有效的 SQL 标识符。

返回值

用于使用指定名称创建索引的 IndexSchema。

createPrimaryIndex 方法

创建表的主索引。

语法

```
IndexSchema TableSchema.createPrimaryIndex(
  String index_name
) throws ULjException
```

参数

- **index_name**　索引的名称。所指定的名称必须是有效的 SQL 标识符。

注释

每个表必须恰好有一个主索引。主索引中的列不能为空。

返回值

用于使用指定名称创建主索引的 IndexSchema。

createUniqueIndex 方法

创建新的唯一索引。

语法

```
IndexSchema TableSchema.createUniqueIndex(
  String index_name
) throws ULjException
```

参数

- **index_name**　索引的名称。所指定的名称必须是有效的 SQL 标识符。

注释

每个索引键都必须是唯一的，或者至少在一列中包含空值。

与具有唯一键约束的列不同，具有唯一索引的列允许空值。外键可以引用主键或唯一键，但不能引用唯一索引。

返回值

用于使用指定名称创建唯一索引的 IndexSchema。

createUniqueKey 方法

创建新的唯一键。

语法

```
IndexSchema TableSchema.createUniqueKey(
  String index_name
) throws ULjException
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

参数

- **index_name**　键的名称。所指定的名称必须是有效的 SQL 标识符。

注释

唯一键是一个约束，用于标识一个或多个唯一标识表中每行的列。一个表可以有一个以上的唯一约束。

返回值

用于使用指定名称创建唯一键的 IndexSchema。

setNoSync 方法

指定表是否应进行同步。

语法

```
TableSchema TableSchema.setNoSync(
  boolean no_sync
) throws ULjException
```

参数

- **no_sync**　如果计划同步对表的任何更改，则设置为 true，否则设置为 false。

注释

如果设置 true，则不对更改信息进行维护。缺省值为 false。

返回值

定义了 NoSync 的 TableSchema。

ULjException 类

取代 UltraLiteJ 数据库抛出的异常。

语法

public **ULjException**

成员

ULjException 的所有成员，包括所有继承的成员。

- “getCausingException 方法”一节第 251 页
- “getErrorCode 方法”一节第 251 页
- “getSqlOffset 方法”一节第 251 页
- “SQLE_AGGREGATES_NOT_ALLOWED 变量”一节第 189 页
- “SQLE_ALIAS_NOT_UNIQUE 变量”一节第 189 页
- “SQLE_ALIAS_NOT_YET_DEFINED 变量”一节第 189 页
- “SQLE_AUTHENTICATION_FAILED 变量”一节第 189 页
- “SQLE_CANNOT_EXECUTE_STMT 变量”一节第 189 页
- “SQLE_CLIENT_OUT_OF_MEMORY 变量”一节第 189 页
- “SQLE_COLUMN_AMBIGUOUS 变量”一节第 190 页
- “SQLE_COLUMN_CANNOT_BE_NULL 变量”一节第 190 页
- “SQLE_COLUMN_NOT_FOUND 变量”一节第 190 页
- “SQLE_COLUMN_NOT_STREAMABLE 变量”一节第 190 页
- “SQLE_COMMUNICATIONS_ERROR 变量”一节第 190 页
- “SQLE_CONFIG_IN_USE 变量”一节第 191 页
- “SQLE_CONVERSION_ERROR 变量”一节第 191 页
- “SQLE_CURSOR_ALREADY_OPEN 变量”一节第 191 页
- “SQLE_DATABASE_ACTIVE 变量”一节第 191 页
- “SQLE_DEVICE_IO_FAILED 变量”一节第 191 页
- “SQLE_DIV_ZERO_ERROR 变量”一节第 191 页
- “SQLE_DOWNLOAD_CONFLICT 变量”一节第 192 页
- “SQLE_ERROR 变量”一节第 192 页
- “SQLE_EXISTING_PRIMARY_KEY 变量”一节第 192 页
- “SQLE_EXPRESSION_ERROR 变量”一节第 192 页
- “SQLE_FILE_BAD_DB 变量”一节第 192 页
- “SQLE_FILE_WRONG_VERSION 变量”一节第 193 页
- “SQLE_FOREIGN_KEY_NAME_NOT_FOUND 变量”一节第 193 页
- “SQLE_IDENTIFIER_TOO_LONG 变量”一节第 193 页
- “SQLE_INCOMPLETE_SYNCHRONIZATION 变量”一节第 193 页
- “SQLE_INDEX_HAS_NO_COLUMNS 变量”一节第 193 页
- “SQLE_INDEX_NOT_FOUND 变量”一节第 193 页
- “SQLE_INDEX_NOT_UNIQUE 变量”一节第 194 页
- “SQLE_INTERRUPTED 变量”一节第 194 页
- “SQLE_INVALID_COMPARISON 变量”一节第 194 页
- “SQLE_INVALID_DISTINCT_AGGREGATE 变量”一节第 194 页
- “SQLE_INVALID_DOMAIN 变量”一节第 194 页
- “SQLE_INVALID_FOREIGN_KEY_DEF 变量”一节第 195 页
- “SQLE_INVALID_GROUP_SELECT 变量”一节第 195 页
- “SQLE_INVALID_INDEX_TYPE 变量”一节第 195 页
- “SQLE_INVALID_LOGON 变量”一节第 195 页
- “SQLE_INVALID_OPTION 变量”一节第 195 页
- “SQLE_INVALID_OPTION_SETTING 变量”一节第 195 页
- “SQLE_INVALID_ORDER 变量”一节第 196 页
- “SQLE_INVALID_PARAMETER 变量”一节第 196 页

- “SQLE_INVALID_UNION 变量”一节第 196 页
- “SQLE_LOCKED 变量”一节第 196 页
- “SQLE_MAX_ROW_SIZE_EXCEEDED 变量”一节第 196 页
- “SQLE_MUST_BE_ONLY_CONNECTION 变量”一节第 197 页
- “SQLE_NAME_NOT_UNIQUE 变量”一节第 197 页
- “SQLE_NO_COLUMN_NAME 变量”一节第 197 页
- “SQLE_NO_CURRENT_ROW 变量”一节第 197 页
- “SQLE_NO_MATCHING_SELECT_ITEM 变量”一节第 198 页
- “SQLE_NO_PRIMARY_KEY 变量”一节第 198 页
- “SQLE_NOERROR 变量”一节第 197 页
- “SQLE_NOT_IMPLEMENTED 变量”一节第 197 页
- “SQLE_OVERFLOW_ERROR 变量”一节第 198 页
- “SQLE_PAGE_SIZE_TOO_BIG 变量”一节第 198 页
- “SQLE_PAGE_SIZE_TOO_SMALL 变量”一节第 198 页
- “SQLE_PARAMETER_CANNOT_BE_NULL 变量”一节第 199 页
- “SQLE_PERMISSION_DENIED 变量”一节第 199 页
- “SQLE_PRIMARY_KEY_NOT_UNIQUE 变量”一节第 199 页
- “SQLE_PUBLICATION_NOT_FOUND 变量”一节第 199 页
- “SQLE_RESOURCE_GOVERNOR_EXCEEDED 变量”一节第 199 页
- “SQLE_ROW_LOCKED 变量”一节第 199 页
- “SQLE_ROW_UPDATED_SINCE_READ 变量”一节第 200 页
- “SQLE_SCHEMA_UPGRADE_NOT_ALLOWED 变量”一节第 200 页
- “SQLE_SERVER_SYNCHRONIZATION_ERROR 变量”一节第 200 页
- “SQLE_SUBQUERY_RESULT_NOT_UNIQUE 变量”一节第 200 页
- “SQLE_SUBQUERY_SELECT_LIST 变量”一节第 200 页
- “SQLE_SYNC_INFO_INVALID 变量”一节第 201 页
- “SQLE_SYNCHRONIZATION_IN_PROGRESS 变量”一节第 201 页
- “SQLE_SYNTAX_ERROR 变量”一节第 201 页
- “SQLE_TABLE_HAS_NO_COLUMNS 变量”一节第 201 页
- “SQLE_TABLE_IN_USE 变量”一节第 201 页
- “SQLE_TABLE_NOT_FOUND 变量”一节第 201 页
- “SQLE_TOO_MANY_PUBLICATIONS 变量”一节第 202 页
- “SQLE_ULTRALITE_DATABASE_NOT_FOUND 变量”一节第 202 页
- “SQLE_ULTRALITE_OBJ_CLOSED 变量”一节第 202 页
- “SQLE_ULTRALITEJ_OPERATION_FAILED 变量”一节第 202 页
- “SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED 变量”一节第 202 页
- “SQLE_UNABLE_TO_CONNECT 变量”一节第 203 页
- “SQLE_UNCOMMITTED_TRANSACTIONS 变量”一节第 203 页
- “SQLE_UNDERFLOW 变量”一节第 203 页
- “SQLE_UNKNOWN_FUNC 变量”一节第 203 页
- “SQLE_UPLOAD_FAILED_AT_SERVER 变量”一节第 203 页
- “SQLE_VALUE_IS_NULL 变量”一节第 203 页
- “SQLE_VARIABLE_INVALID 变量”一节第 204 页
- “SQLE_WRONG_NUM_OF_INSERT_COLS 变量”一节第 204 页
- “SQLE_WRONG_PARAMETER_COUNT 变量”一节第 204 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

getCausingException 方法

返回引发该异常的 ULjException。

语法

abstract ULjException **ULjException.getCausingException()** throws **ULjException**

注释

返回值

如果没有引发的异常，则返回空值；否则返回 ULjException。

getErrorCode 方法

获取与异常关联的错误代码。

语法

abstract int **ULjException.getErrorCode()**

注释

返回值

错误代码。

getSqlOffset 方法

返回 SQL 字符串内的错误偏移。

语法

abstract int **ULjException.getSqlOffset()**

注释

返回值

如果没有与错误消息相关联的 SQL 字符串，则返回 -1；否则返回发生错误的字符串中以零为基数的偏移。

Value 接口

描述读取的行中的列值。

语法

public **Value**

基类

- “ValueReader 接口”一节第 256 页
- “ValueWriter 接口”一节第 260 页

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

成员

Value 的所有成员，包括所有继承的成员。

- “compareValue 方法”一节第 253 页
- “duplicate 方法”一节第 254 页
- “getBlobInputStream 方法”一节第 256 页
- “getBlobOutputStream 方法”一节第 260 页
- “getBoolean 方法”一节第 256 页
- “getBytes 方法”一节第 257 页
- “getClobReader 方法”一节第 257 页
- “getClobWriter 方法”一节第 260 页
- “getDate 方法”一节第 257 页
- “getDecimalNumber 方法”一节第 257 页
- “getDomain 方法”一节第 254 页
- “getDomainSize 方法”一节第 254 页
- “getDouble 方法”一节第 258 页
- “getFloat 方法”一节第 258 页
- “getInt 方法”一节第 258 页
- “getLong 方法”一节第 258 页
- “getSize 方法”一节第 254 页
- “getString 方法”一节第 259 页
- “getType 方法”一节第 255 页
- “getValue 方法”一节第 259 页
- “isNull 方法”一节第 259 页
- “release 方法”一节第 255 页
- “set 方法”一节第 261 页
- “set 方法”一节第 261 页
- “set 方法”一节第 261 页
- “set 方法”一节第 261 页
- “set 方法”一节第 262 页
- “set 方法”一节第 262 页
- “set 方法”一节第 262 页
- “set 方法”一节第 263 页
- “set 方法”一节第 263 页
- “set 方法”一节第 263 页
- “setNull 方法”一节第 263 页

compareValue 方法

比较两个值。

语法

int **Value.compareValue(**
  Value *other*
) throws **ULjException**

参数

- **other**　要与其进行比较的值。

返回值

如果 Value 接口等于另一接口，返回 0；如果小于另一接口，则返回负整数；如果大于另一接口，则返回正整数。

duplicate 方法

复制 Value 对象，并返回结果。

语法

Value **Value.duplicate()** throws **ULjException**

返回值

复制后的 Value 对象。

getDomain 方法

返回值的 Domain 对象。

语法

Domain **Value.getDomain()** throws **ULjException**

返回值

Domain 对象。

getDomainSize 方法

返回值的域大小。

语法

int **Value.getDomainSize()** throws **ULjException**

返回值

域大小。

getSize 方法

返回值的当前大小。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

语法

int **Value.getSize()** throws **ULjException**

返回值

大小。

getType 方法

返回值的域类型。

语法

int **Value.getType()** throws **ULjException**

返回值

域类型。

release 方法

关闭 Value，以释放与其关联的内存资源。

语法

void **Value.release()** throws **ULjException**

ValueReader 接口

读取 Value 对象，并将其解释为 java 变量类型。

语法

public **ValueReader**

派生类

- “Value 接口”一节第 252 页

成员

ValueReader 的所有成员，包括所有继承的成员。

- “getBlobInputStream 方法”一节第 256 页
- “getBoolean 方法”一节第 256 页
- “getBytes 方法”一节第 257 页
- “getClobReader 方法”一节第 257 页
- “getDate 方法”一节第 257 页
- “getDecimalNumber 方法”一节第 257 页
- “getDouble 方法”一节第 258 页
- “getFloat 方法”一节第 258 页
- “getInt 方法”一节第 258 页
- “getLong 方法”一节第 258 页
- “getString 方法”一节第 259 页
- “getValue 方法”一节第 259 页
- “isNull 方法”一节第 259 页

getBlobInputStream 方法

blob InputStream。

语法

java.io.InputStream **ValueReader.getBlobInputStream()** throws **ULjException**

注释

返回值的 blob InputStream。

getBoolean 方法

返回值的布尔解释。

语法

boolean **ValueReader.getBoolean()** throws **ULjException**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

返回值

布尔值。

getBytes 方法

返回值的字节数组。

语法

byte[] **ValueReader.getBytes()** throws **ULjException**

返回值

字节数组。

getClobReader 方法

返回值的 clob 读取器。

语法

java.io.Reader **ValueReader.getClobReader()** throws **ULjException**

返回值

clob 读取器。

getDate 方法

返回值的日期解释。

语法

java.util.Date **ValueReader.getDate()** throws **ULjException**

返回值

值的日期。

getDecimalNumber 方法

返回值的 DecimalNumber 解释。

语法

DecimalNumber **ValueReader.getDecimalNumber()** throws **ULjException**

返回值

DecimalNumber 值。

getDouble 方法

返回值的双精度解释。

语法

double **ValueReader.getDouble()** throws **ULjException**

返回值

双精度值。

getFloat 方法

返回值的浮点解释。

语法

float **ValueReader.getFloat()** throws **ULjException**

返回值

浮点值。

getInt 方法

返回值的整数解释。

语法

int **ValueReader.getInt()** throws **ULjException**

返回值

整数值。

getLong 方法

返回值的长整型解释。

语法

long **ValueReader.getLong()** throws **ULjException**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

返回值

长整型值。

getString 方法

返回值的字符串解释。

语法

String **ValueReader.getString()** throws **ULjException**

返回值

字符串值。

getValue 方法

返回 Value 对象。

语法

Value **ValueReader.getValue()** throws **ULjException**

isNull 方法

测试值是否为空。

语法

boolean **ValueReader.isNull()**

返回值

如果值包含空值，则返回 true；否则返回 false。

ValueWriter 接口

将 java 变量类型的值存储到 Value 对象中。

语法

public **ValueWriter**

派生类

- "Value 接口"一节第 252 页

成员

ValueWriter 的所有成员，包括所有继承的成员。

- "getBlobOutputStream 方法"一节第 260 页
- "getClobWriter 方法"一节第 260 页
- "set 方法"一节第 261 页
- "set 方法"一节第 261 页
- "set 方法"一节第 261 页
- "set 方法"一节第 261 页
- "set 方法"一节第 262 页
- "set 方法"一节第 262 页
- "set 方法"一节第 262 页
- "set 方法"一节第 263 页
- "set 方法"一节第 263 页
- "set 方法"一节第 263 页
- "setNull 方法"一节第 263 页

getBlobOutputStream 方法

返回值的 blob OutputStream。

语法

java.io.OutputStream **ValueWriter.getBlobOutputStream()** throws **ULjException**

返回值

blob OutputStream。

getClobWriter 方法

返回值的 clob 写入器。

语法

java.io.Writer **ValueWriter.getClobWriter()** throws **ULjException**

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

返回值

clob 写入器。

set 方法

设置值的布尔值。

语法

```
void ValueWriter.set(
  boolean value
) throws ULjException
```

参数

- **value** 要设置的值。

set 方法

设置值的 DecimalNumber 形式。

语法

```
void ValueWriter.set(
  DecimalNumber value
) throws ULjException
```

参数

- **value** 要设置的值。

set 方法

设置值的日期形式。

语法

```
void ValueWriter.set(
  java.util.Date value
) throws ULjException
```

参数

- **value** 要设置的值。

set 方法

设置值的整数形式。

语法

```
void ValueWriter.set(
  int value
) throws ULjException
```

参数

- **value**　要设置的值。

set 方法

设置值的长整数形式。

语法

```
void ValueWriter.set(
  long value
) throws ULjException
```

参数

- **value**　要设置的值。

set 方法

设置值的浮点形式。

语法

```
void ValueWriter.set(
  float value
) throws ULjException
```

参数

- **value**　要设置的值。

set 方法

设置值的双精度形式。

语法

