SYBASE iAnywhere

MobiLink
入门

2009 年 2 月

11.0.1 版

版权和商标

目录

关于本手册

主题

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。

读者

本手册适用于那些想要在其信息系统中添加同步的 SQL Anywhere 及其它关系数据库系统的用户。

预备知识

有关 MobiLink 与其它同步和复制技术的比较，请参见“数据交换技术概述” 《SQL Anywhere 11 - 简介》。

关于 SQL Anywhere 文档

完整的 SQL Anywhere 文档以四种形式提供，但所包含信息均相同。

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

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

  管理工具使用同一联机文档来实现帮助功能。

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

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

  使用 DocCommentXchange 可以执行以下任务：

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

  访问 http://dcx.sybase.com。

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

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

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

关于文档集中的手册

SQL Anywhere 文档由以下手册组成：

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

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

文档约定

本节列出了本文档中使用的约定。

操作系统

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

为了简化对操作系统的提及，本文档按如下方式对支持的操作系统进行分组：

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

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

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

目录和文件名

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

下面是文档编写中用于简化目录和文件名的约定：

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

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

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

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

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

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

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

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

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

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

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

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

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

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

命令提示符和命令 shell 语法

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

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

下面是命令行语法的一些示例，不同的平台可能会有不同的形式：

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

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

  如果括号导致出现语法问题，用大括号替代：

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

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

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

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

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

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

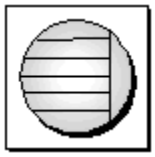
- **环境变量** 本文档介绍设置环境变量。在 Windows shell 中，环境变量使用语法 *%ENVVAR%* 来指定。在 Unix shell 中，环境变量使用语法 *$ENVVAR* 或 *${ENVVAR}* 来指定。

图标

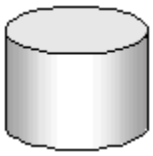
本文档中使用了下列图标。

- 客户端应用程序。

- 数据库服务器，如 Sybase SQL Anywhere。



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



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

- 编程接口。

联系文档小组

我们欢迎您就本帮助文档提出意见、建议和反馈信息。

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

DocCommentXchange

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

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

访问 http://dcx.sybase.com。

查找详细信息并请求技术支持

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

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

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

新闻组位于 *forums.sybase.com* 新闻服务器上。

这些新闻组包括：

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

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

新闻组免责声明

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

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

MobiLink 技术简介

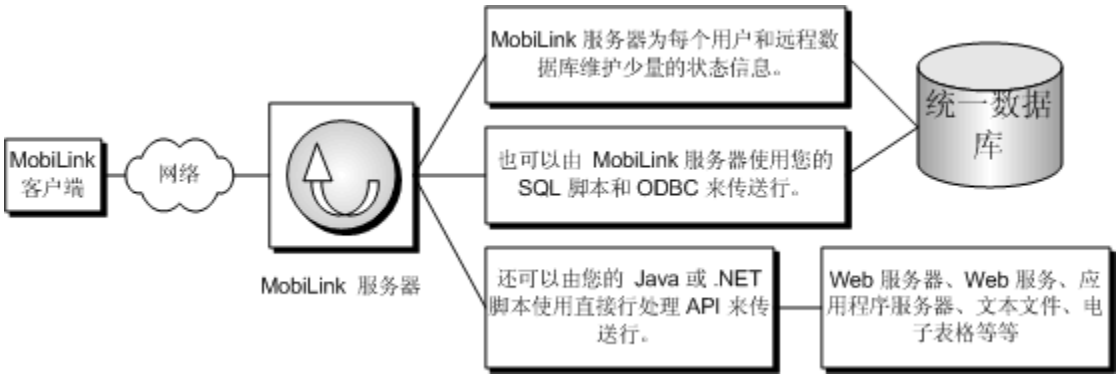
本节介绍 MobiLink 同步技术并说明如何利用该技术在两个或多个数据库之间复制数据。

了解 MobiLink 同步

目录

MobiLink 应用程序的组成部分

在 MobiLink 同步中，许多客户端通过 MobiLink 服务器与中央数据源同步。



- **MobiLink 客户端** 可将客户端安装在手持式设备（如 Palm Pilot 或者 Windows Mobile 设备）、服务器、台式计算机或者智能电话上。共支持两种类型的客户端：UltraLite 和 SQL Anywhere 数据库。在 MobiLink 安装中可以同时使用上述两种或其中任一种客户端。请参见“MobiLink 客户端” 《MobiLink - 客户端管理》。
- **网络** MobiLink 服务器和 MobiLink 客户端之间的连接可使用多种协议。请参见：
  - MobiLink 服务器： “-x 选项”一节 《MobiLink - 服务器管理》
  - UltraLite 和 SQL Anywhere 客户端： “MobiLink 客户端网络协议选项” 《MobiLink - 客户端管理》
- **MobiLink 服务器** 此服务器管理同步过程并提供所有 MobiLink 客户端与统一数据库服务器之间的接口。请参见“MobiLink 服务器” 《MobiLink - 服务器管理》。
- **统一数据库** 通常，此数据库包含 MobiLink 同步所需的系统表和过程，以及同步所需的状态信息。此数据库通常还包括同步系统中信息的集中副本。请参见“MobiLink 统一数据库” 《MobiLink - 服务器管理》。
- **状态信息** MobiLink 服务器通常维护统一数据库中系统表内的同步信息。这通过 ODBC 连接来实现。

  还可以选择在单独数据库中存储状态信息。请参见“MobiLink 系统数据库”一节 《MobiLink - 服务器管理》。
- **SQL 行处理** 如果为 MobiLink 服务器提供 SQL 脚本，则它将使用这些脚本，并通过 ODBC 连接在 MobiLink 服务器和统一数据库之间传输行。请参见“用于编写服务器端同步逻辑的选项”一节第 14 页。
- **直接行处理** 除了统一数据库，还可以选择使用 MobiLink 直接行处理与其它数据源进行同步。请参见“直接行处理” 《MobiLink - 服务器管理》。

- **同步脚本** 可以为远程数据库中的每个表编写同步脚本，并将这些脚本保存在统一数据库的 MobiLink 系统表内。这些脚本确定如何处理上载数据以及下载哪些数据。有两类脚本：表脚本和连接级脚本。请参见：
  - “MobiLink 事件概述”一节 《MobiLink - 服务器管理》
  - “编写同步脚本” 《MobiLink - 服务器管理》
  - “同步事件” 《MobiLink - 服务器管理》
  - “用于编写服务器端同步逻辑的选项”一节第 14 页

MobiLink 特色

MobiLink 同步具有适应性和灵活性。以下是它的一些主要特点：

特点

- **易于入门** 通过使用 [**创建同步模型向导**]，可以快速创建同步应用程序。该向导可以处理复杂同步系统的许多困难的实现细节。Sybase Central [模型] 模式用于脱机查看同步模型，它提供了用于进行更改的简易界面，并提供了用于将模型部署到统一数据库的部署选项。
- **监控和报告** MobiLink 提供了两种同步监控机制：MobiLink 监控器和统计脚本。
- **性能调优** 有多种机制可用于调整 MobiLink 性能。例如，您可以调整争用程度、上载高速缓存大小、数据库连接数、记录详细程度或 BLOB 高速缓存大小。
- **可伸缩性** MobiLink 是一个极具可伸缩性和稳定性的同步平台。一个 MobiLink 服务器便可以处理数千个并发同步，而通过使用负载平衡可以同时运行多个 MobiLink 服务器。MobiLink 服务器是多线程的，并将连接池用于统一数据库。
- **安全** MobiLink 提供了大量安全性选项，其中包括可与现有验证集成的用户验证、加密以及通过交换安全证书发挥作用的传送层安全性。MobiLink 还提供了经 FIPS 认证的安全性选项。

体系结构

- **数据协调** MobiLink 允许选择数据的特定部分进行同步。MobiLink 同步还允许您解决在不同数据库中所做更改之间的冲突。同步过程由同步逻辑控制，可以将该逻辑编写为 SQL、Java 或 .NET 应用程序。同步逻辑的每个部分称为一个**脚本**。例如，您可以使用脚本指定如何将上载的数据应用到统一数据库、指定下载的内容以及处理统一数据库与远程数据库之间的不同模式和名称。基于事件的脚本编写方式为同步过程的设计提供了极大的灵活性，可以设计出冲突解决、错误报告和用户验证等功能。
- **双向同步** 可以在任何位置对数据库进行更改。
- **仅上载同步或仅下载同步** 可以选择仅执行上载或仅执行下载，也可以选择执行双向同步。
- **基于文件的下载** 可以文件形式分发下载，从而实现同步更改的脱机分发。此功能包括用于确保应用了正确数据的功能。
- **服务器启动的同步** 可以从统一数据库启动 MobiLink 同步。这意味着您可以将数据更新推送到远程数据库，并让远程数据库将数据上载到统一数据库中。请参见 MobiLink - 服务器启动的同步。
- **选择网络协议** 可通过 TCP/IP、HTTP 或 HTTPS 进行同步。Palm 设备可以通过 HotSync 进行同步。Windows Mobile 设备可以使用 ActiveSync 进行同步。
- **基于会话** 所有更改都可以通过单个事务进行上载和下载。每次同步成功完成后，统一数据库和远程数据库就会变得一致。（如果希望保留事务的顺序，还可以选择将远程数据库上的每个事务作为单独的事务上载。）

  要么同步整个事务，要么不同步事务的任何部分。这样可以确保每个数据库的事务完整性。
- **数据一致性** MobiLink 使用松散一致性策略来运行。每过一段时间，所有更改都会以一致的方式与每个站点同步，但在不同时刻，不同站点可能拥有不同的数据副本。

- **多种多样的硬件和软件平台** 多种广泛使用的数据库管理系统都可用作 MobiLink 统一数据库，也可以使用 MobiLink 服务器 API 定义与任意数据源同步。远程数据库可以是 SQL Anywhere 或 UltraLite。MobiLink 服务器在 Windows、Unix、Linux 和 Mac OS X 上运行。SQL Anywhere 在 Windows、Windows Mobile 或 Unix、Linux 和 Mac OS X 上运行。UltraLite 在 Palm 或 Windows Mobile 上运行。请参见“支持的平台”一节 《SQL Anywhere 11 - 简介》。

MobiLink 快速入门

MobiLink 专用于在间歇性地与一个或多个中央数据源连接的许多远程应用程序之间同步数据。在基本 MobiLink 应用程序中，远程客户端是 SQL Anywhere 或 UltraLite 数据库，而中央数据源是受支持的 ODBC 兼容关系数据库之一。此体系结构可以使用 MobiLink 服务器 API 进行扩展，这样，对于在服务器端要与哪个目标同步实际上没有任何限制。

在所有 MobiLink 应用程序中，MobiLink 服务器是同步过程的关键。同步通常是在 MobiLink 远程站点连接到 MobiLink 服务器时开始的。同步期间，远程站点的 MobiLink 客户端可以上载自上一次同步以来对远程数据库所做的更改。MobiLink 服务器收到这些数据时即会更新统一数据库，然后便可将更改从统一数据库下载到远程数据库。

开始开发 MobiLink 应用程序的最快速方式是使用 [**创建同步模型向导**]。使用该向导时，需要处理下述大部分步骤。请参见“MobiLink 模型简介”一节第 24 页。

但是，即使使用 MobiLink 模型，也需要了解 MobiLink 同步的过程和组成部分。

MobiLink 应用程序概述

♦ 创建 MobiLink 应用程序

1. 建立统一数据库。
   - 针对数据库运行安装脚本以添加 MobiLink 同步所需的系统对象。或者，可以创建单独的系统数据库来保存这些对象。

     请参见“MobiLink 统一数据库” 《MobiLink - 服务器管理》。
2. 建立远程数据库。
   - 远程数据库可以是 SQL Anywhere、UltraLite 或两者的组合。
   - 在远程数据库中，创建 MobiLink 用户。请参见“MobiLink 用户” 《MobiLink - 客户端管理》。
   - 要确定 SQL Anywhere 远程数据库中的上载项，请创建发布和预订。请参见“发布数据”一节 《MobiLink - 客户端管理》。