```
void ValueWriter.set(
  double value
) throws ULjException
```

参数

- **value**　要设置的值。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

set 方法

设置值的字节数组形式。

语法

void **ValueWriter.set(**
  byte[] *value*
) throws **ULjException**

参数

- **value**　要设置的值。

set 方法

设置值的字符串形式。

语法

void **ValueWriter.set(**
  String *value*
) throws **ULjException**

参数

- **value**　要设置的值。

set 方法

设置值的 Value 对象。

语法

void **ValueWriter.set(**
  Value *value*
) throws **ULjException**

参数

- **value**　要设置的值。

setNull 方法

将值设置为空。

语法

void **ValueWriter.setNull()** throws **ULjException**

UltraLiteJ 系统表

目录

systable 系统表 266
syscolumn 系统表 267
sysindex 系统表 268
sysindexcolumn 系统表 269
sysinternal 系统表 270
syspublications 系统表 271
sysarticles 系统表 272
sysforeignkey 系统表 273
sysfkcol 系统表 274

systable 系统表

systable 系统表中的每一行都描述了数据库中的一个标。

列名	列类型	说明
table_id	INTEGER	表的唯一标识符。
table_name	VARCHAR(128)	表的名称。
table_flags	UNSIGNED SHORT	以下标志之一的按位组合： ● TABLE_IS_SYSTEM ● TABLE_IS_NO_SYNC
table_data	INTEGER	仅供内部使用。
table_autoinc	BIG	仅供内部使用。

约束

PRIMARY INDEX (table_id)

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

syscolumn 系统表

syscolumn 系统表中的每一行都描述了一列。

列名	列类型	说明
table_id	INTEGER	列所属表的标识符。
column_id	INTEGER	列的唯一标识符。
column_name	VARCHAR(128)	列的名称。请参见 "Domain 接口"一节第 156 页。
column_flags	TINY	以下描述属性的标志的按位组合： ● **0x01** 列在主键中。 ● **0x02** 列可为空。
column_domain	INTEGER	列域，用来指示列的域的枚举值。
column_length	INTEGER	列长度。 对于 VARCHAR 和 BINARY 类型的列（在 Domain 接口中定义），这是最大长度（单位为字节）。对于 NUMERIC 类型的列，精度值存储在第一个字节中，而小数位数则存储在第二个字节中。
column_default	TINY	该列的缺省值，它由 ColumnSchema 接口中的某个 COLUMN_DEFAULT 值指定。例如，COLUMN_DEFAULT_AUTOINC 表示自动递增缺省值。

约束

PRIMARY KEY (table_id, column_id)

sysindex 系统表

sysindex 系统表中的每一行都描述了数据库中的一个索引。

列名	列类型	说明
table_id	INTEGER	应用索引的表的唯一标识符。
index_id	INTEGER	索引的唯一标识符。
index_name	VARCHAR(128)	索引的名称。
index_flags	TINY	以下表示索引类型及其持久性的标志的按位组合： ● **0x01** 唯一键。 ● **0x02** 唯一索引。 ● **0x04** 索引是持久性的。 ● **0x08** 主键。
index_data	INTEGER	仅供内部使用。

约束

PRIMARY KEY (table_id, index_id)

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

sysindexcolumn 系统表

sysindexcolumn 系统表中的每一行都描述了 sysindex 中列出的索引的一个列。

列名	列类型	说明
table_id	INTEGER	应用索引的表的唯一标识符。
index_id	INTEGER	该索引列所属索引的唯一标识符。
order	INTEGER	索引中列的顺序。
column_id	INTEGER	被索引列的唯一标识符。
index_column_flag	TINY	指示列在索引中是按升序 (1) 还是降序 (0) 排序。

约束

PRIMARY KEY (table_id, index_id, order)

sysinternal 系统表

sysinternal 系统表中的每一行都用于存储系统选项和其它内部数据。

列名	列类型	说明
name	VARCHAR(128)	选项的名称。
value	VARCHAR(128)	选项值。

约束

PRIMARY KEY (name)

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

syspublications 系统表

syspublications 系统表中的每一行都描述了一个发布。

列名	列类型	说明
publication_id	INTEGER	发布的唯一标识符。
publication_name	VARCHAR(128)	发布的名称。
download_timestamp	TIMESTAMP	上次下载的时间。
last_sync_sent	INTEGER	跟踪发送到 MobiLink 的上载的整数。
last_sync_confirmed	INTEGER	跟踪被确认为已由 MobiLink 接收的上载的整数。

约束

PRIMARY KEY (publication_id)

sysarticles 系统表

sysarticles 系统表中的每一行都描述了一个属于发布的表。

列名	列类型	说明
publication _id	INTEGER	该项目所属的发布的标识符。
table_id	INTEGER	属于发布的表的标识符。

约束

PRIMARY KEY (publication_id, table_id)

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

sysforeignkey 系统表

sysforeignkey 系统表中的每一行都描述了一个属于表的外键。

列名	列类型	说明
table_id	INTEGER	外键所属表的标识符。
foreign_table_id	INTEGER	此外键列所指的表的标识符。
foreign_key_id	INTEGER	外键的标识符。
name	VARCHAR(128)	外键的名称。
index_name	VARCHAR(128)	外键所指索引的名称。

约束

PRIMARY KEY (table_id, foreign_key_id)

sysfkcol 系统表

sysfkcol 系统表中的每一行都描述一个外键列。

列名	列类型	说明
table_id	INTEGER	外键所适用的表的唯一标识符。
foreign_key_id	INTEGER	该列所属外键的唯一标识符。
item_no	SHORT	外键中列的顺序。
column_id	INTEGER	指示外键列的表列的唯一标识符。
foreign_column_id	INTEGER	被指示的表列的唯一标识符。

约束

PRIMARY KEY (table_id, foreign_key_id, item_no)

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

UltraLiteJ 实用程序

目录

J2SE 的实用程序 276
J2ME 的实用程序（用于 BlackBerry 智能手机） 280

实用程序随 UltraLiteJ 一起提供，用于对 UltraLiteJ 数据库执行维护和管理任务。

J2SE 的实用程序

这些实用程序仅针对 UltraLiteJ 的 J2SE 实现提供—它们不适用于与 BlackBerry 智能手机环境一起使用。

UltraLiteJ 数据库信息实用程序 (ULjInfo)

ULjInfo 实用程序显示有关现有 UltraLiteJ 数据库的信息。

语法

ULjInfo -c *filename* **-p** *password* [*options*]

选项	说明
-c *filename*	必需。要检查的 UltraLiteJ 数据库的文件名。
-p *password*	必需。连接到 UltraLiteJ 数据库的口令。
-q	以安静模式运行—不显示消息。
-v	显示详细消息。
-?	显示命令行用法信息。

输出示例（无详细信息）

以下是 ULjInfo 程序的输出示例（无 -v 选项）：

```
C:\ULj\bin>ULjInfo.cmd -c ..\Samples\Demo1.ulj -p sql
SQL Anywhere UltraLite J Database Information Utility
Database name: ..\Samples\Demo1.ulj
Disk file: '..\Samples\Demo1.ulj'
Database ID: 0
Page size: 1024
0 rows for next upload
Date format: YYYY-MM-DD
Date order: YMD
Nearest century: 50
Numeric precision: 30
Numeric scale: 6
Time format: HH:NN:SS.SSS
Timestamp format: YYYY-MM-DD HH:NN:SS.SSS
Timestamp increment: 1
Number of tables: 1
Number of columns: 2
Number of publications: 0
Number of tables that will always be uploaded: 0
Number of tables that are never synchronized: 0
Number of primary keys: 1
Number of foreign keys: 0
Number of indexes: 0
Last download occurred on Thu Jul 05 11:31:05 EDT 2007
Upload OK: true
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

UltraLiteJ 数据库装载实用程序 (ULjLoad)

ULjLoad 实用程序提供从 XML 源文件装载 UltraLiteJ 数据库的功能。XML 文件通常由 ULjUnload 实用程序产生，并且可以自定义。

语法

ULjLoad -c *filename* **-p** *password* [*options*] *inputfile*

选项	说明
-a	将来自 XML 文件的信息添加到现有数据库。如果未指定此选项，则创建新数据库。
-c *filename*	必需。数据库文件的名称。
-d	仅装载数据；忽略模式信息。
-f *directory*	检索列数据的目录，其中列数据大于执行 ULjUnload 的过程中通过 -b 选项指定的最大 BLOB 大小。
-i	插入用于上载同步的行。
-n	仅装载模式信息；忽略行数据。
-p *password*	必需。连接到数据库的口令。
-q	以安静模式运行—不显示消息。
-v	显示详细消息。
-y	如果输出文件存在（且未指定 -a 选项），则将其覆盖。
-?	显示命令行用法信息。
inputfile	包含 xml 语句的输入文件。

用法概览示例

ULjLoad 实用程序命令行包括 **-?** 选项时，会显示以下用法信息：

```
SQL Anywhere UltraLite J Database Load Utility
Usage: uljload [options] <XML file>
        Create and load data into a new UltraLite J database from <XML file>.

Options:
        -a      Add to existing database.
        -c      <file> Database file.
        -d      Data only -- ignore schema.
        -f      <directory>
                Directory to store columns larger than <max blob size>.
        -i      Insert rows for upload synchronization.
        -n      Schema only -- ignore data.
```

```
-p      Password to connect to database.
-q      Quiet: do not print messages.
-v      Verbose messages.
-y      Overwrite file if it already exists.
```

UltraLiteJ 数据库卸载实用程序 (ULjUnload)

ULjUnload 实用程序提供将 UltraLiteJ 数据库—数据或模式，或两者—卸载到 XML 文件的功能。

语法

ULjUnload -c *filename* **-p** *password* [*options*] *outputfile*

选项	说明
-b *max-blob-size*	输出到 XML 的 BLOB/CHAR 数据的最大大小（以字节为单位）。
-c *filename*	必需。要卸载的数据库文件的名称。
-d	仅卸载数据；不输出模式信息。
-e *table, ...*	排除列表中指定的表的数据。
-f *directory*	存储大于通过 -b 选项指定的最大 BLOB 大小的列数据的目录。
-n	仅卸载模式信息；不输出数据。
-p *password*	必需。连接到数据库的口令。
-q	以安静模式运行—不显示消息。
-t *table, ...*	仅输出列表中指定的表的数据。
-v	显示详细消息。
-y	如果输出文件已存在，则将其覆盖。
-?	显示选项用法/帮助信息。
outputfile	输出文件名（此文件包含描述数据库内容的 xml 语句）。

XML 文件内容示例

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ul:ulschema xmlns:ul="urn:ultralite">
 <collation name="1252LATIN1" case_sensitive="no"/>
 <options>
  <option name="dateformat" value="YYYY-MM-DD"/>
  <option name="dateorder" value="YMD"/>
  <option name="nearestcentury" value="50"/>
  <option name="precision" value="30"/>
  <option name="scale" value="6"/>
```

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

```
  <option name="timeformat" value="HH:NN:SS.SSS"/>
  <option name="timestampformat" value="YYYY-MM-DD HH:NN:SS.SSS"/>
  <option name="timestampincrement" value="1"/>
 </options>
 <tables>
  <table name="ULCustomer" sync="changes">
   <columns>
    <column name="cust_id" type="integer" null="no"/>
    <column name="cust_name" type="char(30)" null="yes"/>
   </columns>
   <primarykey>
    <primarycolumn name="cust_id" direction="asc"/>
   </primarykey>
   <indexes/>
  </table>
 </tables>
 <uldata>
  <table name="ULCustomer">
   <row cust_id="2000" cust_name="Apple St. Builders"/>
   <row cust_id="2001" cust_name="Art's Renovations"/>
   <row cust_id="2002" cust_name="Awnings R Us"/>
   <row cust_id="2003" cust_name="Al's Interior Design"/>
   <row cust_id="2004" cust_name="Alpha Hardware"/>
   <row cust_id="2005" cust_name="Ace Properties"/>
   <row cust_id="2006" cust_name="A1 Contracting"/>
   <row cust_id="2007" cust_name="Archibald Inc."/>
   <row cust_id="2008" cust_name="Acme Construction"/>
   <row cust_id="2009" cust_name="ABCXYZ Inc."/>
   <row cust_id="2010" cust_name="Buy It Co."/>
   <row cust_id="2011" cust_name="Bill's Cages"/>
   <row cust_id="2012" cust_name="Build-It Co."/>
   <row cust_id="2013" cust_name="Bass Interiors"/>
   <row cust_id="2014" cust_name="Burger Franchise"/>
   <row cust_id="2015" cust_name="Big City Builders"/>
   <row cust_id="2016" cust_name="Bob's Renovations"/>
   <row cust_id="2017" cust_name="Basements R Us"/>
   <row cust_id="2018" cust_name="BB Interior Design"/>
   <row cust_id="2019" cust_name="Bond Hardware"/>
   <row cust_id="2020" cust_name="Cat Properties"/>
   <row cust_id="2021" cust_name="C &amp; C Contracting"/>
   <row cust_id="2022" cust_name="Classy Inc."/>
   <row cust_id="2023" cust_name="Cooper Construction"/>
   <row cust_id="2024" cust_name="City Schools"/>
   <row cust_id="2025" cust_name="Can Do It Co."/>
   <row cust_id="2026" cust_name="City Corrections"/>
   <row cust_id="2027" cust_name="City Sports Arenas"/>
   <row cust_id="2028" cust_name="Cantaloupe Interiors"/>
   <row cust_id="2029" cust_name="Chicken Franchise"/>
  </table>
 </uldata>
</ul:ulschema>
```

J2ME 的实用程序（用于 BlackBerry 智能手机）

这些实用程序仅适于与 BlackBerry 智能手机环境一起使用。

UltraLiteJ 数据库传输实用程序 (ULjDbT)

ULjDbT 实用程序提供了将 UltraLiteJ 数据库从 BlackBerry 智能手机传送到外部设备（例如，台式计算机、膝上型计算机或服务器）的功能。此外，您还可以删除数据库、显示数据库信息或者查看或以电子邮件形式发送数据库传输日志。该实用程序由两个必须同时运行的应用程序组成—UltraLiteJ 数据库传输桌面应用程序 (ULjDbt) 和 BlackBerry 智能手机客户端应用程序 (*ULjDatabaseTransfer.cod*)。

UltraLiteJ 数据库传输桌面应用程序

此桌面应用程序通过 USB 或 HTTP 连接方法接收 UltraLiteJ 数据库。启动该服务器应用程序后，它会等待 BlackBerry 智能手机通过与客户端应用程序的指定连接传送数据库。连接在应用程序超时时通过应用程序接口手动关闭，或者在完成传送后自行关闭。

BlackBerry 智能手机客户端应用程序

BlackBerry 智能手机客户端应用程序通过 USB 电缆或通过指定给桌面应用程序的 TCP 端口发送 UltraLiteJ 数据库。

此外，您还可以删除数据库、显示数据库信息或者查看或以电子邮件形式发送数据库传输日志。

客户端应用程序是 SQL Anywhere 安装目录的 *UltraLite\UltraLiteJ\J2meRim11* 目录中的签名文件。

♦ 启动客户端应用程序

1. 从 SQL Anywhere 安装目录的 *UltraLite\UltraLiteJ\J2meRim11* 目录中，装载 *ULjDatabaseTransfer.cod*。

   客户端应用程序图标随即出现在应用程序列表中。

2. 启动应用程序并按转动拨轮。
3. 在 [**数据库连接**] 屏幕上完成以下字段：
   - **Database Name** 要传送到外部设备的数据库的名称。
   - **[Database Password]** 允许传送数据的数据库口令。
4. 单击 [**下一步**]。将出现 [**操作**] 屏幕。通过该屏幕可以访问所有客户端应用程序功能。

♦ 使用 BlackBerry 智能手机客户端应用程序传输数据库

1. 在 [**操作**] 屏幕上，选择所需的连接方法（USB 或 HTTP）。
2. 对于 USB 传输，选择 [**通过 USB 将数据库传输到服务器**]。对于 HTTP 传输，请继续进行第 4 步。

3. 按照说明启动数据库传输桌面应用程序（请参见下面的**使用 UltraLiteJ 数据库传输应用程序接收数据库**）。

> **注意**
> 要确保数据库传输成功，应确保设备或模拟器已连接到 BlackBerry Device Manager。对于模拟器，应确保使用 [**USB Cable Connected**] 模拟 USB 连接。

   a. 在桌面应用程序上，确保已选择 [**USB**]，然后单击 [**启动**]。
   b. 在客户端应用程序上，单击 [**下一步**]。
   c. BlackBerry 智能手机开始向外部设备传送数据库。进度信息将显示在桌面应用程序上。
   d. 在客户端和桌面应用程序上均单击 [**确定**] 以将它们关闭。

4. 对于 HTTP 传输，选择 [**通过 HTTP 将数据库传输到服务器**]。
   a. 在 [**HTTP 传输**] 屏幕上，单击 [**下一步**]。
   b. 指定以下值：
      - **Host**　台式计算机的 IP 地址。
      - **端口**　在桌面应用程序的 [**连接属性**] 中指定端口。
      - **URL 后缀**　接收传输的服务器的主机名，包括 http:// 后缀（这是必需项）。

      然后单击 [**下一步**]。

> **注意**
> 要在未识别设备上通过 BES 进行传输，请将 [**后缀**] 留空。在识别设备上，请使用后缀 **;deviceside=false**。
>
> 要通过直接 TCP 进行传输，请使用后缀 **;deviceside=true**。请注意，并不是所有的运营公司都支持此功能。
>
> 如果您知道运营公司 WAP 网关的 APN 信息，就可以使用该网关。您还必须将该信息附加到后缀。请注意，即使要通过 BES，在 BES 和运行 UltraLiteJ 数据库传输实用程序的计算机之间也可能存在开启的防火墙。在这种情况下，需要使用 SSL 隧道程序。在 [**HTTP Transfer Parms**] 屏幕上，指定在防火墙的 BES 侧运行的 SSL 服务器的端口和名称或 IP 地址。还需要指定 SSL 客户端映射到传输应用程序的端口。
>
> 如果从 BlackBerry 模拟器传输数据库，则需要运行 BlackBerry MDS 模拟器，或者将 URL 后缀 **;deviceside=true** 指定给在 BlackBerry 模拟器中运行的 UltraLiteJ 数据库传输实用程序。

   c. 在桌面应用程序上，确保已选择 [**HTTP**]，然后单击 [**启动**]。
   d. 在客户端应用程序上，单击 [**下一步**]。

      BlackBerry 智能手机开始向外部设备传送数据库。进度信息将显示在桌面应用程序上。
   e. 在客户端和桌面应用程序上均单击 [**确定**] 以将它们关闭。

♦ **使用 UltraLiteJ 数据库传输应用程序接收数据库**

1. 从 SQL Anywhere 安装程序的 *Bin32* 目录，运行 *ULjDbTServ.cmd*。

UltraLiteJ 数据库传输应用程序即装载。

2. 在 [**Connect**] 选项卡上，选择所需的 [**Connection Method**]。
3. 在 [**Connection Properties**] 下，指定以下值：
   - **端口**　此字段只适用于 HTTP 连接。键入要连接 BlackBerry 智能手机的 TCP 端口号。通常，此端口号与指定给运行于 BlackBerry 设备之上的 UltraLiteJ 数据库传输实用程序的端口号相匹配；但是，如果您使用的是 SSL，则此端口号可能会不同。
   - **BlackBerry 口令**　此字段只适用于 USB 连接。键入在所连接的 BlackBerry 智能手机被锁定时访问该手机所使用的口令。如果没有口令，则将该字段保留空白。
   - **超时**　服务器应用程序超时并关闭连接之前的空闲分钟数。
   - **输出**　指定文件名和保存传输的数据库的位置。
4. 单击 [**启动**] 打开与 BlackBerry 智能手机的连接。服务器应用程序会一直等待到其超时或建立连接为止。如果已指定了现有文件，系统会询问您是否要将其覆盖。

[**日志**] 选项卡提供了有关服务器状态和传输进度的详细信息，包括错误消息。

♦ 删除数据库

1. 在 [**操作**] 屏幕上，选择 [**删除该数据库**]。
2. 在确认对话框上，单击 [**删除**] 删除该数据库。
3. 在 [**数据库已删除**] 对话框上，单击 [**确定**] 关闭客户端。

♦ 查看数据库信息

1. 在 [**操作**] 屏幕上，选择 [**查看数据库信息**]。向下滚动以查看所有数据库信息。
2. 单击 [**上一个**] 返回到 [**操作**] 屏幕。

♦ 查看日志文件

1. 在 [**数据库连接**] 屏幕上，显示该菜单。
2. 单击 [**日志**]。将显示日志屏幕。
3. 要以电子邮件形式发送日志文件，请输入要将日志发送到的电子邮件地址，然后单击 [**发送电子邮件**]。要返回到上一个屏幕，请按返回键

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

术语表

术语表 285

术语表

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。在版本 10.0.0 中，Adaptive Server Anywhere 更名为 SQL Anywhere 服务器，SQL Anywhere Studio 更名为 SQL Anywhere。

另请参见：“SQL Anywhere”一节第 302 页

包

Java 中相关类的集合。

被引用对象

一种对象（如表），该对象在另一个对象（如视图）的定义中被直接引用。

另请参见：“主键”一节第 312 页

编码

也称作字符编码，编码是一种方法，通过该方法可以将字符集中的每个字符映射到一个或多个字节的信息，这些信息通常以十六进制数字表示。编码的一个例子是 UTF-8。

另请参见：

- “字符集”一节第 312 页
- “代码页”一节第 287 页
- “归类”一节第 291 页

标识符

用于引用数据库对象（如表或列）的字符串。标识符可以包含 A 到 Z、a 到 z、0 到 9、下划线 (_)、at 符号 (@)、数字符号 (#) 或美元符号 ($) 中的任何字符。

并发

同时执行两个或更多个独立并且可能存在竞争关系的进程。SQL Anywhere 会自动使用锁定来隔离事务，并确保每个并发应用程序看到的数据集均一致。

另请参见：

- “事务”一节第 299 页
- “隔离级别”一节第 290 页

参考数据库

MobiLink 中一种用于 UltraLite 客户端开发的 SQL Anywhere 数据库。在开发过程中，可以将一个 SQL Anywhere 数据库同时作为参考数据库和统一数据库使用。通过其它产品建立的数据库无法用作参考数据库。

参照完整性

遵守数据一致性控制规则（具体而言，不同表中主键值与外键值之间的关系）。若要实现参照完整性，每个外键中的值必须与被引用表中行的主键值相符。

另请参见：

- “主键”一节第 312 页
- “外键”一节第 304 页

策略

QAnywhere 中指定应在何时进行消息传输的方式。

插件模块

Sybase Central 中一种用于访问和管理产品的方法。当您安装相应的产品时，插件通常会自动安装并注册 Sybase Central。通常，插件在 Sybase Central 主窗口中作为顶级容器出现，并且使用产品本身的名称，如 SQL Anywhere。

另请参见： “Sybase Central”一节第 303 页

查询

一条或一组 SQL 语句，用于访问和/或操作数据库中的数据。

另请参见： “SQL”一节第 302 页

冲突解决

在 MobiLink 中，冲突解决是指一种逻辑，它指定当两个用户修改不同远程数据库上同一行时的处理方法。

重定向器

一种 Web 服务器插件，用于为客户端与 MobiLink 服务器之间的请求和响应选择发送路径。此插件还实现了负荷平衡和故障转移机制。

抽取

SQL Remote 复制中从统一数据库卸载相应结构和数据的行为。此信息用于初始化远程数据库。

另请参见： “复制”一节第 289 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

触发器

一种特殊形式的存储过程，用户运行修改数据的查询时会自动执行该存储过程。

另请参见：

- “行级触发器”一节第 291 页
- “语句级触发器”一节第 309 页
- “完整性”一节第 305 页

传输规则

QAnywhere 中用于确定何时进行消息传输、传输哪些消息以及应在何时删除消息的逻辑。

窗口

作为分析功能执行对象的行组。一个窗口可以包含一行、多行或所有行的数据，这些数据已根据窗口定义中提供的分组规格进行了分区。窗口会进行移动，以包括为输入中的当前行执行计算所需的行数或行范围。窗口结构的主要优点是，不需要执行附加查询就可以有机会对结果进行分组和分析。

创建者 ID

UltraLite Palm OS 应用程序中一种在创建应用程序时指派的 ID。

存储过程

存储过程是数据库中存储的一组 SQL 指令，用于在数据库服务器上执行一组操作或查询。

代理表

一种本地表，它所包含的元数据可以像访问本地表一样访问远程数据库服务器上的表。

另请参见： “元数据”一节第 310 页

代理 ID

另请参见： “客户端消息存储库 ID”一节第 295 页

代码页

代码页是一种将字符集的字符映射到数字表示的编码，数字表示通常是 0 到 255 之间的一个整数。例如，Windows 代码页 1252 就是一个代码页。就本文档而言，代码页和编码这两个术语可以互换。

另请参见：

- “字符集”一节第 312 页
- “编码”一节第 285 页
- “归类”一节第 291 页

DBA 权限

使用户能够在数据库中执行管理活动的权限级别。DBA 用户在缺省情况下具有 DBA 权限。

另请参见： “数据库管理员 (DBA)”一节第 301 页

dbspace

用于创建更多数据存储空间的附加数据库文件。一个数据库可以包含在最多 13 个独立的文件（一个初始文件和 12 个 dbspace）中。每个表及其索引必须包含在单个数据库文件中。SQL 命令 CREATE DBSPACE 可将新文件添加到数据库中。

另请参见： “数据库文件”一节第 302 页

动态 SQL

执行前由程序以编程方式生成的 SQL。UltraLite 动态 SQL 是一种专用于小型设备的 SQL 变体。

对象树

Sybase Central 中数据库对象的层次。对象树的顶层显示您的 Sybase Central 版本所支持的全部产品。每种产品展开后会显示其自己的对象子树。

另请参见： “Sybase Central”一节第 303 页

EBF

快速错误修正软件。快速错误修正软件是含有一个或多个错误修正软件的软件子集。错误修正软件列在更新程序的发行说明中。错误修正软件更新可能只适用于具有相同版本号的已安装软件。已对该软件执行了一些测试，但该软件尚未进行完全测试。除非您自己已验证了软件的适用性，否则不要随应用程序分发这些文件。

发布

MobiLink 或 SQL Remote 中一种用于标识将要同步的数据的数据库对象。在 MobiLink 中，发布仅存在于客户端。一个发布包括多个项目。SQL Remote 用户可以通过预订发布来接收发布。MobiLink 用户可以通过创建发布的同步预订来同步发布。

另请参见：

- “复制”一节第 289 页
- “项目”一节第 307 页
- “发布更新”一节第 288 页

发布更新

SQL Remote 复制中对一个数据库中的一个或多个发布所做更改的列表。发布更新将作为复制消息的一部分定期发送到远程数据库。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见：

- “复制”一节第 289 页
- “发布”一节第 288 页

发布者

SQL Remote 复制中数据库内可以与其它复制数据库交换复制消息的单个用户。

另请参见： “复制”一节第 289 页。

FILE

SQL Remote 复制中一种使用共享文件来交换复制消息的消息系统。它对测试以及在没有显式消息传送系统的情况下进行的安装很有用。

另请参见“复制”一节第 289 页。

分析树

查询的代数表示。

服务

在 Windows 操作系统上，服务是在运行应用程序的用户 ID 未登录时的应用程序运行方式。

服务器管理请求

一种 QAnywhere 消息，其格式设置为 XML 并发送到 QAnywhere 系统队列，作为一种管理服务器消息存储库或监控 QAnywhere 应用程序的方法。

服务器启动的同步

一种从 MobiLink 服务器启动 MobiLink 同步的方式。

服务器消息存储库

QAnywhere 中在消息传输到客户端消息存储库或 JMS 系统之前服务器上用于临时存储消息的关系数据库。消息通过服务器消息存储库在各客户端之间进行交换。

复制

在物理上不相同的数据库之间共享数据。Sybase 有三种复制技术：MobiLink、SQL Remote 和复制服务器。

复制代理

请参见： “LTM”一节第 296 页

复制服务器

Sybase 的一种基于连接的复制技术，用于与 SQL Anywhere 和 Adaptive Server Enterprise 一起使用。它专用于在一些数据库之间进行接近实时的复制。

另请参见：“LTM”一节第 296 页

复制频率

SQL Remote 复制中一项针对每个远程用户的设置，它决定发布者的消息代理向该远程用户发送复制消息的频率应为多少。

另请参见：“复制”一节第 289 页。

复制消息

SQL Remote 或复制服务器中一种在发布数据库与预订数据库之间发送的通信。消息包含复制系统所需的数据、直通语句及信息。

另请参见：

- “复制”一节第 289 页
- “发布更新”一节第 288 页

隔离级别

一个事务中的操作对其它并发事务中的操作的可见程度。隔离级别有四级，编号依次为 0 至 3。第 3 级提供最高级别的隔离。级别 0 为缺省设置。SQL Anywhere 还支持以下三个快照隔离级别：快照、语句快照和只读语句快照。

另请参见：“快照隔离”一节第 295 页

个人服务器

与客户端应用程序在同一台计算机上运行的数据库服务器。个人数据库服务器通常由单个用户在一台计算机上使用，但它可以支持来自该用户的几个并发连接。

工作表

一种内部存储区域，用于在查询优化过程中存储中间结果。

故障切换

在活动服务器、系统或网络出现故障或意外终止时切换到冗余或备用的服务器、系统或网络。故障转移会自动进行。

关系数据库管理系统 (RDBMS)

一种以相关表的形式存储数据的数据库管理系统。

另请参见：“数据库管理系统 (DBMS)”一节第 301 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

规范化

对数据库模式的改进，目的在于按照基于关系数据库理论的规则消除冗余并改善组织。

归类

定义数据库中文本属性的字符集与排序顺序的组合。对于 SQL Anywhere 数据库，缺省归类取决于运行服务器时所使用的操作系统和语言；例如，英语 Windows 系统上的缺省归类为 1252LATIN1。归类（也称作归类序列）用于对字符串进行比较和排序。

另请参见：

- “字符集”一节第 312 页
- “代码页”一节第 287 页
- “编码”一节第 285 页

行级触发器

每更改一行即执行一次的触发器。

另请参见：

- “触发器”一节第 287 页
- “语句级触发器”一节第 309 页

回退日志

对在每个未提交的事务执行过程中所做更改的记录。当收到 ROLLBACK 请求或者系统出现故障时，未提交的事务会从数据库中回退，将数据库返回其原先的状态。每个事务都有一个单独的回退日志，事务完成时日志会被删除。

另请参见： “事务”一节第 299 页

iAnywhere JDBC 驱动程序

iAnywhere JDBC 驱动程序提供了一个 JDBC 驱动程序，与纯 Java jConnect JDBC 驱动程序相比，该驱动程序拥有一些性能优势和功能优点，但它不是纯 Java 解决方案。建议在大多数情况下使用 iAnywhere JDBC 驱动程序。

另请参见：

- “JDBC”一节第 292 页
- “jConnect”一节第 292 页

InfoMaker

一种报告和数据维护工具，它用于创建复杂的表格、报告、图形、交叉表和表，并创建将这些报告用作构件块的应用程序。

Interactive SQL

一种 SQL Anywhere 应用程序，用于查询和更改数据库中的数据以及修改数据库的结构。Interactive SQL 不但提供了一个用于输入 SQL 语句的窗格，还提供了一些用于返回有关查询处理过程的信息和结果集的窗格。

JAR 文件

Java 档案文件。一种压缩的文件格式，由一个或多个用于 Java 应用程序的包的集合组成。它将安装和运行 Java 程序所需的全部资源都放在一个压缩文件中。

Java 类

Java 中的主要代码结构单元。它是组合在一起的过程和变量的集合，将过程和变量组合在一起的原因是它们都与某个特定的可识别类别有关。

jConnect

JavaSoft JDBC 标准的 Java 实现。它为 Java 开发人员提供多层和异类环境中的本地数据库访问。但在大多数情况下，iAnywhere JDBC 驱动程序是首选的 JDBC 驱动程序。

另请参见：

- “JDBC”一节第 292 页
- “iAnywhere JDBC 驱动程序”一节第 291 页

JDBC

Java 数据库连接。一种 SQL 语言编程接口，它允许 Java 应用程序访问关系数据。首选的 JDBC 驱动程序是 iAnywhere JDBC 驱动程序。

另请参见：

- “jConnect”一节第 292 页
- “iAnywhere JDBC 驱动程序”一节第 291 页

基表

永久性的数据表。有时为区别于临时表和视图，会将这种表称作**基表**。

另请参见：

- “临时表”一节第 295 页
- “视图”一节第 299 页

基于会话的同步

一种同步类型，这种同步会使数据表示在统一数据库和远程数据库都一致。MobiLink 基于会话。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