     要确定 UltraLite 远程数据库中的上载项，请创建发布。请参见“UltraLite 中的发布”一节 《UltraLite - 数据库管理和参考》。
3. 要确定如何应用上载，请创建服务器同步逻辑。请参见“同步脚本介绍”一节 《MobiLink - 服务器管理》。
4. 要下载自上次下载后发生更改的数据，请设置基于时间戳的同步。请参见“基于时间戳的下载”一节 《MobiLink - 服务器管理》。
5. 启动 MobiLink 服务器。请参见“MobiLink 服务器”一节 《MobiLink - 服务器管理》。
6. 在客户端启动同步。
   - 有关 SQL Anywhere 远程数据库的信息，请参见“启动同步”一节 《MobiLink - 客户端管理》。
   - 有关 UltraLite 远程数据库的信息，请参见“设计 UltraLite 中的同步”一节 《UltraLite - 数据库管理和参考》。

入门阅读

- “了解 MobiLink 同步”第 3 页
- “同步技术” 《MobiLink - 服务器管理》

教程

- “教程：MobiLink 简介”第 81 页
- “研究 MobiLink 的 CustDB 示例”第 45 页
- “UltraLite CustDB 示例” 《UltraLite - 数据库管理和参考》
- “研究 MobiLink Contact 示例”第 65 页
- “教程：将 MobiLink 用于 Oracle 10g 统一数据库”第 105 页
- “教程：将 MobiLink 与 Adaptive Server Enterprise 统一数据库结合使用”第 125 页
- “教程：使用 Java 同步逻辑”第 143 页
- “教程：使用 .NET 同步逻辑”第 155 页
- “教程：使用 .NET 和 Java 进行自定义验证”第 167 页
- “教程：直接行处理简介”第 177 页
- “教程：与 Microsoft Excel 同步”第 203 页
- “教程：与 XML 同步”第 223 页

其它入门资源

- MobiLink 提供了许多示例，可用来分析和运行，以探究 MobiLink 的功能。MobiLink 示例随产品安装在 *samples-dir\MobiLink* 中。（有关操作系统上 *samples-dir* 位置的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）
- MobiLink 代码交换示例位于 http://www.sybase.com/detail?id=1058600#319。

设计 MobiLink 应用程序

数据库应用程序有两种基本体系结构：

- **联机应用程序** 用户通过直接连接中央数据库更新数据。连接中断时，用户无法工作。
- **间歇性连接的智能客户端应用程序** 每个用户都有一个本地数据库。无论连接与否，这些数据库应用程序都可供用户使用，并与系统中的其它数据库保持同步。

MobiLink 专用于创建间歇性连接的智能客户端应用程序。智能客户端应用程序可显著提高应用程序的可用性、效率和可伸缩性，但却给应用程序开发人员带来了新的问题。本节介绍智能客户端应用程序开发人员面临的一些主要问题，并介绍如何在 MobiLink 同步环境下实施解决方案。

只同步所需内容

在大部分应用程序中，如果每次要更新远程设备上的部分数据时都需要下载整个统一数据库，这简直就是一场灾难。时间和带宽都将成为制约因素，从而使整个系统难以运转。可采用许多技术来确保只上载和下载用户所需的数据。

首先，每个远程数据库只应包含统一数据库中表和列的一个子集。例如，区域 A 的某个销售人员可能需要区域 B 中某个销售人员或主管的不同表和列。

在放置在远程设备上的表和列中，您只希望将需要同步的内容进行同步。在 MobiLink 应用程序中，只要数据类型匹配，便可在表和列间建立映射，而不必考虑它们的名称。缺省情况下，数据既可以上载也可以下载，但是 MobiLink 还允许指定某些列只能上载或只能下载。

同步应只将与用户相关的行下载到远程数据库。您可能想按远程数据库、用户或其它条件将下载内容进行划分。例如，区域 A 的销售代表可能只需要区域 A 的数据更新内容。

您只想更新发生更改的数据。在 MobiLink 应用程序中，上载是基于事务日志的，因此缺省情况下，只有数据在远程数据库上发生了更改才会上载数据。要针对下载执行相同的操作，可以指定基于时间戳的同步，以便系统能够记录数据成功下载的时间，仅当从该时间开始数据发生了更改才会被下载。

您可能还想实现高优先级同步的系统：对时效性要求很高的数据安排频繁更新，但对时效性要求较低的数据安排在晚上或将设备放到底座中时更新。可以通过创建安排在不同时间运行的不同发布实现高优先级同步。

除此之外，用户还会从同步推送系统获益，在该系统中，将根据需要将数据有效地推送到远程设备。例如，如果货运公司的调度员获悉某处交通中断，他们可以下载一个更新给即将驶入该区域的卡车司机。在 MobiLink 中，这称为服务器启动的同步。

处理上载冲突

假设您有一个仓库。每个雇员有一台手持式设备，用来在加入或移走箱子时更新库存。每个班次以 100 箱开始，因此每个雇员的远程数据库将注册 100，统一数据库也如此。David 移走 20 个箱子。他更新其数据库并进行同步。目前，他的数据库和统一数据库都记录 80。与此同时，Susan 移走十个箱子。但是在 Susan 更新其数据库并进行同步时，她的应用程序认为统一数据库还有 100 个箱子而不是 80 个。这便产生了上载冲突。

在此仓库应用程序中，解决方法是创建冲突解决逻辑，该逻辑假定正确值为 David 更新后的值减去原始值与 Susan 的值的差：


```
80 - (100 - 90) = 70
```

虽然此冲突解决逻辑适用于基于库存的应用程序（如仓库），但并不适用于所有业务应用程序。通过 MobiLink，可以定义涵盖以下方面的冲突解决逻辑：

- **库存模型** 更新正确单元数的行。
- **日期** 最近更新的时间（基于数据库中值更改的时间，而不是值同步的时间）。
- **人员** 例如，经理或记录的拥有者始终优先。
- **自定义** 仅涉及您需要执行的任何其它业务逻辑。

某些情况下，可以自行设计系统以避免发生上载冲突。如果在远程对数据进行了分区因而没有重叠，可能会避免冲突。但是，如果冲突可能发生，则应创建一个编程式解决方案来检测和解决冲突。

唯一主键

要上载数据、检测上载冲突和同步统一数据库中删除的行，在数据库系统的每个同步表中都必须具有唯一主键。每个行具有的主键不仅在数据库内必须唯一，在整个数据库系统内也必须唯一。不得更新主键。

MobiLink 提供了多种方式来保证唯一主键。一种方式是将主键的数据类型设置为 GUID。GUID 即是全局唯一标识符，是 16 字节长的十六进制数。MobiLink 提供了 NEWID 功能，可以为每个新行自动创建 GUID。

另一种解决方案是组合键。在 MobiLink 中，每个远程数据库都有一个称为远程 ID 的唯一值。主键可由远程 ID 与常规主键（如序数值）构成。

SQL Anywhere 还提供了全局自动增量解决方案。可声明将一个列作为全局自动增量，这样，当添加一行后，可通过将最后一个值增加来自动创建主键。此解决方案最适合统一数据库为 SQL Anywhere 的情形。

最后，可以创建分发到远程数据库的主键值池。

与开发同步解决方案过程中的许多决定一样，在选择使用哪个主键系统时，必须结合您对统一数据库和远程数据具有的控制级别。通常，远程数据库必须能够在没有管理的情况下运行。您可能还会发现很难在统一数据库上更改模式。此外，选择 RDBMS 作为统一数据库可能会限制您的选择，因为并非所有 RDBMS 都支持全部功能。

处理删除

同步系统中的另一个问题是如何处理从统一数据库中删除的行。假设我从统一数据库中删除了一行。下次 David 同步其远程数据库时，会下载删除内容—从 David 的数据库中删除该行。但是我在统一数据库中该怎么处理这一行呢？我不能删除这一行，因为我还需要将它下载给 Susan。

可通过以下两种方法处理下载删除：首先，可以为每个表添加一个状态列，指示该行是否删除。这种情况下，行绝对不会被删除—只是标记为要删除。可以偶尔对标记为删除的行进行清理，只要您确信所有远程数据库都是最新的。或者，可以为每个表创建一个影子表。影子表用于存储已删除行的主键值。删除一行后，触发器就会填充影子表，影子表中的值决定在远程数据库中删除的内容。

事务

在同步的数据库系统中，只应同步已提交的数据库事务。此外，所有涉及将要同步的数据的已提交事务也要被同步，否则应生成错误。这是 MobiLink 中的缺省行为。

您还必须考虑与统一数据库的连接的隔离级别。在确保数据一致性的前提下，需使用能提供最佳性能的隔离级别。隔离级别 0 (READ UNCOMMITTED) 通常不适用于同步，因为它会导致数据不一致。

缺省情况下，MobiLink 上载时使用隔离级别 SQL_TXN_READ_COMMITTED，如果可能，将在下载时使用快照隔离（否则，使用 SQL_TXN_READ_COMMITTED）。快照隔离可消除事务在统一数据库上关闭前的下载阻塞问题，但并非所有 RDBMS 都支持这一特性。

夏令时

每年一次的夏令时使时间更改过程中的数据库同步产生了一个问题。在秋季，时间退回一个小时，即 2:00 AM 变为 1:00 AM。如果您试图在 1:00 AM 和 2:00 AM 之间进行同步，则同步时间戳就会不明确：同步时间是第一个 1:15 AM 还是第二个 1:15 AM?

要解决此问题，在秋季时间变更时您可以关闭一个小时，或者您可以将统一数据库服务器调整为协调通用时间 (UTC)。

进一步阅读

- “同步技术” 《MobiLink - 服务器管理》

用于开发 MobiLink 应用程序的选项

MobiLink 提供了各种开发应用程序的方法。可单独使用这些方法，也可以组合使用。

- **创建同步模型向导** 此向导将指导您开发一个应用程序。先从具有模式的中央数据库开始，然后创建远程数据库和同步所需的脚本。该向导还可以在统一数据库上创建影子表来处理下载删除等事项。该向导完成后，您的同步模型会在 [模型] 模式中出现，在其中可以进一步对其进行自定义。提供了一个 [**部署同步模型向导**]，可用于创建数据库和表、更新 MobiLink 系统表以及创建运行 MobiLink 实用程序的脚本。

  部署 MobiLink 模型后，如果进一步对其进行了自定义，则可以使用下述方法之一在 [模型] 模式中或在模型外部进行更改。如果在模型之外进行更改并重新部署模型，则将丢失更改。

  请参见“MobiLink 模型”第 23 页。

- **Sybase Central 管理模式** Sybase Central 的 MobiLink 插件包括一个称为 [管理] 模式的部分，可在其中更新 MobiLink 应用程序的所有元素。

  请参见“模型模式”一节第 30 页。

- **系统过程** 建立中央数据库并将其当作统一数据库操作时，即会创建 MobiLink 同步所用的系统对象。这些对象包括 MobiLink 系统表，其中主要存储 MobiLink 应用程序的服务器端。还包括系统过程和实用程序，可用来将 MobiLink 脚本插入到 MobiLink 系统表、注册远程用户等等。请参见：

  - “MobiLink 服务器系统过程” 《MobiLink - 服务器管理》
  - “MobiLink 实用程序” 《MobiLink - 服务器管理》

- **直接操作 MobiLink 系统表** 高级用户可能想直接在 MobiLink 系统表中添加、删除和更新数据。执行这些操作需要预先了解 MobiLink 的工作原理。

  请参见“MobiLink 服务器系统表” 《MobiLink - 服务器管理》。

用于编写服务器端同步逻辑的选项

可以使用 SQL、Java（使用面向 Java 的 MobiLink 服务器 API）或者 .NET（使用面向 .NET 的 MobiLink 服务器 API）来编写 MobiLink 同步脚本。

如果要与受支持的统一数据库同步，则 SQL 同步逻辑通常是最佳选择。

如果不是要与受支持的统一数据库同步，使用 Java 或 .NET 会很有用。如果您的设计受到 SQL 语言的局限性或者数据库管理系统功能的限制，或者您只需要涵盖不同 RDBMS 类型的可移植性，Java 或 .NET 也非常有用。

Java 和 .NET 同步逻辑可以实现与 SQL 逻辑完全相同的功能。正如 MobiLink 服务器可以在 MobiLink 事件发生时访问 SQL 脚本一样，它也可以在 MobiLink 事件发生时调用 Java 或 .NET 方法。当使用 Java 或 .NET 工作时，可以使用事件进行一些额外处理，但当处理那些直接处理上载或下载行的事件的脚本时，您的实现必须返回 SQL 字符串。除了用于直接行处理的两个事件外，不能直接通过 Java 或 .NET 同步逻辑访问上载和下载：MobiLink 将 Java 或 .NET 返回的字符串作为 SQL 执行。

直接行处理（使用事件 handle_UploadData 和 handle_DownloadData 与某个数据源同步）**真正**实现了直接操纵上载和下载行的功能。

以下是在使用 Java 或 .NET 编写脚本时可能需要考虑的一些场景：

- **直接行处理** 利用 Java 和 .NET 同步逻辑，您可以使用 MobiLink 来访问除了统一数据库外的其它数据源（如应用程序服务器、Web 服务器和文件）中的数据。
- **验证** 可以使用 Java 或 .NET 编写用户验证过程，以使 MobiLink 验证可与公司安全策略相集成。
- **存储过程** 如果您的 RDBMS 不具备使用用户定义存储过程的功能，可使用 Java 或 .NET 创建一种方法。
- **外部调用** 如果程序需要在同步事件中途联系外部服务器，则可使用 Java 或 .NET 同步逻辑执行由同步事件触发的操作。可以在多个连接之间共享 Java 和 .NET 同步逻辑。
- **变量** 如果您的数据库无法处理变量，可使用 Java 或 .NET 创建一个在整个连接或同步中均有效的变量。（或者使用 SQL 脚本，您可以使用用于所有统一数据库类型的用户定义命名参数。请参见“用户定义的命名参数”一节 《MobiLink - 服务器管理》。）

MobiLink 服务器 API

通过 MobiLink 服务器 API 可使用 Java 和 .NET 同步逻辑。MobiLink 服务器 API 是几组用于 MobiLink 同步的类和接口。

面向 Java 的 MobiLink 服务器 API 可以帮助您：

- 像访问 JDBC 连接一样访问与统一数据库的现有 ODBC 连接。
- 使用诸如 JDBC、Web 服务和 JNI 这样的接口访问替代数据源。
- 实现创建与统一数据库的新 JDBC 连接的功能，以便能够在当前同步连接之外更改数据库。例如，即使同步连接执行回退，也可以使用此功能进行错误记录或者审计。

- 与统一数据库同步时，实现在 MobiLink 服务器执行 Java 代码之前编写 Java 代码并进行调试的功能。与 Java 应用程序可以使用的开发环境相比，用于许多数据库管理系统的 SQL 开发环境要相对简单。
- SQL 行处理和直接行处理。
- 完整丰富的 Java 语言及其大量的现有代码和库。

请参见“用于 Java 的 MobiLink 服务器 API 参考”一节 《MobiLink - 服务器管理》。

面向 .NET 的 MobiLink 服务器 API 可以帮助您：

- 使用从 .NET 调用 ODBC 的 iAnywhere 类访问与统一数据库的现有 ODBC 连接。
- 使用诸如 ADO.NET、Web 服务和 OLE DB 这样的接口访问替代数据源。
- 与统一数据库同步时，实现在 MobiLink 服务器执行 .NET 代码前编写 .NET 代码并进行调试的功能。与 .NET 应用程序可以使用的开发环境相比，用于许多数据库管理系统的 SQL 开发环境要相对简单。
- SQL 行处理和直接行处理。
- 在 .NET 公共语言运行库 (CLR) 中运行的代码，这些代码允许访问所有 .NET 库，包括 SQL 行处理和直接行处理。

请参见“用于 .NET 参考的 MobiLink 服务器 API”一节 《MobiLink - 服务器管理》。

另请参见

- “编写同步脚本” 《MobiLink - 服务器管理》
- “同步技术” 《MobiLink - 服务器管理》
- “使用 Java 语言编写同步脚本” 《MobiLink - 服务器管理》
- “使用 .NET 编写同步脚本” 《MobiLink - 服务器管理》
- “直接行处理” 《MobiLink - 服务器管理》

同步过程

同步是 MobiLink 客户端和中心数据源之间的数据交换过程。在此过程中，客户端必须建立并维护与 MobiLink 服务器的会话。如果成功，此会话将使远程数据库和统一数据库保持相互一致的状态。

通常由客户端发起同步过程。此过程首先与 MobiLink 服务器建立连接。

上载和下载

若要上载行，MobiLink 客户端会准备并发送一个**上载**，其中包含自上次同步后在远程数据库上更新、插入或删除的所有行的列表。类似地，若要下载行，MobiLink 服务器会准备并发送一个**下载**，其中包含插入、更新和删除的行的列表。

- **上载** 缺省情况下，MobiLink 客户端会自动跟踪上次成功同步后远程数据库中插入、更新和删除的行。连接一旦建立，MobiLink 客户端就会将包含所有这些更改的列表上载到 MobiLink 服务器。

  上载由远程数据库中被修改行的一组新行值和旧行值组成。（更新具有新行值和旧行值；删除具有旧行值；插入具有新行值。）如果某行被更新或删除，旧行值是指就在上次成功同步后所存在的那些值。如果某一行被插入或更新，新行值是指当前的行值。即使在到达当前状态之前，行已被修改过数次，也不会发送任何中间值。

  MobiLink 服务器接收上载并执行您定义的上载脚本。缺省情况下，所有的更改都会在一次事务中完成应用。完成后，MobiLink 服务器将提交该事务。

- **下载** MobiLink 服务器将使用您创建的同步逻辑，编译要在 MobiLink 客户端上插入、更新或者删除的行的列表。它将这些行下载到 MobiLink 客户端。为了对此列表进行编译，MobiLink 服务器将在统一数据库中打开一个新事务。

  MobiLink 客户端接收下载。当下载到达时，MobiLink 客户端认为统一数据库已成功应用所有上载的更改。并确保这些更改不会再发送到统一数据库中。

  接下来，MobiLink 客户端将自动对下载进行处理，删除旧行、插入新行并更新已更改的行。所有的更改将在远程数据库的一个事务中完成。完成后，MobiLink 客户端将提交该事务。

在 MobiLink 同步过程中，很少有明显的信息交换。客户端建立并上载整个上载。作为响应，MobiLink 服务器建立并下载整个下载。在通信速度较慢而等待时间较长时（例如使用电话线或公共无线网时），对协议的详细程度加以限制是很重要的。

注意
MobiLink 运行时使用 ODBC 隔离级别 SQL_TXN_READ_COMMITTED 作为统一数据库的缺省隔离级别。如果用于统一数据库的 RDBMS 支持快照隔离，并且为数据库启用了快照，则缺省情况下 MobiLink 对下载使用快照隔离。请参见“MobiLink 隔离级别”一节 《MobiLink - 服务器管理》。

另请参见

- “MobiLink 事件概述”一节 《MobiLink - 服务器管理》
- “上载过程中的事件”一节 《MobiLink - 服务器管理》
- “下载过程中的事件”一节 《MobiLink - 服务器管理》

MobiLink 事件

当 MobiLink 客户端发起同步时，将发生多个同步事件。在发生同步事件时，MobiLink 将查找与该事件匹配的脚本。该脚本包含一些说明，详细指出您要完成的操作。如果您已为事件定义了脚本且将脚本放入 MobiLink 系统表中，则该脚本会被调用。

MobiLink 脚本

每次发生事件时，MobiLink 服务器都将执行关联的脚本（如果您已创建了一个关联脚本）。如果没有脚本，将发生序列中的下一个事件。

注意
使用 [**创建同步模型向导**] 创建 MobiLink 应用程序时，会创建所有需要的 MobiLink 脚本。但是，您可以自定义缺省脚本，包括创建新脚本。

以下是表的典型上载脚本。第一个事件 upload_insert 触发 upload_insert 脚本的运行，该脚本将 emp_id 和 emp_name 列的所有更改插入到 emp 表中。upload_delete 和 upload_update 脚本将对 emp 表的删除和更新操作执行类似的功能。

事件	示例脚本内容

upload_insert	`INSERT INTO`
	`emp (emp_id,emp_name)`
	`VALUES {ml r.emp_id}, {ml r.emp_name}`

upload_delete	`DELETE FROM emp` `WHERE emp_id = {ml r.emp_id}`

upload_update	`UPDATE emp`
	`SET emp_name = {ml r.emp_name}`
	`WHERE emp_id = {ml r.emp_id}`

下载脚本使用游标。以下是 download_cursor 脚本的示例：

```
SELECT order_id, cust_id
FROM ULOrder
WHERE last_modified >= {ml s.last_table_download}
AND emp_name = {ml r.emp_id}
```

有关事件和脚本的详细信息，请参见：

- “编写同步脚本” 《MobiLink - 服务器管理》
- “同步事件” 《MobiLink - 服务器管理》

您可以使用 SQL、Java 或 .NET 编写脚本

可以使用统一数据库的本地 SQL 方言或使用 Java 或 .NET 同步逻辑编写脚本。Java 和 .NET 同步逻辑可用于编写由 MobiLink 服务器激活的代码，以连接到数据库、操纵变量、直接操纵上载行操作，或者向下载添加行操作。目前有面向 Java 的 MobiLink 服务器 API 和面向 .NET 的 MobiLink 服务器 API，它们提供能满足同步需求的类和方法。

请参见“用于编写服务器端同步逻辑的选项”一节第 14 页。

有关 RDBMS 相关脚本的信息，请参见“MobiLink 统一数据库” 《MobiLink - 服务器管理》。

存储脚本

SQL 脚本存储在统一数据库的 MobiLink 系统表中。对于使用 MobiLink 服务器 API 编写的脚本，您可将完全限定的方法名作为脚本存储。可以使用多种方法将脚本添加到统一数据库：

- 如果使用 [**创建同步模型向导**]，则在部署项目时会将脚本存储在 MobiLink 系统表中。
- 可以使用建立统一数据库时安装的存储过程，手工向系统表添加脚本。
- 可以使用 Sybase Central 手工向系统表添加脚本。

请参见“添加和删除脚本”一节 《MobiLink - 服务器管理》。

同步过程中的事务

MobiLink 服务器在一个事务中将从每个 MobiLink 客户端上载的更改并入统一数据库中。它在插入新行、删除旧行、更新以及解决任何冲突后，便会提交这些更改。

MobiLink 服务器使用另一个事务准备阻塞下载，其中包括所有的删除、插入以及更新。如果您指定下载确认且客户端确认下载成功，则 MobiLink 服务器将提交下载事务。指定了阻塞下载确认后，如果应用程序遇到问题或无法应答，则 MobiLink 服务器将回退下载事务。缺省情况下，不使用下载确认。

在 SQL 同步脚本中或从 SQL 同步脚本调用的过程或触发器中，不应有任何隐式或显式提交或回退。SQL 脚本内的 COMMIT 或 ROLLBACK 语句会改变同步步骤的事务性质。如果您使用这两个语句，则在出现故障时 MobiLink 将无法保证数据的完整性。

跟踪下载信息

MobiLink 使用上次下载时间戳（存储在远程数据库中），帮助简化创建下载的过程。

下载事务的主要作用是选择统一数据库中的行。如果下载失败，则远程数据库将重新上载与上次下载时间戳相同的时间戳，这样不会丢失数据。

请参见“在脚本中使用上次下载时间”一节 《MobiLink - 服务器管理》。

开始和结束事务

MobiLink 客户端在一个事务中处理下载的信息。行会被插入、更新及删除，从而用统一数据使远程数据库处于最新状态。

MobiLink 服务器使用其它两个事务，一个是在同步开始时，另一个是在结束时。这些事务使您可以记录有关每次同步及其持续时间的信息。这样，您便可以记录有关同步尝试、成功的同步以及同步持续时间的统计数据。因为数据是在过程中的各个点处提交，所以这些事务也可以使您在分析失败的同步尝试时可以提交有用的数据。