基于脚本的上载

MobiLink 中一种将上载过程自定义为使用日志文件的替代方法的方式。

基于 SQL 的同步

MobiLink 中一种使用 MobiLink 事件将表数据与支持 MobiLink 的统一数据库进行同步的方式。对于基于 SQL 的同步，可以直接使用 SQL，也可以使用面向 Java 和 .NET 平台的 MobiLink 服务器 API 返回 SQL。

基于文件的下载

在 MobiLink 中同步数据的一种方式，其中下载以文件的方式进行分发，从而支持脱机分发同步更改。

集成登录

一种登录功能，它允许将同一个用户 ID 和口令用于操作系统登录、网络登录和数据库连接。

监听器

一个程序 (dblsn)，用于 MobiLink 服务器启动的同步。监听器安装在远程设备上，它们被配置为在接收到来自通告程序的信息时启动针对设备的操作。

另请参见： “服务器启动的同步”一节第 289 页

检查点

将对数据库的所有更改都保存到数据库文件中的时间点。在其它时间，所提交的更改仅保存到事务日志中。

检查约束

对列或列集强制实施指定条件的一种限制。

另请参见：

- “约束”一节第 311 页
- “外键约束”一节第 305 页
- “主键约束”一节第 312 页
- “唯一约束”一节第 306 页

脚本

MobiLink 中为处理 MobiLink 事件而编写的代码。脚本通过编程方式控制数据交换，以满足业务需要。

另请参见： “事件模型”一节第 299 页

脚本版本

MobiLink 中为创建同步而一起应用的一组同步脚本。

校验

测试数据库、表或索引是否受到特定类型的文件损坏。

校验和

随数据库页本身一起记录的计算出的数据库页位数。校验和能够确保数据库页写入磁盘时位数相符，因此数据库管理系统可以通过它来验证数据库页的完整性。如果计数相符，即认为数据库页已成功写入。

镜像日志

另请参见： “事务日志镜像”一节第 300 页

角色

概念性数据库建模中从一个角度描述某种关系的动词或短语。您可以用两个角色来描述每种关系。例如，"包含" 和 "隶属于" 便是角色。

角色名

外键的名称。由于它命名外表和主表之间的关系，因此称作角色名。缺省情况下，角色名就是表名，除非其它外键已经使用该名称（在这种情况下，缺省的角色名是表名后接一个三位的唯一数字）。也可以自己创建角色名。

另请参见： “外键”一节第 304 页

局部临时表

一种临时表，仅在复合语句执行期间或连接结束之前存在。当您只需要将数据集装载一次时，局部临时表非常有用。缺省情况下，行会在提交时被删除。

另请参见：

- “临时表”一节第 295 页
- “全局临时表”一节第 298 页

客户端/服务器

一种软件体系结构，在这种体系结构中，一个应用程序（客户端）从另一个应用程序（服务器）获取信息并向该应用程序发送信息。这两个应用程序常位于通过网络连接的不同计算机上。

客户端消息存储库

QAnywhere 中一种用于在远程设备上存储消息的 SQL Anywhere 数据库。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

客户端消息存储库 ID

QAnywhere 中一种对客户端消息存储库进行唯一标识的 MobiLink 远程 ID。

快照隔离

一种为发出读请求的事务返回数据的已提交版本的隔离级别。SQL Anywhere 提供了以下三种快照隔离级别：快照、语句快照和只读语句快照。使用快照隔离时，读操作不会阻塞写操作。

另请参见：“隔离级别”一节第 290 页

连接

关系系统中的一种基本操作，它通过比较指定列中的值将两个或更多个表中的行链接在一起。

连接 ID

用于标识客户端应用程序与数据库之间给定连接的唯一编号。可以使用以下 SQL 语句来确定当前连接 ID：

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

连接类型

SQL Anywhere 提供了四种类型的连接：交叉连接、键连接、自然连接和使用 ON 子句的连接。

另请参见：“连接”一节第 295 页

连接配置

连接到数据库所需的一组参数，如用户名、口令和服务器名称，它们在存储后即可方便地使用。

连接启动的同步

一种 MobiLink 服务器启动的同步，在这种同步下，连接发生变化时会启动同步。

另请参见：“服务器启动的同步”一节第 289 页

连接条件

一种影响连接结果的限制。您可以通过紧跟在连接语句的后面插入 ON 子句或 WHERE 子句来指定连接条件。对于自然连接和关键连接，SQL Anywhere 会生成连接条件。

另请参见：

- “连接”一节第 295 页
- “生成的连接条件”一节第 300 页

临时表

为临时存储数据而创建的表。有两种类型：全局临时表和局部临时表。

另请参见：

- “局部临时表”一节第 294 页
- “全局临时表”一节第 298 页

LTM

日志传送管理器（Log Transfer Manager，简称 LTM）也称作复制代理。LTM 是一个与 Replication Server 一起使用的程序，它读取数据库事务日志并将提交的更改发送到 Sybase 复制服务器。

请参见： “复制服务器”一节第 290 页

轮询

在 MobiLink 服务器启动的同步中，轻量级轮询器（例如 MobiLink 监听器）从通告程序请求推式通知的方式。

另请参见： “服务器启动的同步”一节第 289 页

逻辑索引

指向物理索引的引用（指针）。磁盘上不存储逻辑索引的索引结构。

命令文件

包含 SQL 语句的文本文件。命令文件可以手工建立，也可以通过数据库实用程序自动建立。例如，dbunload 实用程序会创建一个命令文件，其中包含重新创建给定数据库所需的 SQL 语句。

MobiLink

一种基于会话的同步技术，其设计用途是将 UltraLite 和 SQL Anywhere 远程数据库与统一数据库同步。

另请参见：

- “统一数据库”一节第 303 页
- “同步”一节第 303 页
- “UltraLite”一节第 304 页

MobiLink 服务器

运行 MobiLink 同步的计算机程序，即 mlsrv11。

MobiLink 监控器

一种用于监控 MobiLink 同步的图形化工具。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

MobiLink 客户端

有两种 MobiLink 客户端。对于 SQL Anywhere 远程数据库，MobiLink 客户端是 dbmlsync 命令行实用程序。对于 UltraLite 远程数据库，MobiLink 客户端内置于 UltraLite 运行时库中。

MobiLink 系统表

MobiLink 同步所需的系统表。它们由 MobiLink 安装程序脚本安装到 MobiLink 统一数据库中。

MobiLink 用户

MobiLink 用户用于与 MobiLink 服务器进行连接。在远程数据库上创建 MobiLink 用户，然后在统一数据库中注册该用户。MobiLink 用户名完全独立于数据库用户名。

模式

数据库的结构，其中包括表、列和索引以及它们之间的关系。

内连接

一种连接，在这种连接中，仅当两个表都满足连接条件时行才会出现在结果集中。内连接是缺省设置。

另请参见：

- “连接”一节第 295 页
- “外连接”一节第 305 页

ODBC

开放式数据库连接。一种用于与数据库管理系统连接的标准 Windows 接口。ODBC 是 SQL Anywhere 所支持的几种接口之一。

ODBC 管理器

一种随 Windows 操作系统提供的 Microsoft 程序，用于设置 ODBC 数据源。

ODBC 数据源

用户要通过 ODBC 访问的数据的规范以及获取该数据时所需的信息。

PDB

Palm 数据库文件。

PowerDesigner

一种数据库建模应用程序。PowerDesigner 为设计数据库或数据仓库提供了结构化的方法。SQL Anywhere 包括 PowerDesigner 的 Physical Data Model 组件。

PowerJ

一种 Sybase 产品，用于开发 Java 应用程序。

QAnywhere

应用程序到应用程序的消息传递（包括移动设备到移动设备和移动设备与企业之间的消息传递），它使在移动或无线设备上运行的自定义程序能够与处在中央位置的服务器应用程序进行通信。

QAnywhere 代理

QAnywhere 中一种运行在客户端设备上的进程，用于监控客户端消息存储库和确定应在何时传输消息。

嵌入式 SQL

一种 C 语言程序编程接口。SQL Anywhere 嵌入式 SQL 是 ANSI 和 IBM 标准的实现。

轻量级轮询器

在 MobiLink 服务器启动的同步中，轮询来自 MobiLink 服务器的推式通知的设备应用程序。

另请参见：“服务器启动的同步”一节第 289 页

全局临时表

一种临时表，在被显式地删除之前，其数据定义对所有用户都可见。全局临时表允许用户各自打开一个表的相同实例。缺省情况下，行在提交时被删除，并且始终是在连接结束时被删除。

另请参见：

- “临时表”一节第 295 页
- “局部临时表”一节第 294 页

日志文件

SQL Anywhere 所维护的事务日志。该日志文件用于确保在出现系统或介质故障时可以恢复数据库、提高数据库性能以及使用 SQL Remote 实现数据复制。

另请参见：

- “事务日志”一节第 299 页
- “事务日志镜像”一节第 300 页
- “完全备份”一节第 305 页

散列

散列是一种将索引条目转化为键的索引优化。索引散列旨在通过将足够的行实际数据与其行 ID 包括在一起，以避免进行先查找行、后装载行然后再将行解出才能得出索引值的高开销操作。

上载

同步过程的一个阶段，在此阶段数据从远程数据库传送到统一数据库。

设备跟踪

在 MobiLink 服务器启动的同步中，允许使用标识设备的 MobiLink 用户名来对消息进行寻址的功能。

另请参见： “服务器启动的同步”一节第 289 页

实例化视图

实例化视图是指已计算并已存储在磁盘上的视图。实例化视图同时具有视图的特征（使用查询说明进行定义）和表的特征（可以对其执行大多数表操作）。

另请参见：

● “基表”一节第 292 页
● “视图”一节第 299 页

世代号

MobiLink 中的一种机制，用于强制远程数据库先上载数据，然后再应用任何其它下载文件。

另请参见： “基于文件的下载”一节第 293 页

事件模型

MobiLink 中组成同步的事件（如 begin_synchronization 和 download_cursor）序列。如果为事件创建了脚本，则会调用事件。

视图

一种作为对象存储在数据库中的 SELECT 语句。它使用户能够看到一个或多个表中的行子集或列子集。每当用户使用特定表或表组合的视图时，都将利用存储在这些表中的信息重新计算视图。视图对确保安全以及定制数据库信息的外观来使数据访问简单明了有帮助。

事务

组成一个逻辑工作单元的 SQL 语句序列。事务要么全部得到处理，要么根本不做处理。SQL Anywhere 支持事务处理，并内置了锁定功能，使并发事务能够访问数据库而又不损坏数据。事务要么以 COMMIT 语句结束，该语句使对数据的更改成为永久性更改；要么以 ROLLBACK 语句结束，该语句撤消在事务执行过程中所做的全部更改。

事务日志

一种按进行更改的顺序存储对数据库所做全部更改的文件。它会提高性能并支持在数据库文件损坏时恢复数据。

事务日志镜像

同时维护的事务日志文件的完全相同副本（可选）。每当数据库更改写入事务日志文件时，也会同时写入事务日志镜像文件。

镜像文件应该与事务日志保留在不同的设备上，这样在任意设备出现故障时，日志的其它副本会确保数据可以安全地恢复。

另请参见： “事务日志”一节第 299 页

事务完整性

MobiLink 中对整个同步系统事务的有保证维护。要么同步整个事务，要么不对事务的任何部分进行同步。

生成的连接条件

一种自动生成的对连接结果的限制。有两种类型：关键和自然。指定 KEY JOIN 或指定关键字 JOIN 但不使用关键字 CROSS、NATURAL 或 ON 时，会生成关键连接。对于关键连接，所生成的连接条件取决于表之间的外键关系。指定 NATURAL JOIN 时会生成自然连接；所生成的连接条件基于两个表中的公用列名。

另请参见：

- “连接”一节第 295 页
- “连接条件”一节第 295 页

受保护的功能

数据库服务器启动时由 -sf 选项指定的功能，该数据库服务器上运行的任何数据库都无法使用该功能。

授权选项

一种权限级别，它允许用户向其他用户授予权限。

数据操作语言 (DML)

用于操作数据库中数据的 SQL 语句子集。DML 语句可以检索、插入、更新和删除数据库中的数据。

数据定义语言 (DDL)

用于定义数据库中数据结构的 SQL 语句子集。DDL 语句可以创建、修改和删除数据库对象（如表和用户）。

数据类型

数据的格式，如 CHAR 或 NUMERIC。在 ANSI SQL 标准中，数据类型也可以包括对大小、字符集和归类的限制。

另请参见： “域”一节第 309 页

数据立方体

一种多维结果集，每一维都以不同的方式对相同的结果进行分组和排序。数据立方体提供了有关数据的综合性信息，如果不使用数据立方体，要获得同样的信息就必须进行自连接查询和相关子查询。数据立方体是 OLAP 功能的一部分。

数据库

通过主键和外键关联的表的集合。表包含数据库中的信息。表和键一起定义数据库的结构。数据库管理系统会访问此信息。

另请参见：

- “外键”一节第 304 页
- “主键”一节第 312 页
- “数据库管理系统 (DBMS)”一节第 301 页
- “关系数据库管理系统 (RDBMS)”一节第 290 页

数据库对象

包含或接收信息的数据库组件。表、索引、视图、过程和触发器便是数据库对象。

数据库服务器

对所有针对数据库信息的访问进行管理的计算机程序。SQL Anywhere 提供了两种类型的服务器：网络服务器和个人服务器。

数据库管理系统 (DBMS)

用于创建和使用数据库的程序的集合。

另请参见： “关系数据库管理系统 (RDBMS)”一节第 290 页

数据库管理员 (DBA)

具有维护数据库所需权限的用户。DBA 通常负责对数据库模式的所有更改以及管理用户和组。数据库管理员角色自动内置于数据库中，其用户 ID 为 DBA，口令是 sql。

数据库连接

客户端应用程序与数据库之间的通信渠道。必须具有有效的用户 ID 和口令才能建立连接。为用户 ID 授予的特权决定了在连接过程中可以执行的操作。

数据库名称

服务器装载数据库时为数据库指定的名称。缺省数据库名是初始数据库文件的文件名（不含扩展名）。

另请参见： “数据库文件”一节第 302 页

数据库所有者 (dbo)

一种特殊的用户，他拥有不归 SYS 所有的系统对象。

另请参见：

- “数据库管理员 (DBA)”一节第 301 页
- “SYS”一节第 303 页

数据库文件

数据库保存在一个或多个数据库文件中。其中一个是初始文件，后面的文件称作 dbspace。每个表（包括其索引）都必须包含在单个数据库文件中。

另请参见： “dbspace”一节第 288 页

死锁

一组事务会进入的一种特殊状态，在该状态下这些事务都不能继续执行。

SQL

用于与关系数据库进行通信的语言。ANSI 定义了 SQL 的标准，其最新标准是 SQL-2003。SQL 的非官方全称是结构化查询语言。

SQL Anywhere

SQL Anywhere 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。SQL Anywhere 也是包含 SQL Anywhere RDBMS、UltraLite RDBMS、MobiLink 同步软件和其它组件的软件包的名称。

SQL Remote

一种基于消息的数据复制技术，用于在统一数据库与远程数据库之间进行双向复制。统一数据库和远程数据库必须是 SQL Anywhere。

SQL 语句

包含用于将指令传递给 DBMS 的 SQL 关键字的字符串。

另请参见：

- “模式”一节第 297 页
- “SQL”一节第 302 页
- “数据库管理系统 (DBMS)”一节第 301 页

锁定

一种在同时执行多个事务的过程中保护数据完整性的并发控制机制。SQL Anywhere 会自动应用锁以防止两个连接同时更改同一数据，并防止其它连接读取正接受更改的数据。

您可以通过设置隔离级别来控制锁定。

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见：

- “隔离级别”一节第 290 页
- “并发”一节第 285 页
- “完整性”一节第 305 页

索引

一组已排序的、与基表中的一个或多个列关联的键和指针。在表中一个或多个列上设置索引可以提高性能。

Sybase Central

一种数据库管理工具，通过图形用户界面提供 SQL Anywhere 数据库设置、属性和实用程序。Sybase Central 也可用于管理其它 Sybase 产品，其中包括 MobiLink。

SYS

一种拥有大多数系统对象的特殊用户。无法以 SYS 身份登录。

统一数据库

在分布式数据库环境中，是指用于存储数据主副本的数据库。出现冲突或差异时，将把统一数据库视为具有数据的主副本。

另请参见：

- “同步”一节第 303 页
- “复制”一节第 289 页

通信流

MobiLink 中 MobiLink 客户端与 MobiLink 服务器之间进行通信时所使用的网络协议。

通告程序

一种由 MobiLink 服务器启动的同步使用的程序。通告程序集成在 MobiLink 服务器中。它们会检查统一数据库是否有推式请求，并发送推式通知。

另请参见：

- “服务器启动的同步”一节第 289 页
- “监听器”一节第 293 页

同步

利用 MobiLink 技术在数据库之间复制数据的过程。

在 SQL Remote 中，同步专指以初始数据集初始化远程数据库的过程。

另请参见：

- “MobiLink”一节第 296 页
- “SQL Remote”一节第 302 页

推式请求

在 MobiLink 服务器启动的同步中，通告程序通过检查它来确定推式通知是否需要发送到设备的结果集中的一行值。

另请参见： “服务器启动的同步”一节第 289 页

推式通知

QAnywhere 中一种从服务器传送到 QAnywhere 客户端的特殊消息，用于提示客户端启动消息传输。在 MobiLink 服务器启动的同步中，从通告程序传送到包含推式请求数据和内部信息的设备的特殊消息。

另请参见：

- “QAnywhere”一节第 298 页
- “服务器启动的同步”一节第 289 页

UltraLite

一种针对小型设备、移动设备和嵌入式设备进行了优化的数据库。所面向的平台包括手机、传呼机和个人记事本。

UltraLite 运行时

一种过程中关系数据库管理系统，其中包括一个内置 MobiLink 同步客户端。每个 UltraLite 编程接口使用的库以及 UltraLite 引擎中都包括 UltraLite 运行时。

外表

包含外键的表。

另请参见： “外键”一节第 304 页

外部登录

与远程服务器通信时使用的替代登录名和口令。缺省情况下，SQL Anywhere 每次代表其客户端连接到远程服务器时都会使用这些客户端的名称和口令。但是，您可以通过创建外部登录来替换这一缺省设置。外部登录是指与远程服务器通信时使用的替代登录名和口令。

外键

一个表中复制另一个表中主键值的一个或多个列。外键建立表间的关系。

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

另请参见：

- “主键”一节第 312 页
- “外表”一节第 304 页

外键约束

对单个列或一组列的限制，指定表中的数据与某个其它表中数据的关系。对列集施加外键约束可使这些列成为外键。

另请参见：

- “约束”一节第 311 页
- “检查约束”一节第 293 页
- “主键约束”一节第 312 页
- “唯一约束”一节第 306 页

外连接

一种保留表中所有行的连接。SQL Anywhere 支持左、右和完全外连接。左外连接保留表中位于连接运算符左侧的行，当右表中的行不满足连接条件时，它将返回空值。完全外连接保留两个表中的所有行。

另请参见：

- “连接”一节第 295 页
- “内连接”一节第 297 页

完全备份

对整个数据库和事务日志（可选）的备份。完全备份包含数据库中的所有信息，因此可以在系统或介质出现故障时提供保护。

另请参见： “增量备份”一节第 311 页

完整性

遵守完整性规则的情况，完整性规则确保数据正确并准确，而且数据库的关系结构保持不变。

另请参见： “参照完整性”一节第 286 页

网关

一种 MobiLink 对象，存储在 MobiLink 系统表或通告程序属性文件中，包含有关如何发送用于服务器启动同步的消息的信息。

另请参见： “服务器启动的同步”一节第 289 页

网络服务器

从共享公共网络的计算机接受连接的数据库服务器。

另请参见： “个人服务器”一节第 290 页

网络协议

通信类型，如 TCP/IP 或 HTTP。

维护版本

维护版本是一套完整的软件，它升级已安装的具有相同主版本号的较早版本的软件（版本号格式是 *major.minor.patch.build*）。升级程序的发行说明中列出了错误修正软件和其它更改。

唯一约束

对某个列或一组列的限制，它要求所有非空值都各不相同。一个表可以有多个唯一约束。

另请参见：

- “外键约束”一节第 305 页
- “主键约束”一节第 312 页
- “约束”一节第 311 页

谓语

一种条件表达式，可以选择性地将其与逻辑运算符 AND 和 OR 组合在一起，以组成 WHERE 或 HAVING 子句中的条件集。在 SQL 中，求值结果为 UNKNOWN 的谓语将解释为 FALSE。

位数组

位数组是一种用于有效率地存储位序列的数组数据结构。位数组与字符串类似，不同的是其各个部分由 0（零）和 1（一）而不是字符组成。位数组通常用于保存一串布尔值。

Windows

Microsoft Windows 操作系统系列，如 Windows Vista、Windows XP 和 Windows 200x。

Windows CE

请参见“Windows Mobile”一节第 306 页。

Windows Mobile

Microsoft 为移动设备制造的操作系统系列。

文件定义数据库

MobiLink 中一种用于创建下载文件的 SQL Anywhere 数据库。

另请参见： “基于文件的下载”一节第 293 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

物理索引

索引存储在磁盘上的实际索引结构。

系统表

一种表，由 SYS 或 dbo 拥有，用于保存元数据。系统表也称作数据字典表，由数据库服务器创建并维护。

系统对象

由 SYS 或 dbo 拥有的数据库对象。

系统视图

存在于每一个数据库中的一种视图，它以易于理解的格式表示系统表中包含的信息。

下载

同步过程的一个阶段，在此阶段数据从统一数据库传送到远程数据库。

相关名

查询的 FROM 子句中使用的表或视图的名称—要么是表或视图的原始名称，要么是在 FROM 子句中定义的替代名称。

项目

在 MobiLink 或 SQL Remote 中，项目是表示整个表或表中行和列子集的数据库对象。项目在发布中组合在一起。

另请参见：

- “复制”一节第 289 页
- “发布”一节第 288 页

消息存储库

QAnywhere 中客户端和服务器设备上存储消息的数据库。

另请参见：

- “客户端消息存储库”一节第 294 页
- “服务器消息存储库”一节第 289 页

消息类型

SQL Remote 复制中指定远程用户与统一数据库发布者通信方式的数据库对象。一个统一数据库可能定义了几种消息类型，这样一来，不同的远程用户就可以使用不同的消息系统与统一数据库进行通信。

另请参见：

- “复制”一节第 289 页
- “统一数据库”一节第 303 页

消息日志

可存储来自数据库服务器或 MobiLink 服务器等应用程序的消息的日志。此类信息还可以出现在消息窗口中或记录到文件中。消息日志包括信息性消息、错误、警告以及来自 MESSAGE 语句的消息。

消息系统

SQL Remote 复制中用于在统一数据库与远程数据库之间交换消息的协议。SQL Anywhere 包括对以下消息系统的支持：FILE、FTP 和 SMTP。

另请参见：

- “复制”一节第 289 页
- “FILE”一节第 289 页

卸载

卸载数据库时会将数据库的结构和/或数据导出到文本文件（如果是结构，则导出到 SQL 命令文件中；如果是数据，则导出到 ASCII 逗号分隔文件中）。使用卸载实用程序来卸载数据库。

此外，您也可以使用 UNLOAD 语句卸载数据的选定部分。

性能统计

反映数据库系统性能的值。例如，CURRREAD 统计表示数据库服务器已发出但尚未完成的文件读取次数。

业务规则

基于实际要求的准则。通常，业务规则通过检查约束、用户定义数据类型以及事务的正确使用来实现。

另请参见：

- “约束”一节第 311 页
- “用户定义数据类型”一节第 309 页

引用对象

一种对象（如视图），其定义直接引用数据库中的另一个对象（如表）。

另请参见： “外键”一节第 304 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

用户定义数据类型

请参见“域”一节第 309 页。

游标

指向结果集的已命名链接，用于通过编程接口访问和更新行。在 SQL Anywhere 中，游标支持在查询结果中进行向前和向后移动。游标由两部分组成：游标结果集（通常由 SELECT 语句定义）和游标位置。

另请参见：

- “游标结果集”一节第 309 页
- “游标位置”一节第 309 页

游标结果集

与游标关联的查询所得到的行集。

另请参见：

- “游标”一节第 309 页
- “游标位置”一节第 309 页

游标位置

指向游标结果集中一个行的指针。

另请参见：

- “游标”一节第 309 页
- “游标结果集”一节第 309 页

语句级触发器

在整个触发语句完成后执行的触发器。

另请参见：

- “触发器”一节第 287 页
- “行级触发器”一节第 291 页

域

内置数据类型的别名，其中包括适用的精度值和小数位值，还可以选择是否包括 DEFAULT 值和 CHECK 条件。SQL Anywhere 中预定义了一些域，如货币数据类型。也称作用户定义数据类型。

另请参见： “数据类型”一节第 300 页

预订

MobiLink 同步中发布与 MobiLink 用户之间的客户端数据库中的一个链接，它使发布所描述的数据能够得到同步。

SQL Remote 复制中发布与远程用户之间的一种链接，它使用户能够与统一数据库交换该发布上的更新。

另请参见：

- “发布”一节第 288 页
- “MobiLink 用户”一节第 297 页

元数据

数据的数据。元数据描述其它数据的性质和内容。

另请参见： “模式”一节第 297 页

原子事务

保证成功完成或保证根本不予完成的事务。如果错误使原子事务的一部分无法完成，则将回退事务以防止数据库处于不一致的状态。

REMOTE DBA 特权

在 SQL Remote 中，消息代理 (dbremote) 所需的权限级别。MobiLink 中 SQL Anywhere 同步客户端 (dbmlsync) 所需的权限级别。当消息代理或同步客户端作为具有该权限的用户建立连接时，它将具有完全的 DBA 访问权。如果不是通过消息代理或同步客户端进行连接，则该用户 ID 将不具有附加权限。

另请参见： “DBA 权限”一节第 288 页

远程 ID

SQL Anywhere 和 UltraLite 数据库中一种由 MobiLink 使用的唯一标识符。远程 ID 初始情况下设置为 NULL，在数据库第一次同步期间将设置为 GUID。

远程数据库

MobiLink 或 SQL Remote 中一种与统一数据库交换数据的数据库。远程数据库可以共享统一数据库中的全部或部分数据。

另请参见：

- “同步”一节第 303 页
- “统一数据库”一节第 303 页

Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

约束

对特定数据库对象（如表或列）中所包含值的限制。例如，列可以具有唯一性约束，该约束要求该列中的所有值互不相同。表可以具有外键约束，该约束指定该表中的信息与某个其它表中数据的关系。

另请参见：

- “检查约束”一节第 293 页
- “外键约束”一节第 305 页
- “主键约束”一节第 312 页
- “唯一约束”一节第 306 页

运营公司

一种 MobiLink 对象，存储在 MobiLink 系统表或通告程序属性文件中，包含有关供服务器启动的同步使用的公共运营公司的信息。

另请参见： “服务器启动的同步”一节第 289 页

增量备份

仅包含事务日志的备份，通常在两次完全备份之间使用。

另请参见： “事务日志”一节第 299 页

争用

为获取资源而竞争的行为。例如，就数据库而言，如果有两个或更多个用户试图编辑数据库的同一行，就会为获得编辑该行的权利而发生争用。

正则表达式

正则表达式是字符、通配符和运算符的序列，用于定义某种模式以在字符串内进行搜索。

直方图

直方图是列统计信息最重要的组成部分，是一种表示数据分布的方式。SQL Anywhere 维护直方图以为优化程序提供有关列值分布情况的统计信息。

直接行处理

MobiLink 中一种用于将表数据同步到 MobiLink 支持的统一数据库以外的数据源的方法。使用直接行处理时，上载和下载都可以实现。

另请参见：

- “统一数据库”一节第 303 页
- “基于 SQL 的同步”一节第 293 页

主表

包含外键关系中的主键的表。

主键

其值唯一标识表中各行的一个列或多个列。

另请参见：“外键”一节第 304 页

主键约束

一种对主键列的唯一性约束。一个表只能有一个主键约束。

另请参见：

- “约束”一节第 311 页
- “检查约束”一节第 293 页
- “外键约束”一节第 305 页
- “唯一约束”一节第 306 页
- “完整性”一节第 305 页

子查询

嵌套在 SELECT、INSERT、UPDATE 或 DELETE 语句或者其它子查询中的 SELECT 语句。

有两种类型的子查询：相关子查询和嵌套子查询。

字符串

字符串是以单引号围起的字符序列。

字符集

字符集是一组符号，包括字母、数字、空格和其它符号。字符集的一个例子是 ISO-8859-1，又称作 Latin1。

另请参见：

- “代码页”一节第 287 页
- “编码”一节第 285 页
- “归类”一节第 291 页

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

索引

其它

-a 选项
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-b 选项
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
-c 选项
  UltraLiteJ 数据库信息 [ULjInfo] 实用程序, 276
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-d 选项
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-e 选项
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
-f 选项
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-i 选项
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-n 选项
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-p 选项
  UltraLiteJ 数据库信息 [ULjInfo] 实用程序, 276
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-q 选项
  UltraLiteJ 数据库信息 [ULjInfo] 实用程序, 276
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-t 选项
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
-v 选项
  UltraLiteJ 数据库信息 [ULjInfo] 实用程序, 276
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277
-y 选项
  UltraLiteJ 数据库卸载 [ULjUnload] 实用程序, 278
  UltraLiteJ 数据库装载 [ULjLoad] 实用程序, 277

A

addColumnReference 方法
  ForeignKeySchema 接口 [UltraLiteJ API], 171
addColumn 方法
  IndexSchema 接口 [UltraLiteJ API], 175
add 方法
  DecimalNumber 接口 [UltraLiteJ API], 153
ASCENDING 变量
  IndexSchema 接口 [UltraLiteJ API], 173
AutoCommit 模式
  UltraLiteJ 开发, 22

B

BIG 变量
  Domain 接口 [UltraLiteJ API], 159
BINARY_DEFAULT 变量
  Domain 接口 [UltraLiteJ API], 159
BINARY_MAX 变量
  Domain 接口 [UltraLiteJ API], 159
BINARY_MIN 变量
  Domain 接口 [UltraLiteJ API], 159
BINARY 变量
  Domain 接口 [UltraLiteJ API], 159
BIT 变量
  Domain 接口 [UltraLiteJ API], 160
BlackBerry
  JDE Component Package, 74
  SD 卡, 7
  Signature Tool, 74
  ULjDbT 实用程序, 280
  UltraLiteJ 应用程序教程, 65
  UltraLiteJ 数据库传输实用程序, 280
  关于 UltraLiteJ 应用程序, 4
  创建 JDE 项目, 67
  创建 UltraLiteJ 应用程序, 67
  实用程序 (J2ME), 280
  对象存储, 5
  对象存储的限制, 8
  智能手机客户端应用程序, 280

添加同步函数, 76
帮助
技术支持, xii
包
术语定义, 285
被引用对象
术语定义, 285
编码
术语定义, 285
标识符
术语定义, 285
并发
术语定义, 285
并发同步
UltraLiteJ, 10
部署
UltraLiteJ 应用程序, 30
部署 UltraLiteJ 应用程序
关于, 30

C

CHARACTER_MAX 变量
Domain 接口 [UltraLiteJ API], 160
CHECKING_LAST_UPLOAD 变量
syncObserver.States 接口 [UltraLiteJ API], 215
checkpoint 方法
Connection 接口 [UltraLiteJ API], 127
CHECK 约束
术语定义, 293
重定向器
术语定义, 286
close 方法
PreparedStatement 接口 [UltraLiteJ API], 177
ResultSet 接口 [UltraLiteJ API], 181
CollectionOfValueReaders 接口 [UltraLiteJ API]
description, 91
getBlobInputStream 方法, 92
getBoolean 方法, 92
getBytes 方法, 93
getClobReader 方法, 93
getDate 方法, 93
getDecimalNumber 方法, 94
getDouble 方法, 94
getFloat 方法, 94
getInt 方法, 95
getLong 方法, 95
getOrdinal 方法, 96
getString 方法, 96
getValue 方法, 96
isNull 方法, 97
CollectionOfValueWriters 接口 [UltraLiteJ API]
description, 98
getBlobOutputStream 方法, 99
getClobWriter 方法, 99
getOrdinal 方法, 99
set(int, boolean) 方法, 100
set(int, byte[]) 方法, 102
set(int, Date) 方法, 101
set(int, DecimalNumber) 方法, 100
set(int, double) 方法, 102
set(int, float) 方法, 101
set(int, int) 方法, 100
set(int, long) 方法, 101
set(int, String) 方法, 102
set(int, Value) 方法, 103
setNull 方法, 103
COLUMN_DEFAULT_AUTOINC 变量
ColumnSchema 接口 [UltraLiteJ API], 104
COLUMN_DEFAULT_CURRENT_DATE 变量
ColumnSchema 接口 [UltraLiteJ API], 105
COLUMN_DEFAULT_CURRENT_TIMESTAMP 变量
ColumnSchema 接口 [UltraLiteJ API], 106
COLUMN_DEFAULT_CURRENT_TIME 变量
ColumnSchema 接口 [UltraLiteJ API], 105
COLUMN_DEFAULT_GLOBAL_AUTOINC 变量
ColumnSchema 接口 [UltraLiteJ API], 106
COLUMN_DEFAULT_NONE 变量
ColumnSchema 接口 [UltraLiteJ API], 106
COLUMN_DEFAULT_UNIQUE_ID 变量
ColumnSchema 接口 [UltraLiteJ API], 107
ColumnSchema 接口
UltraLiteJ, 16
ColumnSchema 接口 [UltraLiteJ API]
COLUMN_DEFAULT_AUTOINC 变量, 104
COLUMN_DEFAULT_CURRENT_DATE 变量, 105
COLUMN_DEFAULT_CURRENT_TIME 变量, 105
COLUMN_DEFAULT_CURRENT_TIMESTAMP 变量, 106
COLUMN_DEFAULT_GLOBAL_AUTOINC 变量, 106
COLUMN_DEFAULT_NONE 变量, 106

COLUMN_DEFAULT_UNIQUE_ID 变量, 107
description, 104
setDefault 方法, 107
setNullable 方法, 108
COMMITTING_DOWNLOAD 变量
syncObserver.States 接口 [UltraLiteJ API], 215
commit 方法
Connection 接口 [UltraLiteJ API], 128
UltraLiteJ 事务, 22
compareValue 方法
Value 接口 [UltraLiteJ API], 253
ConfigFile 接口 [UltraLiteJ API]
description, 109
ConfigNonPersistent 接口 [UltraLiteJ API]
description, 110
ConfigObjectStore 接口（仅限 J2ME BlackBerry）[UltraLiteJ API]
description, 111
ConfigPersistent 接口 [UltraLiteJ API]
description, 112
getAutoCheckpoint 方法, 112
getCacheSize 方法, 113
getLazyLoadIndexes 方法, 113
hasPersistentIndexes 方法, 113
setAutocheckpoint 方法, 113
setCacheSize 方法, 114
setEncryption 方法, 114
setIndexPersistence 方法, 115
setLazyLoadIndexes 方法, 115
setRowMaximumThreshold 方法, 116
setRowMinimumThreshold 方法, 116
setShadowPaging 方法, 117
setWriteAtEnd 方法, 117
writeAtEnd 方法, 118
ConfigRecordStore 接口（仅限 J2ME）[UltraLiteJ API]
description, 119
Configuration 接口 [UltraLiteJ API]
description, 120
getDatabaseName 方法, 120
getPageSize 方法, 120
setPageSize 方法, 121
setPassword 方法, 121
CONNECTED 变量
Connection 接口 [UltraLiteJ API], 124
Connection 接口 [UltraLiteJ API]
checkpoint 方法, 127
commit 方法, 128
CONNECTED 变量, 124
createDecimalNumber(int, int) 方法, 128
createDecimalNumber(int, int, String) 方法, 128
createDomain(int) 方法, 129
createDomain(int, int) 方法, 129
createDomain(int, int, int) 方法, 129
createForeignKey 方法, 130
createPublication 方法, 130
createSyncParms(int, String, String) 方法, 131
createSyncParms(String, String) 方法, 131
createTable 方法, 132
createUUIDValue 方法, 132
createValue 方法, 133
description, 122
disableSynchronization 方法, 133
dropDatabase 方法, 133
dropForeignKey 方法, 134
dropPublication 方法, 134
dropTable 方法, 134
emergencyShutdown 方法, 135
enableSynchronization 方法, 135
getDatabaseId 方法, 135
getDatabaseInfo 方法, 136
getDatabasePartitionSize 方法, 136
getDatabaseProperty 方法, 136
getLastDownloadTime 方法, 137
getOption 方法, 137
getState 方法, 138
NOT_CONNECTED 变量, 124
OPTION_DATABASE_ID 变量, 124
OPTION_DATE_FORMAT 变量, 124
OPTION_DATE_ORDER 变量, 125
OPTION_ML_REMOTE_ID 变量, 125
OPTION_NEAREST_CENTURY 变量, 125
OPTION_PRECISION 变量, 125
OPTION_SCALE 变量, 125
OPTION_TIME_FORMAT 变量, 126
OPTION_TIMESTAMP_FORMAT 变量, 126
OPTION_TIMESTAMP_INCREMENT 变量, 126
prepareStatement 方法, 138
PROPERTY_DATABASE_NAME 变量, 126
PROPERTY_PAGE_SIZE 变量, 126
release 方法, 139
renameTable 方法, 139
resetLastDownloadTime 方法, 139
rollback 方法, 140

schemaCreateBegin 方法, 140
schemaCreateComplete 方法, 140
setDatabaseId 方法, 140
setOption 方法, 141
startSynchronizationDelete 方法, 142
stopSynchronizationDelete 方法, 142
SYNC_ALL 变量, 126
SYNC_ALL_DB_PUB_NAME 变量, 127
SYNC_ALL_PUBS 变量, 127
synchronize 方法, 142
truncateTable 方法, 142
connect 方法
DatabaseManager 类 [UltraLiteJ API], 148
createColumn(String, short, int, int) 方法
TableSchema 接口 [UltraLiteJ API], 244
createColumn(String, short, int) 方法
TableSchema 接口 [UltraLiteJ API], 244
createColumn(String, short) 方法
TableSchema 接口 [UltraLiteJ API], 243
createConfigurationFile 方法
DatabaseManager 类 [UltraLiteJ API], 148
createConfigurationNonPersistent 方法
DatabaseManager 类 [UltraLiteJ API], 149
createConfigurationObjectStore 方法（仅限 J2ME BlackBerry）
DatabaseManager 类 [UltraLiteJ API], 149
createConfigurationRecordStore 方法（仅限 J2ME）
DatabaseManager 类 [UltraLiteJ API], 150
createDatabase 方法
DatabaseManager 类 [UltraLiteJ API], 150
createDecimalNumber(int, int, String) 方法
Connection 接口 [UltraLiteJ API], 128
createDecimalNumber(int, int) 方法
Connection 接口 [UltraLiteJ API], 128
createDomain(int, int, int) 方法
Connection 接口 [UltraLiteJ API], 129
createDomain(int, int) 方法
Connection 接口 [UltraLiteJ API], 129
createDomain(int) 方法
Connection 接口 [UltraLiteJ API], 129
createForeignKey 方法
Connection 接口 [UltraLiteJ API], 130
createIndex 方法
TableSchema 接口 [UltraLiteJ API], 245
createPrimaryIndex 方法
TableSchema 接口 [UltraLiteJ API], 245
createPublication 方法
Connection 接口 [UltraLiteJ API], 130
createSISHTTPListener 方法（仅限 J2ME BlackBerry）
DatabaseManager 类 [UltraLiteJ API], 150
createSyncParms(int, String, String) 方法
Connection 接口 [UltraLiteJ API], 131
createSyncParms(String, String) 方法
Connection 接口 [UltraLiteJ API], 131
createTable 方法
Connection 接口 [UltraLiteJ API], 132
createUniqueIndex 方法
TableSchema 接口 [UltraLiteJ API], 246
createUniqueKey 方法
TableSchema 接口 [UltraLiteJ API], 246
createUUIDValue 方法
Connection 接口 [UltraLiteJ API], 132
createValue 方法
Connection 接口 [UltraLiteJ API], 133
参考数据库
术语定义, 286
参照完整性
术语定义, 286
策略
术语定义, 286
插件模块
术语定义, 286
查询
术语定义, 286
查找详细信息并请求技术协助
技术支持, xiii
冲突解决
术语定义, 286
抽取
术语定义, 286
触发器
术语定义, 287
传输规则
术语定义, 287
窗口 (OLAP)
术语定义, 287
创建者 ID
术语定义, 287
从数据库表中选择数据
UltraLiteJ, 21
存储过程
术语定义, 287
错误

提供反馈, xii

D

DatabaseInfo 接口 [UltraLiteJ API]
description, 144
getCommitCount 方法, 144
getDbFormat 方法, 144
getLogSize 方法, 145
getNumberRowsToUpload 方法, 145
getPageReads 方法, 145
getPageSize 方法, 145
getPageWrites 方法, 146
getRelease 方法, 146
DatabaseManager 对象
UltraLiteJ, 13
DatabaseManager 类 [UltraLiteJ API]
connect 方法, 148
createConfigurationFile 方法, 148
createConfigurationNonPersistent 方法, 149
createConfigurationObjectStore 方法（仅限 J2ME BlackBerry）, 149
createConfigurationRecordStore 方法（仅限 J2ME）, 150
createDatabase 方法, 150
createSISHTTPListener 方法（仅限 J2ME BlackBerry）, 150
description, 147
release 方法, 151
setErrorLanguage 方法, 151
DATE 变量
Domain 接口 [UltraLiteJ API], 160
DBA 权限
术语定义, 288
DBMS
术语定义, 301
dbspaces
术语定义, 288
DCX
关于, viii
DDL
术语定义, 300
DecimalNumber 接口 [UltraLiteJ API]
add 方法, 153
description, 153
divide 方法, 153
getString 方法, 154
isNull 方法, 154
multiply 方法, 154
set 方法, 155
setNull 方法, 155
subtract 方法, 155