另请参见

- “MobiLink 事件概述”一节 《MobiLink - 服务器管理》
- “上载过程中的事件”一节 《MobiLink - 服务器管理》
- “下载过程中的事件”一节 《MobiLink - 服务器管理》

如何处理同步失败

MobiLink 同步具有容错能力。例如，如果通信链接在同步过程中失败，远程数据库和统一数据库的状态将保持一致。

在客户端，故障由一个返回代码指示。

同步失败的处理方式依发生时间的不同而异。以下情况采用不同的方式进行处理：

- **上载中的故障** 如果在建立或应用上载时出现故障，远程数据库的状态将与同步开始时的状态完全相同。在服务器端，任何已应用的上载部分将被回退。
- **上载和下载之间的故障** 如果故障在上载完成之后、MobiLink 客户端接收到下载之前发生，客户端将无法确定上载更改是否成功地应用到统一数据库中。上载可能已被全部应用并提交，故障也可能出现在服务器应用整个上载之前。此时 MobiLink 服务器会自动回退统一数据库中的未完成事务。

  MobiLink 客户端保留所有已上载更改的记录，以防备出现必须再次发送这些更改的情况。下一次客户端同步时，它会在创建新的上载之前请求上一个上载的状态。如果没有提交上一个上载，则新的上载将包含自上次上载时起发生的所有更改。
- **下载中的故障** 如果应用下载时远程设备发生故障，则所有已应用的下载部分都将回退，而远程数据库将与下载前保持相同的状态。

  如果正在使用阻塞下载确认，则 MobiLink 服务器也会回退统一数据库中的已下载事务。

  如果正在使用非阻塞下载确认，则下载事务已经提交，但既不调用 nonblocking_download_ack 脚本，也不调用 publication_nonblocking_download_ack 脚本。

  如果不使用下载确认，则服务器端不受下载故障影响。

在所有可能发生故障的情形中，数据都不会丢失。这将由 MobiLink 服务器与 MobiLink 客户端为您进行管理。开发人员或用户不必对保持应用程序中的数据一致性而担忧。

如何处理上载

当 MobiLink 服务器接收到来自 MobiLink 客户端的上载时，整个上载在同步完成之前是由 MobiLink 服务器存储的。这是因为：

- **过滤下载行** 确定下载行的最常用技术是下载最近一次下载后修改过的行。在同步的时候，上载先于下载进行。在上载期间插入或更新的任何行都是上一次下载后修改过的行。

  很难编写一个从下载中去掉那些作为上载部分而发送的行的 download_cursor 脚本。因此，MobiLink 服务器将自动从下载中删除这些行。

- **处理插入和更新** 缺省情况下，上载中的表会按照可避免参照完整性违规的顺序应用到统一数据库中。上载中的表将根据外键关系排序。例如，如果表 A 和表 C 都有外键引用了表 B 中的主键列，那么首先上载表 B 行的插入和更新。
- **在插入和更新之后处理删除** 删除操作在应用了所有插入和更新之后应用到统一数据库。当应用删除操作时，将以与其在上载中出现方式相反的顺序处理表。当一个要删除的行引用另一个表中也要被删除的行时，这一操作顺序将保证引用行在被引用行之前删除。
- **死锁** 将上载应用于统一数据库时，可能会因为与其它事务并发而出现死锁。这些事务可能是来自其它 MobiLink 服务器数据库连接的上载事务，或者是来自其它使用统一数据库的应用程序事务。上载事务发生死锁时将被回退，而 MobiLink 服务器将自动重新开始应用上载。

> **性能提示**
> 编写同步脚本时务必要尽可能避免争用。在多个用户同时进行同步时，争用会对性能产生明显的影响。

参照完整性与同步

除 UltraLiteJ 以外，所有 MobiLink 客户端在将下载项并入远程数据库时都将实施参照完整性。

缺省情况下，MobiLink 客户端会自动删除所有违反参照完整性的行，而非放弃下载事务。

此功能包括以下优点：

- 同步脚本中的防错保护。由于脚本的灵活性，有可能意外地将破坏远程数据库完整性的行下载下来。MobiLink 客户端可自动保持参照完整性，无需干预。
- 可以使用该参照完整性机制有效地从远程数据库中删除信息。MobiLink 客户端只将删除发送给父记录，从而自动移除所有的子记录。这将大大降低 MobiLink 必须发送给远程数据库的通信量。

如果 MobiLink 客户端必须显式删除行以保持参照完整性，它会如下所述提供通知：

- 对于 SQL Anywhere 客户端，dbmlsync 将在日志中写入一个条目。同时还有 dbmlsync 事件挂接供您使用。请参见：
  - “sp_hook_dbmlsync_download_ri_violation”一节 《MobiLink - 客户端管理》
  - “sp_hook_dbmlsync_download_log_ri_violation”一节 《MobiLink - 客户端管理》
- 对于 UltraLite 客户端，将引发 SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY 警告。此警告以表名为参数。为了保持参照完整性，每删除一行都将引发此警告。如果您希望同步仍然继续，应用程序可以忽略警告。如果希望显式处理警告，可以使用错误回调函数来捕获它们，例如，计算将删除的行数。

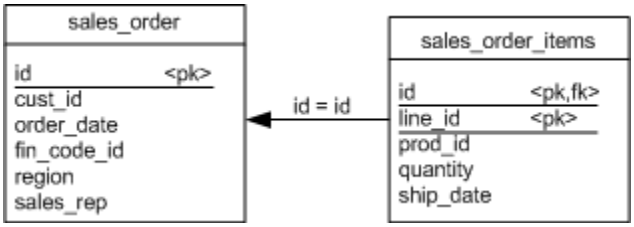
  在引发警告时，如果要放弃同步，则必须执行同步观察器，然后从错误回调函数给观察器发信号（或者通过一个全局变量）。这种情况下，将会在下次调用观察器时放弃同步。

在事务结束时检查参照完整性

MobiLink 客户端在单个事务中并入来自下载的更改。为了提供更大的灵活性，参照完整性检查将在此事务结束处进行。由于检查被延迟，因此数据库可能会暂时出现违反参照完整性的状态。这是因为违反参照完整性的行在提交下载之前将被自动删除。

示例

假设 UltraLite 销售应用程序包含以下两个表。一个表包含销售订单。另一张表是各订单中销售的项目。它们具有以下关系：



如果您使用 sales_order 表的 download_delete_cursor 删除一个订单，缺省参照完整性机制将自动地删除 sales_order_items 表中所有指向被删除销售订单的行。

这种机制有以下优点：

- 您不需要 sales_order_items 表脚本，因为该表中的行将被自动删除。
- 同步的效率得以提高。您不需要下载从 sales_order_items 表中删除的行。如果每个销售订单包含很多项目，由于现在减少了下载，所以性能得以提高。此技术在使用较慢的通信方法时尤其适用。

更改缺省行为

对于 SQL Anywhere 客户端，可以使用 sp_hook_dbmlsync_download_ri_violation 客户端事件挂接来处理参照完整性违规。Dbmlsync 还会在其日志中写入一个条目。

请参见：

- “sp_hook_dbmlsync_download_log_ri_violation”一节 《MobiLink - 客户端管理》
- “sp_hook_dbmlsync_download_ri_violation”一节 《MobiLink - 客户端管理》

安全

在大范围的分布式系统如 MobiLink 系统中，有关确保数据安全性涉及以下几个方面：

- **统一数据库中的数据保护** 统一数据库中的数据可以使用数据库用户验证系统和其它安全性功能进行保护。

  有关详细信息，请参见数据库文档。如果使用的是 SQL Anywhere 统一数据库，请参见“保护数据的安全”《SQL Anywhere 服务器 - 数据库管理》。

- **远程数据库中的数据保护** 如果使用的是 SQL Anywhere 远程数据库，则使用 SQL Anywhere 安全性功能对数据进行保护。缺省情况下，这些功能旨在防止通过客户端/服务器通信进行未经授权的访问，但是却无法防止直接从数据库文件中抽取信息这种后果严重的行为。

  客户端的文件由客户端操作系统的安全性功能进行保护。

  如果使用的是 SQL Anywhere 远程数据库，请参见“保护数据的安全”《SQL Anywhere 服务器 - 数据库管理》。

  如果使用的是 UltraLite 数据库，请参见“保护 UltraLite 数据库”一节《UltraLite - 数据库管理和参考》。

- **同步中的数据保护** 从 MobiLink 客户端到 MobiLink 服务器进行的通信可以通过 MobiLink 传送层安全性功能进行保护。请参见“传送层安全”《SQL Anywhere 服务器 - 数据库管理》。

- **保护同步系统不被未授权用户访问** MobiLink 同步可以由基于口令的用户验证系统提供保护。这种机制可防止未授权用户进行数据同步。请参见“MobiLink 用户”《MobiLink - 客户端管理》。

MobiLink 模型

目录

MobiLink 模型简介

同步模型是一种可用来轻松创建 MobiLink 应用程序的工具。同步模型是一个文件，它由 Sybase Central 中的 [**创建同步模型向导**] 创建。

运行 [**创建同步模型向导**] 时，将提示您连接统一数据库以便获得模式信息。如果尚未将数据库设置为统一数据库，则该向导可应用设置脚本创建同步所需要的 MobiLink 系统表和其它对象；除此之外，在您部署模型之前不对统一数据库进行任何更改。向导完成后，到数据库的连接会被关闭。

完成 [**创建同步模型向导**] 后，模型将以**[模型] 模式**显示。可以使用 [**模型**] 模式自定义模型。如果在 [**模型**] 模式下，则您以脱机方式工作：不对统一数据库进行任何更改。模型存储在扩展名为 *.mlsm* 的模型文件中。

模型完成后，可以使用 [**部署同步模型向导**] 进行部署。[**部署同步模型向导**] 使用您选择的部署选项创建脚本文件以运行 MobiLink 服务器和客户端。在部署时您可以选择对现有数据库进行更改，也可以选择让向导创建您自己要运行的文件。

部署之后，可以继续自定义模型或数据库，然后重新部署。或者，也可以使用在整个 MobiLink 文档中介绍的技术在 [**模型**] 模式外修改已部署的同步系统。然而，在 [**模型**] 模式之外更改同步系统时，不能执行反向工程来将更改返回到模型中。

MobiLink 模型的限制

下面是使用 [**创建同步模型向导**] 和 [**模型**] 模式时的一些限制：

- **在模型之外作出的更改无法重新部署**　如果部署模型后在该模型之外对其进行了更改，则这些更改不会保存在模型中。如果只是将模型用作起始点，而随后部署模型，接着在模型之外进行所有的更改，那么不会出现任何问题。但如果希望能够重新部署该模型，则最好在 [**模型**] 模式中进行更改，以便保存更改并能够重新部署这些更改。
- **DB2 主机**　MobiLink 模型不使用 DB2 主机统一数据库。
- **MobiLink 系统数据库**　不能使用 MobiLink 系统数据库。
- **多个发布**　不能创建多个发布。部署模型之后，可以使用非模型方法（如 CREATE PUBLICATION 语句）添加更多发布，但不能执行反向工程将这些添加的发布返回模型中。
- **视图**　在选择用于表映射的统一数据库表时，不能选择视图。
- **生成的远程数据库**　如果在 [**创建同步模型向导**] 中创建新的远程模式，则新的远程数据库各列将不包含统一数据库中各列的外键、索引或缺省列值。UltraLite 数据库不支持 NCHAR 或 NVARCHAR 列，因此不能使用具有这些数据类型列的统一数据库表来为 UltraLite 远程数据库生成新的远程模式。部署后，您可以创建或更改远程模式，然后更新模型的模式。
- **计算列**　如果想要同步具有计算列的统一数据库表，则不能上载到表。如果部署具有计算列的同步模型，则在创建用于基于时间戳下载的触发器时部署可能会发生错误。解决办法可以是将计算列从同步中排除，或者将表配置为仅下载（使用快照下载或编辑生成的统一 SQL 文件，将计算列从触发器定义中移除）。

  部署新的远程模式来创建新的远程数据库时，复制计算列会导致语法错误。处理计算列时应执行以下操作之一：

- 将同步模型部署到现有远程数据库。
- 从远程模式中排除计算列。请注意，如果要同步具有计算列的统一数据库表，则不能上载到该表。

Microsoft SQL 服务器 AdventureWorks 示例数据库包含计算列。如果使用此数据库创建模型，请将这些计算列设置为仅下载或将其从同步中排除。

- **逻辑删除** MobiLink 同步模型支持逻辑删除，它会假定逻辑删除列仅存在于统一数据库中，而不在远程数据库中。在将统一模式复制到新的远程模式时，将忽略所有与模型的同步设置中的逻辑删除列相匹配的列。对于新模型，将删除缺省列名称。

  将逻辑删除列名称添加至远程模式：

  1. 在向导中，选择 [**使用逻辑删除**]。
  2. 重命名逻辑删除列，以使其不与统一数据库中的任何列名称相匹配。
  3. 完成向导后，更新远程模式并保留缺省表选择。逻辑删除列名称将出现在模式更改列表中，并将被添加到远程模式中。

> **注意**
> 您需要设置列映射，将远程数据库的逻辑删除列映射到统一数据库的逻辑删除列。

部署注意事项

- **对象名过长** 部署时创建的数据库对象名称的长度可能要超过数据库所支持的名称长度（因为新对象名称通过向基表名称添加后缀创建而成）。如果发生这种情况，请仅部署到文件（而非直接部署到数据库）并编辑生成的 SQL 文件以替换出现的所有该过长名称。
- **新远程模式** 如果在 [**创建同步模型向导**] 中创建新的远程模式，则新的远程数据库各列将不包含统一数据库中各列的索引。外键和缺省列值被复制到新的远程数据库中，但是，此支持依赖于由 ODBC 驱动程序返回的数据库元数据，并且可能会因为驱动程序问题而导致语法错误或其它错误。例如，如果驱动程序报告一个缺省列值，但采用的格式不能用于在 SQL Anywhere 或 UltraLite 远程数据库中声明此类缺省列值，则会产生错误（包括部署时的语法错误）。

  部署新的远程模式来创建新的远程数据库时，复制计算列会导致语法错误。处理计算列时应执行以下操作之一：将其部署到现有远程数据库以避免使用模型的远程模式，或者

  - 将同步模型部署到现有远程数据库。
  - 从远程模式中排除计算列。请注意，如果要同步具有计算列的统一数据库表，则不能上载到该表。

  UltraLite 数据库不支持 NCHAR 或 NVARCHAR 列，因此不能使用具有这些数据类型列的统一数据库表来为 UltraLite 远程数据库生成新的远程模式。

  部署后，您可以创建或更改远程模式，然后更新模型的模式。
- **代理表** 可以与用于将表代理到另一数据库的统一数据库表保持同步，但如果使用 TIMESTAMP 列来进行基于时间戳的下载，则还需要将 TIMESTAMP 列同时添加到基表和代理表中。[**部署同步模型向导**] 无法向代理表或其基表添加列，因此您需要使用基表和代理表中的现有列，或是使用影子表或快照下载。

- **实例化视图** 如果您在使用基于时间戳的下载并且选择了将时间戳列添加到统一表中，在部署前您必须禁用依赖于表的任何实例化视图。否则，在尝试变更表时可能会出错。对于 SQL Anywhere 统一数据库，使用 sa_dependent_views 系统过程找出表是否具有相关的实例化视图。请参见 sa_dependent_views 系统过程。

其它注意事项

- **创建基于 Oracle 统一数据库的远程数据库** 当使用 Oracle 统一数据库作为 SQL Anywhere 或 UltraLite 远程数据库的基础时，最好将统一数据库中的 DATE 列更改为 TIMESTAMP。否则，在上载时将丢失次秒级信息。
- **代理表** 如果要使用通过触发器维护的影子表，则触发器和影子表应在基表上。基表上的触发器无法修改用于为基表定义代理表的数据库中的影子表。

创建模型

♦ **使用 [创建同步模型向导] 设置 MobiLink 应用程序**

1. 选择 [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 选择 [**工具**] » [**MobiLink 11**] » [**设置 MobiLink 同步**]。
3. 在 [**欢迎**] 页面上为模型选择名称和位置。模型存储在扩展名为 *.mlsm* 的模型文件中。单击 [**下一步**]。
4. 在 [**主键需求**] 页面中将三个复选框选中。有关主键的详细信息，请参见“维护唯一主键”一节 《MobiLink - 服务器管理》。单击 [**下一步**]。
5. 在 [**统一数据库模式**] 页面上，单击 [**选择统一数据库**] 并选择数据库。单击 [**下一步**]。

   必须连接到 MobiLink 应用程序中的统一数据库，以便向导能够获得它的模式信息。

   如果尚未将此数据库设置为用作统一数据库，向导会提示您进行设置。MobiLink 设置过程会将系统对象添加到 MobiLink 所需的数据库中。如果您选择做出选择，这些对象就会立即添加到统一数据库中。（可以选择稍后在 [**部署同步模型向导**] 中或通过自行应用设置文件进行此设置。）有关详细信息，请参见“建立统一数据库”一节 《MobiLink - 服务器管理》。

   如果您选择其它的统一数据库或在退出向导时，将关闭与该统一数据库的连接。从这时起，您在此向导中所做的更改即是针对模型文件，而不是统一数据库。
6. [**远程数据库模式**] 页面出现。可基于统一数据库或现有远程数据库创建您的远程数据库模式。现有远程数据库可以是 SQL Anywhere 或 UltraLite。（部署时，可将此模式应用至新的或现有的远程数据库。请参见“远程数据库”一节第 27 页。）

   对于新的 SQL Anywhere 远程数据库，远程表的所有者与相应统一数据库表的所有者相同。如果需要不同的所有者，应改用具备所设置的表所有权的现有远程数据库。
7. 请按照 [**创建同步模型向导**] 中的剩余说明进行操作。请尽可能使用基于最佳做法提出的缺省建议值。所有页面都有联机帮助。
8. 单击 [**完成**]。

   单击 [**完成**] 后，刚刚创建的模型即会以 [**模型**] 模式打开。同时，与统一数据库的连接将关闭。您现在以脱机方式工作，且可对模型进行更改。在部署该模型前，不会在模型外进行任何更改：在部署前，统一数据库没有变化，也不创建或更改远程数据库。请参见“模型模式”一节第 30 页和“部署模型”一节第 40 页。

注意

- 一个模型只能有一个发布。请参见“发布数据”一节 《MobiLink - 客户端管理》。
- 一个模型只能有一个版本。请参见“脚本版本”一节 《MobiLink - 服务器管理》。

远程数据库

模型包含远程数据库的模式。该模式可从现有远程数据库或统一数据库中获取。

在以下情况下使用现有远程数据库：

- 如果您已拥有了远程数据库，尤其是其模式不是统一数据库模式的子集的远程数据库。
- 如果统一数据库和远程数据库的列需要具有不同的类型。例如，如果需要将统一数据库上的 NCHAR 列映射到 UltraLite 远程数据库上的 CHAR 列。
- 如果远程数据库表的所有者需要不同于统一数据库表的所有者。对于新的 SQL Anywhere 远程数据库，远程表的所有者与相应统一数据库表的所有者相同。如果需要不同的所有者，应使用具备所设置的表所有权的现有 SQL Anywhere 远程数据库。（UltraLite 数据库没有所有者。）

提示
如果需要更改现有远程数据库的模式，则在模型之外对数据库进行更改，然后运行 [**更新模式向导**] 来更新模型。

部署模型时，无论您如何在模型中创建远程模式，您的远程数据库始终有三个选项。远程数据库的部署时选项为：

- **创建新的远程数据库** 部署可以使用同步模型中的远程模式来创建新的远程数据库。
- **更新没有用户表的现有远程数据库** 如果您部署到一个空的远程数据库，则在该数据库中创建模型中的远程模式。如果您希望使用非缺省的数据库创建选项（如归类），则此选项会派上用场。

  对于 SQL Anywhere 数据库，您可以参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》的注释部分中的选项列表，这些选项无法在数据库创建之后进行设置。

  对于 UltraLite 数据库，数据库属性无法在数据库创建之后进行更改。请参见“为 UltraLite 选择数据库创建参数”一节 《UltraLite - 数据库管理和参考》。
- **更新具有与模型中的模式相匹配的模式的现有远程数据库** 如果您希望同步一个现有的远程数据库，则此选项会很有用。如果直接部署到现有远程数据库，则不会对任何现有远程数据做出更改。如果尝试直接部署到其模式与模型中的远程模式不匹配的现有远程数据库，则系统会提示您更新模型中的远程模式。

对于 SQL Anywhere 远程数据库，表的所有者与原始数据库的所有者相同。（UltraLite 表没有所有者。）

另请参见

- “创建模型”一节第 27 页
- “部署模型”一节第 40 页

更改您的统一数据库

在可以将某一数据库用作 MobiLink 统一数据库之前，必须添加表、列以及触发器等同步所需的对象。可以通过运行针对数据库的安装脚本来实现这一点。每个受支持的 RDBMS 都有一个不同的安装脚本。这些脚本都位于 *install-dir\MobiLink\setup* 中。可以在文本编辑器中打开某个脚本，验证其具体执行的功能。

当在 Sybase Central 中创建 Mobilink 同步模型时，软件可按您的选择自动运行脚本，或者您也可以手工运行脚本。首次启动 [**创建同步模型向导**] 时系统将提示您安装 MobiLink 设置。如果此时不安装，则在部署模型时将再次提示安装。

另请参见

- “建立统一数据库”一节 《MobiLink - 服务器管理》
- “部署模型”一节第 40 页

模型模式

完成 [**创建同步模型向导**] 或打开现有模型时，模型将以 [**模型**] 模式显示。可以使用 [**模型**] 模式进一步自定义模型。如果在 [**模型**] 模式下工作，则您处于脱机状态，并且所做的更改仅更改模型文件。在部署前，不对统一数据库或远程数据库进行任何更改。

管理模式

Sybase Central 的 MobiLink 插件包括两种模式：[**模型**] 模式和 [**管理**] 模式。在工具栏中有一个 [**模式**] 菜单可以在两种模式之间切换。

可以在 [**管理**] 模式中自定义同步应用程序。然而，如果您先部署模型然后在该模型外部做出更改，则不能执行反向工程将更改回复到模型中。因此，如果您计划重新部署模型，则切勿在 [**管理**] 模式下更改模型。

在 Sybase Central 外部修改 MobiLink 应用程序

也可以在 Sybase Central 外部修改已部署的模型。

例如，可以使用系统过程添加或修改 MobiLink 脚本。请参见“MobiLink 系统过程”一节《MobiLink - 服务器管理》。

当在 Sybase Central 外部进行更改时，针对 [**管理**] 模式的规则同样适用：不能执行反向工程将更改返回模型中。

修改表映射和列映射

表映射指示应同步哪些表、应如何同步表及如何在远程数据库和统一数据库之间映射这些同步数据。

仅上载、仅下载和非同步的表或列

缺省情况下，MobiLink 执行完整的双向同步。每个表都可以更改为仅上载或仅下载。您也可以选择不同步表。

在模型中，只能将表指定为仅下载，但不能创建仅下载发布。这是因为一个模型只能有一个发布。

♦ **更改表映射方向**

1. 在 [**模型**] 模式中打开 [**映射**] 选项卡。
2. 在 [**表映射**] 窗格中选择一个远程表。
3. 在 [**映射方向**] 下拉列表中选择以下选项之一：
   - **双向**
   - **只上载到统一的**
   - **只下载到远程**
   - **未同步**