decrypt 方法
EncryptionControl 接口 [UltraLiteJ API], 169
DESCENDING 变量
IndexSchema 接口 [UltraLiteJ API], 174
disableSynchronization 方法
Connection 接口 [UltraLiteJ API], 133
DISCONNECTING 变量
syncObserver.States 接口 [UltraLiteJ API], 216
divide 方法
DecimalNumber 接口 [UltraLiteJ API], 153
DML
术语定义, 300
DocCommentXchange (DCX)
关于, viii
DOMAIN_MAX 变量
Domain 接口 [UltraLiteJ API], 160
Domain 接口 [UltraLiteJ API]
BIG 变量, 159
BINARY 变量, 159
BINARY_DEFAULT 变量, 159
BINARY_MAX 变量, 159
BINARY_MIN 变量, 159
BIT 变量, 160
CHARACTER_MAX 变量, 160
DATE 变量, 160
description, 156
DOMAIN_MAX 变量, 160
DOUBLE 变量, 160
getName 方法, 167
getPrecision 方法, 167
getScale 方法, 167
getSize 方法, 167
getType 方法, 168
INTEGER 变量, 161
LONGBINARY 变量, 161
LONGBINARY_DEFAULT 变量, 161
LONGBINARY_MIN 变量, 161
LONGVARCHAR 变量, 162
LONGVARCHAR_DEFAULT 变量, 162
LONGVARCHAR_MIN 变量, 162
NUMERIC 变量, 162
PRECISION_DEFAULT 变量, 162
PRECISION_MAX 变量, 163

PRECISION_MIN 变量, 163
REAL 变量, 163
SCALE_DEFAULT 变量, 163
SCALE_MAX 变量, 163
SCALE_MIN 变量, 164
SHORT 变量, 164
TIME 变量, 164
TIMESTAMP 变量, 164
TINY 变量, 164
UINT16_MAX 变量, 165
UNSIGNED_BIG 变量, 165
UNSIGNED_INTEGER 变量, 165
UNSIGNED_SHORT 变量, 165
UUID 变量, 166
VARCHAR 变量, 166
VARCHAR_DEFAULT 变量, 166
VARCHAR_MIN 变量, 166
DONE 变量
syncObserver.States 接口 [UltraLiteJ API], 216
DOUBLE 变量
Domain 接口 [UltraLiteJ API], 160
dropDatabase 方法
Connection 接口 [UltraLiteJ API], 133
dropForeignKey 方法
Connection 接口 [UltraLiteJ API], 134
dropPublication 方法
Connection 接口 [UltraLiteJ API], 134
dropTable 方法
Connection 接口 [UltraLiteJ API], 134
duplicate 方法
Value 接口 [UltraLiteJ API], 254
代理 ID
术语定义, 287
代理表
术语定义, 287
代码列表
BlackBerry 应用程序教程, 80
代码页
术语定义, 287
动态 SQL
术语定义, 288
对象树
术语定义, 288

E

EBF
术语定义, 288
emergencyShutdown 方法
Connection 接口 [UltraLiteJ API], 135
enableSynchronization 方法
Connection 接口 [UltraLiteJ API], 135
EncryptionControl 接口 [UltraLiteJ API]
decrypt 方法, 169
description, 169
encrypt 方法, 170
initialize 方法, 170
encrypt 方法
EncryptionControl 接口 [UltraLiteJ API], 170
ERROR 变量
syncObserver.States 接口 [UltraLiteJ API], 216
executeQuery 方法
PreparedStatement 接口 [UltraLiteJ API], 178
execute 方法
PreparedStatement 接口 [UltraLiteJ API], 177
EXPIRED 变量
SyncResult.AuthStatusCode 接口 [UltraLiteJ API], 238

F

FILE
术语定义, 289
FILE 消息类型
术语定义, 289
FINISHING_UPLOAD 变量
syncObserver.States 接口 [UltraLiteJ API], 216
ForeignKeySchema 接口
UltraLiteJ, 16
ForeignKeySchema 接口 [UltraLiteJ API]
addColumnReference 方法, 171
description, 171
发布
UltraLiteJ sysarticles 系统表, 272
UltraLiteJ syspublications 系统表, 271
UltraLiteJ 模式描述, 271
术语定义, 288
模式中列出的 UltraLiteJ 表, 272
发布更新
术语定义, 288
发布者
术语定义, 289
反馈
报告错误, xii
提供, xii
文档, xii

请求更新, xii
分析树
术语定义, 289
服务
术语定义, 289
服务器管理请求
术语定义, 289
服务器启动的同步
术语定义, 289
服务器消息存储库
术语定义, 289
复制
术语定义, 289
复制代理
术语定义, 289
复制服务器
术语定义, 290
复制频率
术语定义, 290
复制消息
术语定义, 290

G

getAcknowledgeDownload 方法
SyncParms 类 [UltraLiteJ API], 221
getAuthenticationParms 方法
SyncParms 类 [UltraLiteJ API], 221
getAuthStatus 方法
SyncResult 类 [UltraLiteJ API], 234
getAuthValue 方法
SyncResult 类 [UltraLiteJ API], 234
getAutoCheckpoint 方法
ConfigPersistent 接口 [UltraLiteJ API], 112
getBlobInputStream 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 92
ValueReader 接口 [UltraLiteJ API], 256
getBlobOutputStream 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 99
ValueWriter 接口 [UltraLiteJ API], 260
getBoolean 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 92
ValueReader 接口 [UltraLiteJ API], 256
getBytes 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 93
ValueReader 接口 [UltraLiteJ API], 257
getCacheSize 方法
ConfigPersistent 接口 [UltraLiteJ API], 113
getCausingException 方法
ULjException 类 [UltraLiteJ API], 251
getCertificateCompany 方法
StreamHTTPSParms 接口 [UltraLiteJ API], 210
getCertificateName 方法
StreamHTTPSParms 接口 [UltraLiteJ API], 210
getCertificateUnit 方法
StreamHTTPSParms 接口 [UltraLiteJ API], 211
getClobReader 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 93
ValueReader 接口 [UltraLiteJ API], 257
getClobWriter 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 99
ValueWriter 接口 [UltraLiteJ API], 260
getColumnCount 方法
ResultSetMetadata 接口 [UltraLiteJ API], 183
getCommitCount 方法
DatabaseInfo 接口 [UltraLiteJ API], 144
getCurrentTableName 方法
SyncResult 类 [UltraLiteJ API], 235
getDatabaseId 方法
Connection 接口 [UltraLiteJ API], 135
getDatabaseInfo 方法
Connection 接口 [UltraLiteJ API], 136
getDatabaseName 方法
Configuration 接口 [UltraLiteJ API], 120
getDatabasePartitionSize 方法
Connection 接口 [UltraLiteJ API], 136
getDatabaseProperty 方法
Connection 接口 [UltraLiteJ API], 136
getDate 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 93
ValueReader 接口 [UltraLiteJ API], 257
getDbFormat 方法
DatabaseInfo 接口 [UltraLiteJ API], 144
getDecimalNumber 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 94
ValueReader 接口 [UltraLiteJ API], 257

getDomain 方法
Value 接口 [UltraLiteJ API], 254
getDouble 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 94
ValueReader 接口 [UltraLiteJ API], 258
getErrorCode 方法
ULjException 类 [UltraLiteJ API], 251
getFloat 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 94
ValueReader 接口 [UltraLiteJ API], 258
getHost 方法
StreamHTTPParms 接口 [UltraLiteJ API], 205
getIgnoredRows 方法
SyncResult 类 [UltraLiteJ API], 235
getInt 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 95
ValueReader 接口 [UltraLiteJ API], 258
getLastDownloadTime 方法
Connection 接口 [UltraLiteJ API], 137
getLazyLoadIndexes 方法
ConfigPersistent 接口 [UltraLiteJ API], 113
getLivenessTimeout 方法
SyncParms 类 [UltraLiteJ API], 222
getLogSize 方法
DatabaseInfo 接口 [UltraLiteJ API], 145
getLong 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 95
ValueReader 接口 [UltraLiteJ API], 258
getName 方法
Domain 接口 [UltraLiteJ API], 167
getNewPassword 方法
SyncParms 类 [UltraLiteJ API], 222
getNumberRowsToUpload 方法
DatabaseInfo 接口 [UltraLiteJ API], 145
getOption 方法
Connection 接口 [UltraLiteJ API], 137
getOrdinal 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 96
CollectionOfValueWriters 接口 [UltraLiteJ API], 99
getOutputBufferSize 方法
StreamHTTPParms 接口 [UltraLiteJ API], 206
getPageReads 方法
DatabaseInfo 接口 [UltraLiteJ API], 145
getPageSize 方法
Configuration 接口 [UltraLiteJ API], 120
DatabaseInfo 接口 [UltraLiteJ API], 145
getPageWrites 方法
DatabaseInfo 接口 [UltraLiteJ API], 146
getPassword 方法
SyncParms 类 [UltraLiteJ API], 222
getPlan 方法
PreparedStatement 接口 [UltraLiteJ API], 178
getPort 方法
StreamHTTPParms 接口 [UltraLiteJ API], 206
getPrecision 方法
Domain 接口 [UltraLiteJ API], 167
getPublications 方法
SyncParms 类 [UltraLiteJ API], 223
getReceivedByteCount 方法
SyncResult 类 [UltraLiteJ API], 235
getReceivedRowCount 方法
SyncResult 类 [UltraLiteJ API], 235
getRelease 方法
DatabaseInfo 接口 [UltraLiteJ API], 146
getResultSetMetadata 方法
ResultSet 接口 [UltraLiteJ API], 181
getResultSet 方法
PreparedStatement 接口 [UltraLiteJ API], 178
getScale 方法
Domain 接口 [UltraLiteJ API], 167
getSendColumnNames 方法
SyncParms 类 [UltraLiteJ API], 223
getSentByteCount 方法
SyncResult 类 [UltraLiteJ API], 236
getSentRowCount 方法
SyncResult 类 [UltraLiteJ API], 236
getSize 方法
Domain 接口 [UltraLiteJ API], 167
Value 接口 [UltraLiteJ API], 254
getSqlOffset 方法
ULjException 类 [UltraLiteJ API], 251
getState 方法
Connection 接口 [UltraLiteJ API], 138
getStreamErrorCode 方法
SyncResult 类 [UltraLiteJ API], 236
getStreamErrorMessage 方法
SyncResult 类 [UltraLiteJ API], 236
getStreamParms 方法

SyncParms 类 [UltraLiteJ API], 223
getString 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 96
DecimalNumber 接口 [UltraLiteJ API], 154
ValueReader 接口 [UltraLiteJ API], 259
getSyncedTableCount 方法
SyncResult 类 [UltraLiteJ API], 237
getSyncObserver 方法
SyncParms 类 [UltraLiteJ API], 224
getSyncResult 方法
SyncParms 类 [UltraLiteJ API], 224
getTableOrder 方法
SyncParms 类 [UltraLiteJ API], 224
getTotalTableCount 方法
SyncResult 类 [UltraLiteJ API], 237
getTrustedCertificates 方法
StreamHTTPSParms 接口 [UltraLiteJ API], 211
getType 方法
Domain 接口 [UltraLiteJ API], 168
Value 接口 [UltraLiteJ API], 255
getUpdateCount 方法
PreparedStatement 接口 [UltraLiteJ API], 179
getURLSuffix 方法
StreamHTTPParms 接口 [UltraLiteJ API], 206
getUserName 方法
SyncParms 类 [UltraLiteJ API], 225
getValue 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 96
ValueReader 接口 [UltraLiteJ API], 259
getVersion 方法
SyncParms 类 [UltraLiteJ API], 225
隔离级别
术语定义, 290
个人服务器
术语定义, 290
工作表
术语定义, 290
故障切换
术语定义, 290
关键连接
术语定义, 300
管理
UltraLiteJ 事务, 22
规范化
术语定义, 291
归类
术语定义, 291

H

hasPersistentIndexes 方法
ConfigPersistent 接口 [UltraLiteJ API], 113
hasResultSet 方法
PreparedStatement 接口 [UltraLiteJ API], 179
HTTP_STREAM 变量
SyncParms 类 [UltraLiteJ API], 221
HTTPS_STREAM 变量
SyncParms 类 [UltraLiteJ API], 220
环境变量
命令 shell, xi
命令提示符, xi
回退
UltraLiteJ 事务, 22
回退日志
术语定义, 291
获取帮助
技术支持, xii

I

iAnywhere JDBC 驱动程序
术语定义, 291
iAnywhere 开发人员社区
新闻组, xiii
IN_USE 变量
SyncResult.AuthStatusCode 接口 [UltraLiteJ API], 238
IndexSchema 接口
UltraLiteJ, 16
IndexSchema 接口 [UltraLiteJ API]
addColumn 方法, 175
ASCENDING 变量, 173
DESCENDING 变量, 174
description, 173
PERSISTENT 变量, 174
PRIMARY_INDEX 变量, 174
UNIQUE_INDEX 变量, 174
UNIQUE_KEY 变量, 174
InfoMaker
术语定义, 291
initialize 方法
EncryptionControl 接口 [UltraLiteJ API], 170
install-dir
文档用法, x

INTEGER 变量
Domain 接口 [UltraLiteJ API], 161
Interactive SQL
术语定义, 292
INVALID 变量
SyncResult.AuthStatusCode 接口 [UltraLiteJ API], 238
isDownloadOnly 方法
SyncParms 类 [UltraLiteJ API], 225
isNull 方法
CollectionOfValueReaders 接口 [UltraLiteJ API], 97
DecimalNumber 接口 [UltraLiteJ API], 154
ValueReader 接口 [UltraLiteJ API], 259
isPingOnly 方法
SyncParms 类 [UltraLiteJ API], 226
isUploadOK 方法
SyncResult 类 [UltraLiteJ API], 237
isUploadOnly 方法
SyncParms 类 [UltraLiteJ API], 226

J

JAR 文件
术语定义, 292
Java 开发
UltraLiteJ API, 90
Java 类
术语定义, 292
jConnect
术语定义, 292
JDBC
术语定义, 292
校验
术语定义, 294
校验和
术语定义, 294
基表
术语定义, 292
基于 SQL 的同步
术语定义, 293
基于会话的同步
术语定义, 292
基于脚本的上载
术语定义, 293
基于文件的下载
术语定义, 293
集成登录
术语定义, 293
技术支持
新闻组, xiii
加密
UltraLiteJ 开发, 23
监听器
术语定义, 293
检查点
术语定义, 293
脚本
术语定义, 293
脚本版本
术语定义, 294
角色
术语定义, 294
角色名
术语定义, 294
教程
UltraLiteJ BlackBerry CustDB, 26
UltraLiteJ BlackBerry 教程, 65
创建 UltraLiteJ BlackBerry 应用程序, 67
镜像日志
术语定义, 294
局部临时表
术语定义, 294

K

开发人员社区
新闻组, xiii
客户端/服务器
术语定义, 294
客户端消息存储库
术语定义, 294
客户端消息存储库 ID
术语定义, 295
快照隔离
术语定义, 295

L

LONGBINARY_DEFAULT 变量
Domain 接口 [UltraLiteJ API], 161
LONGBINARY_MIN 变量
Domain 接口 [UltraLiteJ API], 161
LONGBINARY 变量
Domain 接口 [UltraLiteJ API], 161
LONGVARCHAR_DEFAULT 变量
Domain 接口 [UltraLiteJ API], 162

LONGVARCHAR_MIN 变量
Domain 接口 [UltraLiteJ API], 162
LONGVARCHAR 变量
Domain 接口 [UltraLiteJ API], 162
LTM
术语定义, 296
联机手册
PDF, viii
连接
UltraLiteJ 数据库, 13
术语定义, 295
连接 ID
术语定义, 295
连接对象
UltraLiteJ, 13
连接类型
术语定义, 295
连接配置
术语定义, 295
连接启动的同步
术语定义, 295
连接条件
术语定义, 295
列
UltraLiteJ syscolumn 系统表, 267
临时表
术语定义, 295
轮询
术语定义, 296
逻辑索引
术语定义, 296

M

MobiLink
术语定义, 296
MobiLink 服务器
术语定义, 296
MobiLink 监控器
术语定义, 296
MobiLink 客户端
术语定义, 297
MobiLink 系统表
术语定义, 297
MobiLink 用户
术语定义, 297
multiply 方法
DecimalNumber 接口 [UltraLiteJ API], 154
命令 shell
大括号, xi
引号, xi
括号, xi
环境变量, xi
约定, xi
命令提示符
大括号, xi
引号, xi
括号, xi
环境变量, xi
约定, xi
命令文件
术语定义, 296
模糊处理
UltraLiteJ 开发, 23
模式
UltraLiteJ, 16
术语定义, 297

N

next 方法
ResultSet 接口 [UltraLiteJ API], 182
next 方法（ResultSet 对象）
UltraLiteJ 数据检索示例, 21
NOT_CONNECTED 变量
Connection 接口 [UltraLiteJ API], 124
NUMERIC 变量
Domain 接口 [UltraLiteJ API], 162
内连接
术语定义, 297
示例代码, 36

O

ODBC
术语定义, 297
ODBC 管理器
术语定义, 297
ODBC 数据源
术语定义, 297
onError 方法
SISRequestHandler 接口（仅限 J2ME BlackBerry）[UltraLiteJ API], 185
onRequest 方法
SISRequestHandler 接口（仅限 J2ME BlackBerry）[UltraLiteJ API], 185
OPTION_DATABASE_ID 变量

Connection 接口 [UltraLiteJ API], 124
OPTION_DATE_FORMAT 变量
Connection 接口 [UltraLiteJ API], 124