> **小心**
> 缺省情况下，不同步影子表。切勿尝试对其执行同步操作。

4. 必要时，从 [**统一表**] 下拉列表中选择要映射到的统一表。

♦ **不同步列**

1. 在 [**模型**] 模式中打开 [**映射**] 选项卡。
2. 在 [**表映射**] 窗格中选择一个表。

   该表的列信息会显示在下部窗格的 [**列映射**] 选项卡中。
3. 选择一列。
4. 在 [**映射方向**] 下拉列表中选择 [**未同步**]。

   请注意，必须同步主键。

更改表映射和列映射

如果模型基于某现有远程数据库，则列映射表示最佳猜测。您应检查这些映射并根据需要自定义它们。

♦ **更改表映射**

1. 在 [**模型**] 模式中打开 [**映射**] 选项卡。
2. 在 [**表映射**] 窗格中选择一个表。
3. 更改所映射的统一表：从 [**统一表**] 下拉列表中，选择另一不同表。
4. 更改所映射的远程表：
   - 如果远程数据库以统一数据库为基础，请使用 [**创建新远程表**] 窗口。请参见“在远程数据库模式基于统一数据库时创建远程表”一节第 31 页。
   - 如果远程数据库基于现有远程数据库：从 [**远程表**] 下拉列表中，选择另一不同表。
5. 更改表的列映射：选择表，然后打开下部窗格中的 [**列映射**] 选项卡。

修改模型创建的远程数据库

可以按以下操作在模型中修改远程数据库的模式。

在远程数据库模式基于统一数据库时创建远程表

要向模型中的远程数据库模式添加表，请使用 [**创建新远程表**] 窗口。这些表被添加到模型中并进行同步映射。要在 [**模型**] 模式中打开 [**创建新远程表**] 窗口，请选择 [**文件**] » [**新建**] » [**远程表**]。

如果希望向远程数据库中添加表而该表却不在统一数据库中，则可以将该表添加到统一数据库中，运行 [**更新模式向导**]，然后使用 [**创建新远程表**] 窗口将该表添加到模型中。要在 [**模型**] 模式中打开 [**更新模式向导**]，请从 [**文件**] 菜单中选择 [**更新模式**]。

请参见“在 [模型] 模式中更新模式”一节第 39 页。

在远程数据库模式基于现有远程数据库时创建远程表

如果想为现有远程数据库创建新表，则在模型外修改远程数据库，然后使用 [**更新模式向导**] 来更新模型中的远程数据库模式。随后需要在 [**映射**] 选项卡中映射新远程表。请参见：

- “在 [模型] 模式中更新模式”一节第 39 页
- “修改表映射和列映射”一节第 30 页

删除远程表和列

可利用 [**模型**] 模式删除位于模型中的远程数据库模式中的表和列。右击该行并选择 [**删除**]，可将某远程表或列标记为删除。在保存模型时，会从模式中删除该远程表或列。删除某个远程表或列意味着在部署到新的远程数据库时不会在该远程数据库中创建该表或该列。

不能在 [**模型**] 模式下从统一数据库中删除表或列。要更改统一模式，可在 [**模型**] 模式外修改统一数据库，然后运行 [**更新模式向导**]。

注意
不能删除主键。

修改下载类型

下载类型可以是时间戳、快照或自定义。可在 [**映射**] 选项卡的 [表映射] 窗格中更改下载类型。

- **基于时间戳的下载** 选择此选项可使用基于时间戳的下载作为缺省选项。仅下载自上次同步后所更改的行。请参见“基于时间戳的下载”一节 《MobiLink - 服务器管理》。
- **快照下载** 选择此选项可使用快照下载作为缺省选项。即使自上次同步后并未对行进行更改，也会下载所有的行。请参见：
  - “快照同步”一节 《MobiLink - 服务器管理》
  - “何时使用快照同步”一节 《MobiLink - 服务器管理》
- **自定义下载逻辑** 如果想要编写自己的 download_cursor 和 download_delete_cursor 脚本而不是生成这些脚本，请选择此选项。请参见：
  - “编写同步脚本” 《MobiLink - 服务器管理》
  - “编写 download_cursor 脚本”一节 《MobiLink - 服务器管理》
  - “编写 download_delete_cursor 脚本”一节 《MobiLink - 服务器管理》

♦ 更改下载类型

1. 在 [**模型**] 模式中打开 [**映射**] 选项卡。
2. 在 [**表映射**] 窗格中选择一个远程表。
3. 在 [**下载类型**] 下拉列表中，选择 [**时间戳**]、[**快照**] 或 [**自定义**]。
4. 如果选择 [**自定义**]，则右键单击表并选择 [**转到事件**]，以打开 [**事件**] 选项卡。

5. 使用合适的业务逻辑编辑您的 download_cursor 脚本和 download_delete_cursor 脚本。

修改处理删除的方式

如果使用快照下载，则会在下载快照之前删除远程数据库中的所有行。如果使用基于时间戳的下载，则可以确定在远程数据库上处理统一数据库中的删除的方式。

如果在将行从统一数据库中删除时也希望将其从远程数据库中删除，则需要保留该行的记录才能删除它。可以使用影子表或逻辑删除来执行此操作。

♦ 更改处理删除的方式

1. 在 [**模型**] 模式中打开 [**映射**] 选项卡。
2. 在 [**表映射**] 窗格中选择一个远程表。
3. 如果想从统一数据库下载删除项，可在 [**删除**] 列中选中此复选框。如果不希望从统一数据库下载删除，请清除此复选框。
4. 如果选择下载删除，请打开下部窗格中的 [**下载删除**] 选项卡。

   要记录删除，可以选择使用影子表或逻辑删除。

另请参见

- “处理删除”一节 《MobiLink - 服务器管理》
- “编写 download_delete_cursor 脚本”一节 《MobiLink - 服务器管理》
- “download_cursor 表事件”一节 《MobiLink - 服务器管理》

修改下载子集

每个 MobiLink 远程数据库都可以同步统一数据库中的数据子集。可以为每个表自定义下载子集。

下载子集选项为：

- **用户** 选择此选项可按 MobiLink 用户名对数据进行分区，这样会将不同的数据下载到不同的已注册 MobiLink 用户。

  要使用此选项，MobiLink 用户名必须在统一数据库上。您可在部署时选择 MobiLink 用户名，以便选择与统一数据库上现有值匹配的名称。（用于 MobiLink 用户名的列所属的类型必须支持您用于用户名的值。）如果 MobiLink 用户名不在您进行子集划分的表中，则必须连接其所在的表。

- **远程** 选择此选项可按远程 ID 对数据进行分区，这样会将不同的数据下载到不同的远程数据库。

  要使用此选项，远程 ID 必须在您的统一数据库上。缺省情况下，会将远程 ID 创建为 GUID，但可对远程 ID 进行设置，使其匹配统一数据库上的现有值。（用于远程 ID 的列所属的类型必须支持您用于远程 ID 的值。）如果远程 ID 不在您进行子集划分的表中，则必须连接其所在的表。

- **自定义** 选择此选项可使用 SQL 表达式，以确定下载哪些行。每个同步仅下载 SQL 表达式为 true 的那些行。此 SQL 表达式与 download_cursor 脚本中所使用的表达式相同。部分是为您生成。

♦ 更改下载子集

1. 在 [**模型**] 模式中打开 [**映射**] 选项卡。
2. 在 [**表映射**] 窗格中选择一个远程表。
3. 在 [**下载子集**] 下拉列表中，选择以下下载子集之一：[**无**]、[**用户**]、[**远程**] 或 [**自定义**]。
4. 如果选择 [**用户**]、[**远程**] 或 [**自定义**]，请打开下部窗格中的 [**下载子集**] 选项卡。
5. 如果选择 [**用户**] 或 [**远程**]，则可以利用 [**下载子集**] 选项卡来标识统一表中包含 MobiLink 用户名或远程 ID 的列，或者输入表连接以获取 MobiLink 用户名或远程 ID。
6. 如果选择 [**自定义**]，则 [**下载子集**] 选项卡有两个文本框，您可以向其中添加信息以构造 download_cursor 脚本。不必编写完整的 download_cursor。只需添加额外信息以标识下载子集上的连接和其它限制即可。
   - 如果您的 download_cursor 脚本需要到其它表的连接，可在第一个文本框（[**要添加到下载游标的 FROM 子句中的表**]）中输入表名。如果连接需要多个表，则用逗号分隔它们。
   - 在第二个文本框（[**要在下载游标的 WHERE 子句中使用的 SQL 表达式**]）中，输入指定连接和下载子集的 WHERE 条件。

另请参见

- “MobiLink 用户简介”一节 《MobiLink - 客户端管理》
- “远程 ID”一节 《MobiLink - 客户端管理》
- “在远程数据库之间对行进行分区”一节 《MobiLink - 服务器管理》
- “在脚本中使用远程 ID 和 MobiLink 用户名”一节 《MobiLink - 客户端管理》
- “在脚本中使用上次下载时间”一节 《MobiLink - 服务器管理》
- “编写 download_cursor 脚本”一节 《MobiLink - 服务器管理》

示例（用户）

例如，CustDB 中的 ULOrder 表可在用户之间共享。缺省情况下，会将订单指派给创建它们的雇员。但有时另一位雇员需要查看其他雇员所创建的订单。例如，经理可能需要查看部门内雇员创建的所有订单。CustDB 数据库可通过 ULEmpCust 表提供这些订单。这样您就可以将客户指派给雇员。他们下载该雇员客户关系的所有订单。

要想知道这是如何完成的，可先查看 ULOrder 的 download_cursor 脚本，但不要下载子集。选择 [**映射**] 选项卡中的 ULEmpCust 表。为 [**下载类型**] 列选择 [基于时间戳]，为 [**下载子集**] 列选择 [无]。右击该表并选择 [**转到事件**]。该表的 download_cursor 脚本如下所示：

```
SELECT "DBA"."ULOrder"."order_id",
 "DBA"."ULOrder"."cust_id",
 "DBA"."ULOrder"."prod_id",
 "DBA"."ULOrder"."emp_id",
 "DBA"."ULOrder"."disc",
 "DBA"."ULOrder"."quant",
 "DBA"."ULOrder"."notes",
 "DBA"."ULOrder"."status"
```



```
FROM "DBA"."ULOrder"
WHERE "DBA"."ULOrder"."last_modified" >= {ml s.last_table_download}
```

现在，回到 [**映射**] 选项卡。将 ULOrder 的 [**下载子集**] 列更改为 [用户]。打开下部窗格中的 [**下载子集**] 选项卡。选择 [**在已连接关系表中使用列**]。对要连接的表，选择 ULEmpCust。对要匹配的列，选择 emp_id。连接条件应为 emp_id = emp_id。

在顶部窗格中右击该表并选择 [**转到事件**]。现在，该表的 download_cursor 脚本如下所示：

```
SELECT "DBA"."ULOrder"."order_id",
   "DBA"."ULOrder"."cust_id",
   "DBA"."ULOrder"."prod_id",
   "DBA"."ULOrder"."emp_id",
   "DBA"."ULOrder"."disc",
   "DBA"."ULOrder"."quant",
   "DBA"."ULOrder"."notes",
   "DBA"."ULOrder"."status"
FROM "DBA"."ULOrder", "DBA"."ULEmpCust"
WHERE "DBA"."ULOrder"."last_modified" >= {ml s.last_table_download}
AND "DBA"."ULOrder"."emp_id" = "DBA"."ULEmpCust"."emp_id"
AND "DBA"."ULEmpCust"."emp_id" = {ml s.username}
```

示例（自定义）

例如，假设您要按 MobiLink 用户对 Customer 表的下载进行子集划分，并且仅要下载 active=1 的行。而 MobiLink 用户名不在您进行子集划分的表中，因此您需要创建与名为 SalesRep 的表的连接，该表中包含这些用户名。

在 [**映射**] 选项卡中，为 Customer 表的 [**下载类型**] 列选择 [**基于时间戳**]，为 [**下载子集**] 列选择 [**自定义**]。打开下部窗格中的 [**下载子集**] 选项卡。在第一个文本框（[**要添加到下载游标的 FROM 子句中的表**]）中，键入：

```
SalesRep
```

在第二个文本框（[**要在下载游标的 WHERE 子句中使用的 SQL 表达式**]）中，键入：

```
SalesRep.ml_username = {ml s.username}
   AND Customer.active = 1
   AND Customer.cust_id = SalesRep.cust_id
```

在顶部窗格中右击该表并选择 [**转到事件**]。现在，该表的 download_cursor 脚本如下所示：

```
SELECT "DBA"."Customer"."cust_id",
 "DBA"."Customer"."cust_name"
FROM "DBA"."Customer", SalesRep
WHERE "DBA"."Customer"."last_modified" >= {ml s.last_table_download}
 AND SalesRep.ml_username = {ml s.username}
 AND Customer.active = 1
 AND Customer.cust_id = SalesRep.cust_id
```

WHERE 子句的最后一行创建了 Customer 与 SalesRep 的键连接。

修改冲突检测和冲突解决

如果在远程数据库和统一数据库中均更新某行，则下次同步数据库时会发生冲突。

可使用以下选项来检测冲突：

- **无冲突检测** 如果不希望有任何冲突检测，请选择此选项。应用已上载的更新而不检查冲突。这样就不必从统一数据库读取当前的行值，因此同步更新的速度可能会更快。
- **基于行的冲突检测** 如果自上次同步后行已由远程和统一数据库更新，则会检测到冲突。

  此选项定义 upload_fetch 和 upload_update 脚本。请参见“使用 upload_fetch 脚本检测冲突”一节 《MobiLink - 服务器管理》。
- **基于列的冲突检测** 如果远程数据库和统一数据库中行的相同列都已更新，则会检测到冲突。

  此选项定义 upload_fetch_column_conflict 脚本。请参见“使用 upload_fetch 脚本检测冲突”一节 《MobiLink - 服务器管理》。

  如果表中包含 BLOB 且选择 [基于列的冲突检测]，则使用 [基于行的冲突检测]。

可使用以下选项来解决冲突：

- **统一** 先入优先：拒绝存在冲突的上载更新。
- **远程** 后入优先：始终应用上载更新。

  仅在使用基于列的冲突检测时，才使用此选项。否则，如果您选择无冲突检测，可获得同样的结果和更佳的性能。
- **时间戳** 最新的更新优先。要使用此选项，您必须为表创建和维护时间戳列。此时间戳列应记录上次更改行的时间。该列在统一数据库和远程数据库中都应存在。远程数据库和统一数据库必须使用相同的时区（首选 UTC）且其时钟必须同步才可正常工作。
- **自定义** 编写自己的 resolve_conflict 脚本。向导完成后，可在 [**事件**] 选项卡中完成此工作。

  请参见“使用 resolve_conflict 脚本解决冲突”一节 《MobiLink - 服务器管理》。

♦ 自定义冲突检测和冲突解决

1. 在 [**模型**] 模式中打开 [**映射**] 选项卡。
2. 在 [**表映射**] 窗格中选择一个远程表。
3. 在 [**冲突检测**] 下拉列表中，选择 [**无**]、[**基于行**] 或 [**基于列**]。如果选择 [**无**]，则操作到此结束。
4. 如果选择了 [**基于行**] 或 [**基于列**]，请从 [**冲突解决**] 下拉列表中选择 [**统一**]、[**远程**]、[**时间戳**] 或 [**自定义**]。
5. 如果选择 [**时间戳**] 冲突解决，请打开下部窗格中的 [**冲突解决**] 选项卡，然后输入要使用的时间戳列的名称。
6. 如果选择 [**自定义**] 冲突解决，请打开 [**事件**] 选项卡，然后为此表编写 resolve_conflict 脚本。

另请参见

- “冲突处理”一节 《MobiLink - 服务器管理》

在模型中修改脚本

在 MobiLink [**模型**] 模式中打开 [**事件**] 选项卡以执行以下操作：

- 查看和修改由 [**创建同步模型向导**] 生成的脚本。
- 创建新脚本。

可从 [**事件**] 选项卡顶部获知所选脚本所属的组。为了您的方便，将一个表的所有脚本归组到一起。还可从 [**事件**] 选项卡顶部获知所选脚本的名称及该脚本是否由 [**创建同步模型向导**] 生成、是否为用户定义脚本或者是否替换了生成的脚本。还可获知编写同步逻辑使用的语言是 SQL、.NET 还是 Java。

添加或更改脚本时，脚本会完全在您的控制之下；当在 [**模型**] 模式中更改某相关设置时，该脚本不会自动更改。例如，如果您更改了模型的 download_delete_cursor 脚本，然后在 [**模型**] 模式中清除 [**删除**]，则您的自定义 download_delete_cursor 脚本不会受到影响。

可以使用 [**文件**] 菜单中的选项来恢复已更改的生成脚本、恢复已设置为要忽略的脚本或删除已添加的新脚本。选择要恢复或删除的脚本，然后选择 [**文件**] 以查看您的选项。

要查找某特定表的脚本，可打开 [**映射**] 选项卡，选择行，然后选择 [**文件**] » [**转到事件**]。[**事件**] 选项卡随即在相应的表中打开。

在 [模型] 模式中向外部服务器验证

要向外部 POP3、IMAP 或 LDAP 服务器验证，请在 [**模型**] 模式中打开 [**验证**] 选项卡，然后选择 [**为此同步模型启用自定义验证**]。

必须输入有关主机和端口的信息，如果是 LDAP 服务器，则需输入 LDAP 服务器的 URL。

有关这些字段的详细信息，请参见“外部验证程序属性”一节 《MobiLink - 客户端管理》。

在 [模型] 模式中设置服务器启动的同步

服务器启动的同步允许您在统一数据库中发生某些更改时在客户端启动同步。[**模型**] 模式提供了设置服务器启动的同步的方法。此方法提供了便于设置和运行的有限版本的服务器启动同步。

[通知] 选项卡

♦ **设置服务器启动的同步（Sybase Central [模型] 模式和 [部署同步模型向导]）**

1. 使用 [**创建同步模型向导**] 创建 MobiLink 模型。
2. 在 [**模型**] 模式中已打开模型时，在该模型的顶部打开 [**通知**] 选项卡。
3. 选择 [**启用服务器启动的同步**]。
4. 选择一个用于通知的统一数据库表。

   对此表中的数据进行更改会导致向远程数据库发送通知。该通知触发同步。

   仅可以为此目的选择表，对此，您已经定义了基于时间戳的下载游标（缺省值）。该通知基于下载游标的内容。

   请参见“编写 download_cursor 脚本”一节 《MobiLink - 服务器管理》。

5. 选择轮询间隔。此时间为两次轮询之间的时间。可以选择预定义的轮询间隔时间，也可以输入间隔时间。缺省值是 30 秒。

   如果通告程序丢失数据库连接，它将在数据库重新可用后的第一个轮询间隔自动恢复。

6. 或者，更改通告程序数据库连接的隔离级别。缺省值为 [读取已提交的]。

   请注意设置隔离级别的后果。较高的级别会增加争用，并可能导致性能降低。隔离级别 0 允许读取未提交的数据—最终可能被回退的数据。

7. 还可以更改通过其发送通知的网关。缺省为 default_device_tracker 网关。请参见“网关和运营公司”一节 《MobiLink - 服务器启动的同步》。

♦ 使用服务器启动的同步部署模型

1. 部署模型：
   a. 从 [**文件**] 菜单中选择 [**部署**]。
   b. 请按照 [**部署同步模型向导**] 中的说明进行操作。

      请参见“部署模型”一节第 40 页。
   c. 在 [**服务器启动的同步监听器**] 页面上，为监听器配置选项。
2. 完成模型部署。有关所创建文件的信息，请参见“同步已部署的模型”一节第 42 页。
3. 要使用服务器启动的同步，必须完成以下操作：
   a. 启动 MobiLink 服务器。
   b. 执行第一次同步（如果尚未执行同步）。
   c. 启动监听器。
4. 浏览到第一次启动 [**创建同步模型向导**] 时选择的目录。该目录保存扩展名为 *.mlsm* 的模型。该目录还保存以下子目录：
   - *\mlsrv*
   - *\remote*
   - *\consolidated*

关于服务器启动的同步（不在 [模型] 模式中）

在 [**模型**] 模式版本的服务器启动同步中，MobiLink 服务器使用表的 download_cursor 脚本来确定何时启动同步。实现方法是使用 download_cursor 脚本为通告程序生成 request_cursor。如果使用此版本的服务器启动的同步，则不能自定义您的 request_cursor。

请参见“编写 download_cursor 脚本”一节 《MobiLink - 服务器管理》和“request_cursor 事件”一节 《MobiLink - 服务器启动的同步》。

[**模型**] 模式还设置一个缺省的设备跟踪器网关来发送通知。您可以自定义网关。请参见“网关和运营公司”一节 《MobiLink - 服务器启动的同步》。

虽然 Sybase Central 的 [**模型**] 模式提供了一个简化版的服务器启动的同步，但您也可以设置一个完整版的服务器启动的同步。有关服务器启动同步的完整实现的说明，请参见 MobiLink - 服务器启动的同步。

有关在 [**模型**] 模式之外设置服务器启动同步需要进行哪些操作的概述，请参见“服务器启动的同步快速入门指南”一节 《MobiLink - 服务器启动的同步》。

在 [模型] 模式中更新模式

可利用 [**更新模式向导**] 在模型中更新统一数据库和远程数据库模式。

在您部署完模型并执行了以下操作后，[**更新模式向导**] 才能发挥其最大效用：

- 对需要包括在该模型中的远程数据库模式做了更改。
- 对需要包括在该模型中的统一数据库模式做了更改。

  例如，在重新部署为一个或多个表创建了基于时间戳下载的模型之前，需要运行 [更新模式]。由于先前的部署通过添加时间戳列或影子表而更改了统一数据库的模式，因此需要更新该模式。

与用于向模型添加远程表的 [**创建新远程表**] 窗口不同，[**更新模式向导**] 不会映射表。需要使用 [**映射**] 选项卡创建表映射。请参见“修改模型创建的远程数据库”一节第 31 页。

♦ 更新模式

1. 在 [**模型**] 模式中，选择 [**文件**] » [**更新模式**]。
2. 选择以下选项之一：
   - **统一数据库模式**　更新了模型中的统一模式。模型中的远程模式未作更改。
   - **远程数据库模式**　更新了模型中的远程模式。模型中的统一模式未作更改。
   - **统一和远程数据库模式**　模型中的统一模式和远程模式均得到更新，以与现有数据库的模式相匹配。
3. 按照 [**更新模式向导**] 中的说明进行操作。

   单击 [**完成**] 后，将关闭与统一数据库和远程数据库的所有连接。在部署该模型前，不会在模型外进行任何更改：在部署前，统一数据库没有变化，也不创建或更改远程数据库。
4. 在 [**映射**] 选项卡中映射新远程表。请参见“修改表映射和列映射”一节第 30 页。

部署模型

准备好尝试所建模型后，可以使用 [**部署同步模型向导**] 对其进行部署。可以部署许多内容：

- 对统一数据库进行的更改。
- SQL Anywhere 或 UltraLite 远程数据库（可选择创建数据库、将表添加到现有空数据库中，或使用已包含您的远程表的现有数据库）。
- 用于部署模型的批处理文件（生成的批处理文件在文件开头具有变量声明，可在运行批处理文件之前对其进行编辑）。
- 用于运行 MobiLink 服务器和 MobiLink 客户端的批处理文件。
- 服务器启动的同步配置。

部署模型时会保存模型文件。

部署到统一数据库

[**部署同步模型向导**] 提供了两个用于部署到统一数据库的选项：

- 通过填充 MobiLink 系统表和创建所需的所有影子表、列、触发器和存储过程，将模型直接应用到统一数据库。还可以选择创建能够运行 MobiLink 应用程序的批处理文件。
- 创建包含所有相同更改的 SQL 文件。可以随时检查、变更和运行此文件。其效果与直接应用更改相同。