OPTION_DATE_ORDER 变量
Connection 接口 [UltraLiteJ API], 125
OPTION_ML_REMOTE_ID 变量
Connection 接口 [UltraLiteJ API], 125
OPTION_NEAREST_CENTURY 变量
Connection 接口 [UltraLiteJ API], 125
OPTION_PRECISION 变量
Connection 接口 [UltraLiteJ API], 125
OPTION_SCALE 变量
Connection 接口 [UltraLiteJ API], 125
OPTION_TIME_FORMAT 变量
Connection 接口 [UltraLiteJ API], 126
OPTION_TIMESTAMP_FORMAT 变量
Connection 接口 [UltraLiteJ API], 126
OPTION_TIMESTAMP_INCREMENT 变量
Connection 接口 [UltraLiteJ API], 126

P

PDB
术语定义, 297
PDF
文档, viii
PERSISTENT 变量
IndexSchema 接口 [UltraLiteJ API], 174
PowerDesigner
术语定义, 297
PowerJ
术语定义, 298
PRECISION_DEFAULT 变量
Domain 接口 [UltraLiteJ API], 162
PRECISION_MAX 变量
Domain 接口 [UltraLiteJ API], 163
PRECISION_MIN 变量
Domain 接口 [UltraLiteJ API], 163
preparedStatement 接口
UltraLiteJ, 19
PreparedStatement 接口 [UltraLiteJ API]
close 方法, 177
description, 176
execute 方法, 177
executeQuery 方法, 178
getPlan 方法, 178
getResultSet 方法, 178
getUpdateCount 方法, 179
hasResultSet 方法, 179
prepareStatement 方法
Connection 接口 [UltraLiteJ API], 138
previous 方法
ResultSet 接口 [UltraLiteJ API], 182
previous 方法（ResultSet 对象）
UltraLiteJ 数据检索示例, 21
PRIMARY_INDEX 变量
IndexSchema 接口 [UltraLiteJ API], 174
PROPERTY_DATABASE_NAME 变量
Connection 接口 [UltraLiteJ API], 126
PROPERTY_PAGE_SIZE 变量
Connection 接口 [UltraLiteJ API], 126
配置对象
UltraLiteJ, 13

Q

QAnywhere
术语定义, 298
QAnywhere 代理
术语定义, 298
嵌入式 SQL
术语定义, 298
全局临时表
术语定义, 298

R

RDBMS
术语定义, 290
REAL 变量
Domain 接口 [UltraLiteJ API], 163
RECEIVING_TABLE 变量
syncObserver.States 接口 [UltraLiteJ API], 216
RECEIVING_UPLOAD_ACK 变量
syncObserver.States 接口 [UltraLiteJ API], 217
release 方法
Connection 接口 [UltraLiteJ API], 139
DatabaseManager 类 [UltraLiteJ API], 151
Value 接口 [UltraLiteJ API], 255
REMOTE DBA 权限
术语定义, 310
renameTable 方法
Connection 接口 [UltraLiteJ API], 139
resetLastDownloadTime 方法
Connection 接口 [UltraLiteJ API], 139
ResultSetMetadata 接口 [UltraLiteJ API]
description, 183

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

getColumnCount 方法, 183
ResultSet 对象
UltraLiteJ 数据检索示例, 21
ResultSet 接口 [UltraLiteJ API]
close 方法, 181
description, 180
getResultSetMetadata 方法, 181
next 方法, 182
previous 方法, 182
rollback 方法
Connection 接口 [UltraLiteJ API], 140
UltraLiteJ 事务, 22
ROLLING_BACK_DOWNLOAD 变量
syncObserver.States 接口 [UltraLiteJ API], 217
日志文件
术语定义, 298

S

samples-dir
文档用法, x
SCALE_DEFAULT 变量
Domain 接口 [UltraLiteJ API], 163
SCALE_MAX 变量
Domain 接口 [UltraLiteJ API], 163
SCALE_MIN 变量
Domain 接口 [UltraLiteJ API], 164
schemaCreateBegin 方法
Connection 接口 [UltraLiteJ API], 140
schemaCreateComplete 方法
Connection 接口 [UltraLiteJ API], 140
SELECT 语句
UltraLiteJ 数据检索示例, 21
SENDING_DOWNLOAD_ACK 变量
syncObserver.States 接口 [UltraLiteJ API], 217
SENDING_HEADER 变量
syncObserver.States 接口 [UltraLiteJ API], 217
SENDING_TABLE 变量
syncObserver.States 接口 [UltraLiteJ API], 217
set(boolean) 方法
ValueWriter 接口 [UltraLiteJ API], 261
set(byte[]) 方法
ValueWriter 接口 [UltraLiteJ API], 263
set(Date) 方法
ValueWriter 接口 [UltraLiteJ API], 261
set(DecimalNumber) 方法
ValueWriter 接口 [UltraLiteJ API], 261
set(double) 方法
ValueWriter 接口 [UltraLiteJ API], 262
set(float) 方法
ValueWriter 接口 [UltraLiteJ API], 262
set(int, boolean) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 100
set(int, byte[]) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 102
set(int, Date) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 101
set(int, DecimalNumber) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 100
set(int, double) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 102
set(int, float) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 101
set(int, int) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 100
set(int, long) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 101
set(int, String) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 102
set(int, Value) 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 103
set(int) 方法
ValueWriter 接口 [UltraLiteJ API], 261
set(long) 方法
ValueWriter 接口 [UltraLiteJ API], 262
set(String) 方法
ValueWriter 接口 [UltraLiteJ API], 263
set(Value) 方法
ValueWriter 接口 [UltraLiteJ API], 263
setAcknowledgeDownload 方法
SyncParms 类 [UltraLiteJ API], 226
setAuthenticationParms 方法
SyncParms 类 [UltraLiteJ API], 227
setAutocheckpoint 方法
ConfigPersistent 接口 [UltraLiteJ API], 113

setCacheSize 方法
ConfigPersistent 接口 [UltraLiteJ API], 114
setCertificateCompany 方法
StreamHTTPSParms 接口 [UltraLiteJ API], 211
setCertificateName 方法
StreamHTTPSParms 接口 [UltraLiteJ API], 211
setCertificateUnit 方法
StreamHTTPSParms 接口 [UltraLiteJ API], 212
setDatabaseId 方法
Connection 接口 [UltraLiteJ API], 140
setDefault 方法
ColumnSchema 接口 [UltraLiteJ API], 107
setDownloadOnly 方法
SyncParms 类 [UltraLiteJ API], 227
setEncryption 方法
ConfigPersistent 接口 [UltraLiteJ API], 114
setErrorLanguage 方法
DatabaseManager 类 [UltraLiteJ API], 151
setHost 方法
StreamHTTPParms 接口 [UltraLiteJ API], 207
setIndexPersistence 方法
ConfigPersistent 接口 [UltraLiteJ API], 115
setLazyLoadIndexes 方法
ConfigPersistent 接口 [UltraLiteJ API], 115
setLivenessTimeout 方法
SyncParms 类 [UltraLiteJ API], 228
setNewPassword 方法
SyncParms 类 [UltraLiteJ API], 228
setNoSync 方法
TableSchema 接口 [UltraLiteJ API], 247
setNullable 方法
ColumnSchema 接口 [UltraLiteJ API], 108
setNull 方法
CollectionOfValueWriters 接口 [UltraLiteJ API], 103
DecimalNumber 接口 [UltraLiteJ API], 155
ValueWriter 接口 [UltraLiteJ API], 263
setOption 方法
Connection 接口 [UltraLiteJ API], 141
setOutputBufferSize 方法
StreamHTTPParms 接口 [UltraLiteJ API], 207
setPageSize 方法
Configuration 接口 [UltraLiteJ API], 121
setPassword 方法
Configuration 接口 [UltraLiteJ API], 121
SyncParms 类 [UltraLiteJ API], 229
setPingOnly 方法
SyncParms 类 [UltraLiteJ API], 229
setPort 方法
StreamHTTPParms 接口 [UltraLiteJ API], 208
setPublications 方法
SyncParms 类 [UltraLiteJ API], 230
setRowMaximumThreshold 方法
ConfigPersistent 接口 [UltraLiteJ API], 116
setRowMinimumThreshold 方法
ConfigPersistent 接口 [UltraLiteJ API], 116
setSendColumnNames 方法
SyncParms 类 [UltraLiteJ API], 230
setShadowPaging 方法
ConfigPersistent 接口 [UltraLiteJ API], 117
setSyncObserver 方法
SyncParms 类 [UltraLiteJ API], 231
setTableOrder 方法
SyncParms 类 [UltraLiteJ API], 231
setTrustedCertificates 方法
StreamHTTPSParms 接口 [UltraLiteJ API], 212
setUploadOnly 方法
SyncParms 类 [UltraLiteJ API], 232
setURLSuffix 方法
StreamHTTPParms 接口 [UltraLiteJ API], 208
setUserName 方法
SyncParms 类 [UltraLiteJ API], 232
setVersion 方法
SyncParms 类 [UltraLiteJ API], 233
setWriteAtEnd 方法
ConfigPersistent 接口 [UltraLiteJ API], 117
set 方法
DecimalNumber 接口 [UltraLiteJ API], 155
SHORT 变量
Domain 接口 [UltraLiteJ API], 164
SISListener 接口（仅限 J2ME BlackBerry） [UltraLiteJ API]
description, 184
startListening 方法, 184
stopListening 方法, 184
SISRequestHandler 接口（仅限 J2ME BlackBerry） [UltraLiteJ API]
description, 185
onError 方法, 185
onRequest 方法, 185
SQL
术语定义, 302
SQL Anywhere
文档, viii

术语定义, 302
SQLCode 接口 [UltraLiteJ API]
description, 186
SQLE_AGGREGATES_NOT_ALLOWED 变量 [UltraLiteJ]
UltraLiteJ API, 189
SQLE_ALIAS_NOT_UNIQUE 变量 [UltraLiteJ]
UltraLiteJ API, 189
SQLE_ALIAS_NOT_YET_DEFINED 变量 [UltraLiteJ]
UltraLiteJ API, 189
SQLE_AUTHENTICATION_FAILED 变量 [UltraLiteJ]
UltraLiteJ API, 189
SQLE_CANNOT_EXECUTE_STMT 变量 [UltraLiteJ]
UltraLiteJ API, 189
SQLE_CLIENT_OUT_OF_MEMORY 变量 [UltraLiteJ]
UltraLiteJ API, 189
SQLE_COLUMN_AMBIGUOUS 变量 [UltraLiteJ]
UltraLiteJ API, 190
SQLE_COLUMN_CANNOT_BE_NULL 变量 [UltraLiteJ]
UltraLiteJ API, 190
SQLE_COLUMN_NOT_FOUND 变量 [UltraLiteJ]
UltraLiteJ API, 190
SQLE_COLUMN_NOT_STREAMABLE 变量 [UltraLiteJ]
UltraLiteJ API, 190
SQLE_COMMUNICATIONS_ERROR 变量 [UltraLiteJ]
UltraLiteJ API, 190
SQLE_CONFIG_IN_USE 变量 [UltraLiteJ]
UltraLiteJ API, 191
SQLE_CONVERSION_ERROR 变量 [UltraLiteJ]
UltraLiteJ API, 191
SQLE_CURSOR_ALREADY_OPEN 变量 [UltraLiteJ]
UltraLiteJ API, 191
SQLE_DATABASE_ACTIVE 变量 [UltraLiteJ]
UltraLiteJ API, 191
SQLE_DEVICE_IO_FAILED 变量 [UltraLiteJ]
UltraLiteJ API, 191
SQLE_DIV_ZERO_ERROR 变量 [UltraLiteJ]
UltraLiteJ API, 191
SQLE_DOWNLOAD_CONFLICT 变量 [UltraLiteJ]
UltraLiteJ API, 192
SQLE_ERROR 变量 [UltraLiteJ]
UltraLiteJ API, 192
SQLE_EXISTING_PRIMARY_KEY 变量 [UltraLiteJ]
UltraLiteJ API, 192
SQLE_EXPRESSION_ERROR 变量 [UltraLiteJ]
UltraLiteJ API, 192
SQLE_FILE_BAD_DB 变量 [UltraLiteJ]
UltraLiteJ API, 192
SQLE_FILE_WRONG_VERSION 变量 [UltraLiteJ]
UltraLiteJ API, 193
SQLE_FOREIGN_KEY_NAME_NOT_FOUND 变量 [UltraLiteJ]
UltraLiteJ API, 193
SQLE_IDENTIFIER_TOO_LONG 变量 [UltraLiteJ]
UltraLiteJ API, 193
SQLE_INCOMPLETE_SYNCHRONIZATION 变量 [UltraLiteJ]
UltraLiteJ API, 193
SQLE_INDEX_HAS_NO_COLUMNS 变量 [UltraLiteJ]
UltraLiteJ API, 193
SQLE_INDEX_NOT_FOUND 变量 [UltraLiteJ]
UltraLiteJ API, 193
SQLE_INDEX_NOT_UNIQUE 变量 [UltraLiteJ]
UltraLiteJ API, 194
SQLE_INTERRUPTED 变量 [UltraLiteJ]
UltraLiteJ API, 194
SQLE_INVALID_COMPARISON 变量 [UltraLiteJ]
UltraLiteJ API, 194
SQLE_INVALID_DISTINCT_AGGREGATE 变量 [UltraLiteJ]
UltraLiteJ API, 194
SQLE_INVALID_DOMAIN 变量 [UltraLiteJ]
UltraLiteJ API, 194
SQLE_INVALID_FOREIGN_KEY_DEF 变量 [UltraLiteJ]
UltraLiteJ API, 195
SQLE_INVALID_GROUP_SELECT 变量 [UltraLiteJ]
UltraLiteJ API, 195
SQLE_INVALID_INDEX_TYPE 变量 [UltraLiteJ]
UltraLiteJ API, 195
SQLE_INVALID_LOGON 变量 [UltraLiteJ]
UltraLiteJ API, 195

SQLE_INVALID_OPTION_SETTING 变量 [UltraLiteJ]
UltraLiteJ API, 195
SQLE_INVALID_OPTION 变量 [UltraLiteJ]
UltraLiteJ API, 195
SQLE_INVALID_ORDER 变量 [UltraLiteJ]
UltraLiteJ API, 196
SQLE_INVALID_PARAMETER 变量 [UltraLiteJ]
UltraLiteJ API, 196
SQLE_INVALID_UNION 变量 [UltraLiteJ]
UltraLiteJ API, 196
SQLE_LOCKED 变量 [UltraLiteJ]
UltraLiteJ API, 196
SQLE_MAX_ROW_SIZE_EXCEEDED 变量 [UltraLiteJ]
UltraLiteJ API, 196
SQLE_MUST_BE_ONLY_CONNECTION 变量 [UltraLiteJ]
UltraLiteJ API, 197
SQLE_NAME_NOT_UNIQUE 变量 [UltraLiteJ]
UltraLiteJ API, 197
SQLE_NO_COLUMN_NAME 变量 [UltraLiteJ]
UltraLiteJ API, 197
SQLE_NO_CURRENT_ROW 变量 [UltraLiteJ]
UltraLiteJ API, 197
SQLE_NO_MATCHING_SELECT_ITEM 变量 [UltraLiteJ]
UltraLiteJ API, 198
SQLE_NO_PRIMARY_KEY 变量 [UltraLiteJ]
UltraLiteJ API, 198
SQLE_NOERROR 变量 [UltraLiteJ]
UltraLiteJ API, 197
SQLE_NOT_IMPLEMENTED 变量 [UltraLiteJ]
UltraLiteJ API, 197
SQLE_OVERFLOW_ERROR 变量 [UltraLiteJ]
UltraLiteJ API, 198
SQLE_PAGE_SIZE_TOO_BIG 变量 [UltraLiteJ]
UltraLiteJ API, 198
SQLE_PAGE_SIZE_TOO_SMALL 变量 [UltraLiteJ]
UltraLiteJ API, 198
SQLE_PARAMETER_CANNOT_BE_NULL 变量 [UltraLiteJ]
UltraLiteJ API, 199
SQLE_PERMISSION_DENIED 变量 [UltraLiteJ]
UltraLiteJ API, 199
SQLE_PRIMARY_KEY_NOT_UNIQUE 变量 [UltraLiteJ]
UltraLiteJ API, 199
SQLE_PUBLICATION_NOT_FOUND 变量 [UltraLiteJ]
UltraLiteJ API, 199
SQLE_RESOURCE_GOVERNOR_EXCEEDED 变量 [UltraLiteJ]
UltraLiteJ API, 199
SQLE_ROW_LOCKED 变量 [UltraLiteJ]
UltraLiteJ API, 199
SQLE_ROW_UPDATED_SINCE_READ 变量 [UltraLiteJ]
UltraLiteJ API, 200
SQLE_SCHEMA_UPGRADE_NOT_ALLOWED 变量 [UltraLiteJ]
UltraLiteJ API, 200
SQLE_SERVER_SYNCHRONIZATION_ERROR 变量 [UltraLiteJ]
UltraLiteJ API, 200
SQLE_SUBQUERY_RESULT_NOT_UNIQUE 变量 [UltraLiteJ]
UltraLiteJ API, 200
SQLE_SUBQUERY_SELECT_LIST 变量 [UltraLiteJ]
UltraLiteJ API, 200
SQLE_SYNC_INFO_INVALID 变量 [UltraLiteJ]
UltraLiteJ API, 201
SQLE_SYNCHRONIZATION_IN_PROGRESS 变量 [UltraLiteJ]
UltraLiteJ API, 201
SQLE_SYNTAX_ERROR 变量 [UltraLiteJ]
UltraLiteJ API, 201
SQLE_TABLE_HAS_NO_COLUMNS 变量 [UltraLiteJ]
UltraLiteJ API, 201
SQLE_TABLE_IN_USE 变量 [UltraLiteJ]
UltraLiteJ API, 201
SQLE_TABLE_NOT_FOUND 变量 [UltraLiteJ]
UltraLiteJ API, 201
SQLE_TOO_MANY_PUBLICATIONS 变量 [UltraLiteJ]
UltraLiteJ API, 202
SQLE_ULTRALITE_DATABASE_NOT_FOUND 变量 [UltraLiteJ]
UltraLiteJ API, 202
SQLE_ULTRALITE_OBJ_CLOSED 变量 [UltraLiteJ]
UltraLiteJ API, 202

 Copyright © 2009, iAnywhere Solutions, Inc. - SQL Anywhere 11.0.1

SQLE_ULTRALITEJ_OPERATION_FAILED 变量 [UltraLiteJ]
UltraLiteJ API, 202
SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED 变量 [UltraLiteJ]
UltraLiteJ API, 202
SQLE_UNABLE_TO_CONNECT 变量 [UltraLiteJ]
UltraLiteJ API, 203
SQLE_UNCOMMITTED_TRANSACTIONS 变量 [UltraLiteJ]
UltraLiteJ API, 203
SQLE_UNDERFLOW 变量 [UltraLiteJ]
UltraLiteJ API, 203
SQLE_UNKNOWN_FUNC 变量 [UltraLiteJ]
UltraLiteJ API, 203
SQLE_UPLOAD_FAILED_AT_SERVER 变量 [UltraLiteJ]
UltraLiteJ API, 203
SQLE_VALUE_IS_NULL 变量 [UltraLiteJ]
UltraLiteJ API, 203
SQLE_VARIABLE_INVALID 变量 [UltraLiteJ]
UltraLiteJ API, 204
SQLE_WRONG_NUM_OF_INSERT_COLS 变量 [UltraLiteJ]
UltraLiteJ API, 204
SQLE_WRONG_PARAMETER_COUNT 变量 [UltraLiteJ]
UltraLiteJ API, 204
SQL Remote
术语定义, 302
SQL 语句
术语定义, 302
STARTING 变量
syncObserver.States 接口 [UltraLiteJ API], 217
startListening 方法
SISListener 接口（仅限 J2ME BlackBerry） [UltraLiteJ API], 184
startSynchronizationDelete 方法
Connection 接口 [UltraLiteJ API], 142
stopListening 方法
SISListener 接口（仅限 J2ME BlackBerry） [UltraLiteJ API], 184
stopSynchronizationDelete 方法
Connection 接口 [UltraLiteJ API], 142
StreamHTTPParms 接口 [UltraLiteJ API]
description, 205
getHost 方法, 205
getOutputBufferSize 方法, 206
getPort 方法, 206
getURLSuffix 方法, 206
setHost 方法, 207
setOutputBufferSize 方法, 207
setPort 方法, 208
setURLSuffix 方法, 208
StreamHTTPSParms 接口 [UltraLiteJ API]
description, 209
getCertificateCompany 方法, 210
getCertificateName 方法, 210
getCertificateUnit 方法, 211
getTrustedCertificates 方法, 211
setCertificateCompany 方法, 211
setCertificateName 方法, 211
setCertificateUnit 方法, 212
setTrustedCertificates 方法, 212
subtract 方法
DecimalNumber 接口 [UltraLiteJ API], 155
Sybase Central
术语定义, 303
SYNC_ALL_DB_PUB_NAME 变量
Connection 接口 [UltraLiteJ API], 127
SYNC_ALL_PUBS 变量
Connection 接口 [UltraLiteJ API], 127
SYNC_ALL 变量
Connection 接口 [UltraLiteJ API], 126
synchronize 方法
Connection 接口 [UltraLiteJ API], 142
syncObserver.States 接口 [UltraLiteJ API]
CHECKING_LAST_UPLOAD 变量, 215
COMMITTING_DOWNLOAD 变量, 215
DISCONNECTING 变量, 216
DONE 变量, 216
ERROR 变量, 216
FINISHING_UPLOAD 变量, 216
RECEIVING_TABLE 变量, 216
RECEIVING_UPLOAD_ACK 变量, 217
ROLLING_BACK_DOWNLOAD 变量, 217
SENDING_DOWNLOAD_ACK 变量, 217
SENDING_HEADER 变量, 217
SENDING_TABLE 变量, 217
STARTING 变量, 217
SyncObserver.States 接口 [UltraLiteJ API]
description, 215
syncObserver 接口 [UltraLiteJ API]
syncProgress 方法, 213

SyncObserver 接口 [UltraLiteJ API]
description, 213
SyncParms 方法
SyncParms 类 [UltraLiteJ API], 221
SyncParms 类 [UltraLiteJ API]
description, 219
getAcknowledgeDownload 方法, 221
getAuthenticationParms 方法, 221
getLivenessTimeout 方法, 222
getNewPassword 方法, 222
getPassword 方法, 222
getPublications 方法, 223
getSendColumnNames 方法, 223
getStreamParms 方法, 223
getSyncObserver 方法, 224
getSyncResult 方法, 224
getTableOrder 方法, 224
getUserName 方法, 225
getVersion 方法, 225
HTTP_STREAM 变量, 221
HTTPS_STREAM 变量, 220
isDownloadOnly 方法, 225
isPingOnly 方法, 226
isUploadOnly 方法, 226
setAcknowledgeDownload 方法, 226
setAuthenticationParms 方法, 227
setDownloadOnly 方法, 227
setLivenessTimeout 方法, 228
setNewPassword 方法, 228
setPassword 方法, 229
setPingOnly 方法, 229
setPublications 方法, 230
setSendColumnNames 方法, 230
setSyncObserver 方法, 231
setTableOrder 方法, 231
setUploadOnly 方法, 232
setUserName 方法, 232
setVersion 方法, 233
SyncParms 方法, 221
syncProgress 方法
syncObserver 接口 [UltraLiteJ API], 213
SyncResult.AuthStatusCode 接口 [UltraLiteJ API]
description, 238
EXPIRED 变量, 238
IN_USE 变量, 238
INVALID 变量, 238
UNKNOWN 变量, 239
VALID 变量, 239
VALID_BUT_EXPIRES_SOON 变量, 239
SyncResult 类 [UltraLiteJ API]
description, 234
getAuthStatus 方法, 234
getAuthValue 方法, 234
getCurrentTableName 方法, 235
getIgnoredRows 方法, 235
getReceivedByteCount 方法, 235
getReceivedRowCount 方法, 235
getSentByteCount 方法, 236
getSentRowCount 方法, 236
getStreamErrorCode 方法, 236
getStreamErrorMessage 方法, 236
getSyncedTableCount 方法, 237
getTotalTableCount 方法, 237
isUploadOK 方法, 237
SYS
术语定义, 303
SYS_ARTICLES 变量
TableSchema 接口 [UltraLiteJ API], 241
SYS_COLUMNS 变量
TableSchema 接口 [UltraLiteJ API], 241
SYS_FKEY_COLUMNS 变量
TableSchema 接口 [UltraLiteJ API], 241
SYS_FOREIGN_KEYS 变量
TableSchema 接口 [UltraLiteJ API], 242
SYS_INDEX_COLUMNS 变量
TableSchema 接口 [UltraLiteJ API], 242
SYS_INDEXES 变量
TableSchema 接口 [UltraLiteJ API], 242
SYS_INTERNAL 变量
TableSchema 接口 [UltraLiteJ API], 242
SYS_PRIMARY_INDEX 变量
TableSchema 接口 [UltraLiteJ API], 242
SYS_PUBLICATIONS 变量
TableSchema 接口 [UltraLiteJ API], 243
SYS_TABLES 变量
TableSchema 接口 [UltraLiteJ API], 243
sysarticles 系统表 [UltraLiteJ]
关于, 272
syscolumn 系统表 [UltraLiteJ]
关于, 267
sysfkcol 系统表 [UltraLiteJ]
关于, 274
sysforeignkey 系统表 [UltraLiteJ]
关于, 273

sysindexcolumn 系统表 [UltraLiteJ]
关于, 269
sysindex 系统表 [UltraLiteJ]
关于, 268
sysinternal 系统表 [UltraLiteJ]
关于, 270
syspublications 系统表 [UltraLiteJ]
关于, 271
systable 系统表 [UltraLiteJ]
关于, 266
散列
术语定义, 298
上载
术语定义, 299
设备跟踪
术语定义, 299
生成的连接条件
术语定义, 300
实例化视图
术语定义, 299
实用程序
UltraLiteJ 数据库信息 [ULjInfo], 276
UltraLiteJ 数据库卸载 [ULjUnload], 278
UltraLiteJ 数据库装载 [ULjLoad], 277
示例代码
CreateDb, 32
CreateSales, 37
DumpSchema, 51
encrypted, 47
LoadDb, 33
obfuscate, 43
ReadInnerJoin, 36
ReadSeq, 35
Reorg, 41
SalesReport, 40
SortTransactions, 41
UltraLiteJ, 31
同步, 25
世代号
术语定义, 299
事件模型
术语定义, 299
事务
UltraLiteJ 管理, 22
术语定义, 299
事务处理
UltraLiteJ 管理, 22
事务日志
术语定义, 299
事务日志镜像
术语定义, 300
事务完整性
术语定义, 300
视图
术语定义, 299
授权选项
术语定义, 300
受保护的功能
术语定义, 300
术语表
SQL Anywhere 术语列表, 285
数据操作
UltraLiteJ，使用 SQL, 18
数据操作语言
术语定义, 300
数据库
术语定义, 301
数据库对象
术语定义, 301
数据库服务器
术语定义, 301
数据库管理员
术语定义, 301
数据库连接
术语定义, 301
数据库名称
术语定义, 301
数据库所有者
术语定义, 302
数据库文件
术语定义, 302
数据类型
术语定义, 300
数据立方体
术语定义, 301
死锁
术语定义, 302
索引
UltraLiteJ sysindex 系统表, 268
UltraLiteJ sysindexcolumn 系统表, 269
术语定义, 303
锁定
术语定义, 302

T

TABLE_IS_NOSYNC 变量
TableSchema 接口 [UltraLiteJ API], 243
TABLE_IS_SYSTEM 变量
TableSchema 接口 [UltraLiteJ API], 243
TableSchema 接口
UltraLiteJ, 16
TableSchema 接口 [UltraLiteJ API]
createColumn(String, short) 方法, 243
createColumn(String, short, int) 方法, 244
createColumn(String, short, int, int) 方法, 244
createIndex 方法, 245
createPrimaryIndex 方法, 245
createUniqueIndex 方法, 246
createUniqueKey 方法, 246
description, 240
setNoSync 方法, 247
SYS_ARTICLES 变量, 241
SYS_COLUMNS 变量, 241
SYS_FKEY_COLUMNS 变量, 241
SYS_FOREIGN_KEYS 变量, 242
SYS_INDEX_COLUMNS 变量, 242
SYS_INDEXES 变量, 242
SYS_INTERNAL 变量, 242
SYS_PRIMARY_INDEX 变量, 242
SYS_PUBLICATIONS 变量, 243
SYS_TABLES 变量, 243
TABLE_IS_NOSYNC 变量, 243
TABLE_IS_SYSTEM 变量, 243
TIMESTAMP 变量
Domain 接口 [UltraLiteJ API], 164
TIME 变量
Domain 接口 [UltraLiteJ API], 164
TINY 变量
Domain 接口 [UltraLiteJ API], 164
truncateTable 方法
Connection 接口 [UltraLiteJ API], 142
提交
UltraLiteJ 事务, 22
通告程序
术语定义, 303
通信流
术语定义, 303
同步
UltraLiteJ, 25
术语定义, 303
添加到 BlackBerry 应用程序, 76
统一数据库
术语定义, 303
图标
此帮助文档中使用的, xi
推式请求
术语定义, 304
推式通知
术语定义, 304

U

UINT16_MAX 变量
Domain 接口 [UltraLiteJ API], 165
ULjDbT
UltraLiteJ 实用程序, 280
ULjException 类 [UltraLiteJ API]
description, 248
getCausingException 方法, 251
getErrorCode 方法, 251
getSqlOffset 方法, 251
ULjInfo 实用程序
语法, 276
ULjLoad 实用程序
语法, 277
ULjUnload 实用程序
语法, 278
UltraLite
术语定义, 304
UltraLiteJ
API, 90
BlackBerry CustDB 教程, 26
BlackBerry 应用程序教程, 65
HTTP 和 HTTPS 通信, 5
ULjDbT 实用程序, 280
ULjInfo 实用程序, 276
ULjLoad 实用程序, 277
ULjUnload 实用程序, 278
UltraLiteJ 数据库传输实用程序, 280
事务, 5
事务处理, 22
使用 SQL 进行数据操作, 18
关于, 4
内置更改跟踪, 5
创建数据库, 69
功能, 5
功能限制, 7
加密, 5, 23

同步, 10, 25
同步发布, 5
字符集和归类, 5
实用程序 (J2ME), 280
实用程序 (J2SE), 276
并发同步, 10
并发和锁定, 5
开发应用程序, 11
支持的 SQL 语句, 18
数据库存储区, 5, 8, 13
数据操作, 19
数据检索, 21
检查点和恢复, 5
示例代码, 31
系统表模式, 51
部署, 30
高速缓存管理, 5
UltraLiteJ API
CollectionOfValueReaders 接口, 91
CollectionOfValueWriters 接口, 98
ColumnSchema 接口, 104
ConfigFile 接口, 109
ConfigNonPersistent 接口, 110
ConfigObjectStore 接口（仅限 J2ME BlackBerry）, 111
ConfigPersistent 接口, 112
ConfigRecordStore 接口（仅限 J2ME）, 119
Configuration 接口, 120
Connection 接口, 122
DatabaseInfo 接口, 144
DatabaseManager 类, 147
DecimalNumber 接口, 153
description, 90
Domain 接口, 156
EncryptionControl 接口, 169
ForeignKeySchema 接口, 171
IndexSchema 接口, 173
PreparedStatement 接口, 176
ResultSet 接口, 180
ResultSetMetadata 接口, 183
SISListener 接口（仅限 J2ME BlackBerry）, 184
SISRequestHandler 接口（仅限 J2ME BlackBerry）, 185
SQLCode 接口, 186
StreamHTTPParms 接口, 205
StreamHTTPSParms 接口, 209
SyncObserver 接口, 213
SyncObserver.States 接口, 215
SyncParms 类, 219
SyncResult 类, 234
SyncResult.AuthStatusCode 接口, 238
TableSchema 接口, 240
ULjException 类, 248
Value 接口, 252
ValueReader 接口, 256
ValueWriter 接口, 260
UltraLiteJ 数据库
存储发布, 271
存储索引, 268
描述发布, 272
描述外键, 273, 274
描述表, 266
描述表的列, 269
UltraLiteJ 数据库传输实用程序
BlackBerry, 280
UltraLiteJ 数据库卸载实用程序
语法, 278
UltraLiteJ 数据库信息实用程序
语法, 276
UltraLiteJ 数据库装载实用程序
语法, 277
UltraLite 数据库
在 UltraLiteJ 中连接, 13
UltraLite 运行时
术语定义, 304
UNIQUE_INDEX 变量
IndexSchema 接口 [UltraLiteJ API], 174
UNIQUE_KEY 变量
IndexSchema 接口 [UltraLiteJ API], 174
UNKNOWN 变量
SyncResult.AuthStatusCode 接口 [UltraLiteJ API], 239
UNSIGNED_BIG 变量
Domain 接口 [UltraLiteJ API], 165
UNSIGNED_INTEGER 变量
Domain 接口 [UltraLiteJ API], 165
UNSIGNED_SHORT 变量
Domain 接口 [UltraLiteJ API], 165
UUID 变量
Domain 接口 [UltraLiteJ API], 166

V

VALID_BUT_EXPIRES_SOON 变量

SyncResult.AuthStatusCode 接口 [UltraLiteJ API], 239
VALID 变量
SyncResult.AuthStatusCode 接口 [UltraLiteJ API], 239
ValueReader 接口 [UltraLiteJ API]
description, 256
getBlobInputStream 方法, 256
getBoolean 方法, 256
getBytes 方法, 257
getClobReader 方法, 257
getDate 方法, 257
getDecimalNumber 方法, 257
getDouble 方法, 258
getFloat 方法, 258
getInt 方法, 258
getLong 方法, 258
getString 方法, 259
getValue 方法, 259
isNull 方法, 259
ValueWriter 接口 [UltraLiteJ API]
description, 260
getBlobOutputStream 方法, 260
getClobWriter 方法, 260
set(boolean) 方法, 261
set(byte[]) 方法, 263
set(Date) 方法, 261
set(DecimalNumber) 方法, 261
set(double) 方法, 262
set(float) 方法, 262
set(int) 方法, 261
set(long) 方法, 262
set(String) 方法, 263
set(Value) 方法, 263
setNull 方法, 263
Value 接口 [UltraLiteJ API]
compareValue 方法, 253
description, 252
duplicate 方法, 254
getDomain 方法, 254
getSize 方法, 254
getType 方法, 255
release 方法, 255
VARCHAR_DEFAULT 变量
Domain 接口 [UltraLiteJ API], 166
VARCHAR_MIN 变量
Domain 接口 [UltraLiteJ API], 166
VARCHAR 变量
Domain 接口 [UltraLiteJ API], 166

W

Windows
术语定义, 306
Windows Mobile
术语定义, 306
writeAtEnd 方法
ConfigPersistent 接口 [UltraLiteJ API], 118
外表
术语定义, 304
外部登录
术语定义, 304
外键
UltraLiteJ 系统表, 273, 274
术语定义, 304
外键约束
术语定义, 305
外连接
术语定义, 305
完全备份
术语定义, 305
完整性
术语定义, 305
网关
术语定义, 305
网络服务器
术语定义, 305
网络协议
术语定义, 306
唯一约束
术语定义, 306
维护版本
术语定义, 306
位数组
术语定义, 306
谓语
术语定义, 306
文档
SQL Anywhere, viii
约定, ix
文件定义数据库
术语定义, 306
物理索引
术语定义, 307

X

系统表
- UltraLiteJ, 51
- UltraLiteJ sysarticles, 272
- UltraLiteJ syscolumn, 267
- UltraLiteJ sysfkcol, 274
- UltraLiteJ sysforeignkey, 273
- UltraLiteJ sysindex, 268
- UltraLiteJ sysindexcolumn, 269
- UltraLiteJ sysinternal, 270
- UltraLiteJ syspublications, 271
- UltraLiteJ systable, 266
- 术语定义, 307

系统对象
- 术语定义, 307

系统视图
- 术语定义, 307

下载
- 术语定义, 307

相关名
- 术语定义, 307

项目
- 术语定义, 307

消息存储库
- 术语定义, 307

消息类型
- 术语定义, 307

消息日志
- 术语定义, 308

消息系统
- 术语定义, 308

卸载
- 术语定义, 308

新闻组
- 技术支持, xiii

行级触发器
- 术语定义, 291

性能统计
- 术语定义, 308

选择
- UltraLiteJ 行, 21

Y

业务规则
- 术语定义, 308

疑难解答
- 新闻组, xiii

引用对象
- 术语定义, 308

应用程序
- UltraLiteJ 开发, 11
- 部署 BlackBerry, 74

用户定义数据类型
- 术语定义, 309

游标
- 术语定义, 309

游标结果集
- 术语定义, 309

游标位置
- 术语定义, 309

语句级触发器
- 术语定义, 309

域
- 术语定义, 309

预订
- 术语定义, 310

预准备语句
- UltraLiteJ, 19

元数据
- 术语定义, 310

原子事务
- 术语定义, 310

远程 ID
- 术语定义, 310

远程数据库
- 术语定义, 310

约定
- 命令 shell, xi
- 命令提示符, xi
- 文档, ix
- 文档中的文件名, x

约束
- 术语定义, 311

运营公司
- 术语定义, 311

Z

增量备份
- 术语定义, 311

争用
- 术语定义, 311

正则表达式
- 术语定义, 311

支持

新闻组, xiii
直方图
术语定义, 311
直接行处理
术语定义, 311
智能手机
BlackBerry 实用程序 (J2ME), 280
主表
术语定义, 312
主键
术语定义, 312
主键约束
术语定义, 312
主题
图标, xi
子查询
术语定义, 312
自然连接
术语定义, 300
字符串
术语定义, 312
字符集
术语定义, 312