♦ 从 SQL 文件部署统一数据库

- 在运行 [**部署同步模型向导**] 时，如果选择创建一个在以后运行的文件（在 [**统一数据库部署目标**] 页面上），则必须运行位于模型的 *consolidated* 子文件夹中的批处理文件。此文件创建您已选择在统一数据库中创建的所有对象，其中包括同步脚本、影子表和触发器。该文件也可在统一数据库中注册 MobiLink 用户。

  要运行此文件，请浏览到 *consolidated* 目录，然后运行以 *_consolidated.bat* 结尾的文件。必须将连接信息包括在内。例如，运行：

  ```
  MyModel_consolidated.bat
  "dsn=my_odbc_datasource;uid=myuserid;pwd=mypassword"
  ```

  对于一些驱动器，DSN 可以具有用户 ID 和口令，因此不需要指定它们。

注意
如果您的部署创建影子表，则必须以基表（为这些基表创建影子表）所有者的身份或管理员身份连接统一数据库。

部署远程数据库

可以选择使用现有的远程数据库或利用向导创建一个数据库。向导可以直接创建远程数据库，或者您可以利用向导创建一个 SQL 文件，然后运行该文件来创建远程数据库。

利用缺省数据库创建选项，向导使用您在模型中指定的数据库所有者创建远程数据库（SQL Anywhere 或 UltraLite）。另一方面，也可以在 [**创建同步模型向导**] 之外使用您自己的自定义设置

来创建远程数据库，然后使用该向导添加所需的远程表，也可以部署到已经含有远程表的现有远程数据库。

♦ **从 SQL 文件部署远程数据库**

- 在运行 [**部署同步模型向导**] 时，如果选择创建一个在以后运行的文件（在 [**新 SQL Anywhere 远程数据库**] 页面或 [**新 UltraLite 远程数据库**] 页面上），则必须运行位于 *remote* 目录中的批处理文件。此文件将创建您已选择要在远程数据库中创建的所有对象，其中包括表、发布、预订和 MobiLink 用户。

  要运行此文件，请浏览到 *remote* 目录，然后运行以 *_remote.bat* 结尾的文件。例如，运行：

  ```
  MyModel_remote.bat
  ```

  如果要使用现有远程数据库，系统将提示您输入口令。

部署批处理文件以运行同步工具

向导可以创建以下批处理文件：

- 使用指定的选项运行 MobiLink 服务器的批处理文件。
- 对于 SQL Anywhere 远程数据库，批处理文件使用指定的选项运行 dbmlsync。
- 对于 UltraLite 远程数据库，批处理文件使用指定的选项运行 ulsync。Ulsync 用于测试同步，因此没有可用的 UltraLite 应用程序时，它可以帮助您开始工作。
- 如果正在设置服务器启动的同步，[**部署同步模型向导**] 还可以创建可运行通告程序和监听器的批处理文件。

♦ **部署模型**

1. 在 [**模型**] 模式中，选择 [**文件**] » [**部署**]。
2. 请按照 [**部署同步模型向导**] 中的说明进行操作。
3. 完成后，所选更改即部署完毕。如果有同名现有文件，将覆盖这些文件。
4. 要同步应用程序，请参见“同步已部署的模型”一节第 42 页。

重新部署模型

部署模型后，可对其进行更改。通过在 [**模型**] 模式中进行更改然后再重新部署即可实现此操作。还可以在 Sybase Central [**管理**] 模式中变更已部署的应用程序，或者使用系统过程或其它方法直接更改数据库。然而，在 [**模型**] 模式之外变更已部署的模型时，不能执行反向工程来将更改返回到模型中。重新部署模型时，将覆盖在模型外部对您的同步应用程序作出的更改。

如果对远程数据库或统一数据库的模式进行了任何更改，则在重新部署前需要在模型中更新该模式。部署经常会导致模式更改，因此即使您未做任何其它更改可能也需要更新该模式。例如，如果部署一个模型，该模型在统一数据库的每个同步表中都添加一个时间戳列（此为创建模型时的缺省行为），则在重新部署之前需要更新模型中的统一模式。同样，如果向统一数据库添加了一个表然后希望重新部署，则需要更新模型中的统一模式，然后再创建新的远程表。

要运行 [**更新模式向导**]，请选择 [**文件**] » [**更新模式**]。

请参见“在 [模型] 模式中更新模式”一节第 39 页。

同步已部署的模型

部署模型时，可以选择在 [**创建同步模型向导**] 的第一页上所选择的位置处创建目录和文件。将根据当时所选择的模型名称为这些文件和目录命名。

假定您将模型命名为 **MyModel** 并保存在 *c:\SyncModels* 中。根据所选择的部署选项，可以有以下文件：

目录（基于示例的名称和位置）	说明和内容（基于示例的名称）
c:\SyncModels	包含模型文件，保存为 *MyModel.mlsm*。
c:\SyncModels\MyModel	包含保存部署文件的文件夹。
c:\SyncModels\MyModel\consolidated	包含用于统一数据库的部署文件： ● *MyModel_consolidated.sql* - 用于设置统一数据库的 SQL 文件。 ● *MyModel_consolidated.bat* - 用于运行 SQL 文件的批处理文件。
c:\SyncModels\MyModel\mlsrv	包含用于 MobiLink 服务器的部署文件： ● *MyModel_mlsrv.bat* - 用于运行 MobiLink 服务器的批处理文件。如果已经设置服务器启动的同步，则运行该文件也将启动通告程序。

目录（基于示例的名称和位置）	说明和内容（基于示例的名称）
c:\SyncModels\MyModel\remote	包含用于远程数据库的部署文件： ● *dblsn.txt* - 如果设置服务器启动的同步，它将是包含监听器选项设置的文本文件。该文件由 *MyModel_dblsn.bat* 使用。 ● *MyModel_dblsn.bat* - 如果设置服务器启动的同步，该文件为用于运行监听器的批处理文件。 ● *MyModel_dbmlsync.bat* - 如果部署了一个 SQL Anywhere 远程数据库，则它是用于通过 dbmlsync 同步 SQL Anywhere 数据库的批处理文件。 ● *MyModel_remote.bat* - 用于运行 *MyModel_remote.sql* 的批处理文件。 ● *MyModel_remote.db* - 如果选择创建一个新的 SQL Anywhere 远程数据库，则此文件即是该数据库。 ● *MyModel_remote.sql* - 用于设置新 SQL Anywhere 远程数据库的 SQL 文件。 ● *MyModel_remote.udb* - 如果选择创建一个新的 UltraLite 远程数据库，则此文件即是该数据库。 ● *MyModel_ulsync.bat* - 如果部署了一个 UltraLite 数据库，则为用于通过 ulsync 实用程序来测试与 UltraLite 远程数据库的同步的批处理文件。

运行批处理文件

必须从命令行运行由 [**部署同步模型向导**] 创建的批处理文件，并且必须将连接信息包括在内。在运行这些批处理文件之前，可能需要创建 ODBC 数据源。

请参见“创建 ODBC 数据源”一节 《SQL Anywhere 服务器 - 数据库管理》。

♦ 使用批处理文件同步模型

1. 如果尚未在统一数据库上运行 MobiLink 设置脚本，则在部署之前运行这些脚本。

   请参见“建立统一数据库”一节 《MobiLink - 服务器管理》。

2. 在运行 [**部署同步模型向导**] 时，如果选择创建一个在以后运行的文件（在 [**统一数据库部署目标**] 页面上），则必须运行位于模型的 *consolidated* 子文件夹中的批处理文件。此文件创建您已选择在统一数据库中创建的所有对象，其中包括同步脚本、影子表和触发器。该文件也可在统一数据库中注册 MobiLink 用户。

   要运行此文件，请浏览到 *consolidated* 目录，然后运行以 *_consolidated.bat* 结尾的文件。必须将连接信息包括在内。例如，运行：

   ```
   MyModel_consolidated.bat "dsn=MY_ODBC_DATASOURCE"
   ```

3. 在运行 [**部署同步模型向导**] 时，如果选择创建一个在以后运行的文件（在 [**新 SQL Anywhere 远程数据库**] 页面或 [**新 UltraLite 远程数据库**] 页面上），则必须运行位于 *remote* 目录中的批处理文件。此文件将创建您已选择要在远程数据库中创建的所有对象，其中包括表、发布、预订和 MobiLink 用户。

要运行此文件，请浏览到 *remote* 目录，然后运行以 *_remote.bat* 结尾的文件。例如，运行：

```
MyModel_remote.bat
```

如果要使用现有远程数据库，系统将提示您输入口令。

4. 通过运行 *mlsrv\MyModel_mlsrv.bat* 启动 MobiLink 服务器。如果设置服务器启动的同步，则运行该文件也将启动通告程序。必须将连接信息包括在内。例如，运行：

   ```
   MyModel_mlsrv.bat "dsn=MY_ODBC_DATASOURCE"
   ```

5. 执行同步。

   对于 SQL Anywhere 远程数据库：

   - 将 REMOTE DBA 权限授予非 DBA 用户（建议）。例如，使用 Interactive SQL 执行以下语句：

     ```
     GRANT REMOTE DBA
     TO userid, IDENTIFIED BY password
     ```

   - 以具有 REMOTE DBA 权限的用户身份进行连接。
   - 启动位于 *remote* 目录中的远程数据库。例如，运行：

     ```
     dbeng11 MyModel_remote.db
     ```

   - 启动 dbmlsync、SQL Anywhere MobiLink 客户端。运行 *remote* 目录中以 *_dbmlsync.bat* 结尾的文件。必须将连接信息包括在内。例如，运行：

     ```
     MyModel_dbmlsync.bat "uid=dba;pwd=sql;eng=MyModel_remote"
     ```

   对于 UltraLite 远程数据库：

   - 要测试同步，请运行 *remote* 目录中以 *_ulsync.bat* 结尾的文件。
   - 或者，运行 UltraLite 应用程序。

6. 如果设置服务器启动的同步，则需要执行第一次同步，然后启动监听器。要创建远程 ID 文件，必须进行第一次同步。要启动监听器，请运行 *remote* 目录中以 *_dblsn.bat* 结尾的文件。例如，运行：

   ```
   MyModel_dblsn.bat
   ```

另请参见

- “GRANT REMOTE DBA 语句 [MobiLink] [SQL Remote]”一节 《SQL Anywhere 服务器 - SQL 参考》
- “dbmlsync 权限”一节 《MobiLink - 客户端管理》

研究 MobiLink 的 CustDB 示例

目录

MobiLink CustDB 教程简介

CustDB 是一个销售状态应用程序。CustDB 示例是适合于 MobiLink 开发人员的宝贵资源。它提供了如何实现开发 MobiLink 应用程序所需的多种技术的若干示例。

此应用程序旨在说明几种常用的同步技术。为了从本章学到更多知识，请在阅读本章的同时学习该示例应用程序。

对每种支持的操作系统和数据库类型都提供了一种版本的 CustDB。

有关 CustDB 的位置和设置说明的信息，请参见“建立 CustDB 统一数据库”一节第 48 页。

方案

CustDB 方案如下。

统一数据库位于总部。以下数据存储在统一数据库中：

- 保存同步元数据的 MobiLink 系统表，其中包括实现同步逻辑的同步脚本。
- 包括所有客户、产品和订单信息在内的 CustDB 数据，这些数据存储在基表的行中。

有两种类型的远程数据库：移动经理和移动销售代表。

每个移动销售代表的数据库中包含所有的产品，但仅包含分配给该销售代表的订单；而移动经理的数据库中包含所有的产品和订单。

同步设计

CustDB 示例应用程序中的同步设计将使用以下功能：

- **完整的表下载** ULProduct 表中的所有行和列都完全与远程数据库共享。
- **列子集** ULCustomer 表中所有行（但不是所有列）都与远程数据库共享。
- **行子集** 不同的远程用户可从 ULOrder 表中获得不同的行集合。

  有关行子集的详细信息，请参见“在远程数据库之间对行进行分区”一节 《MobiLink - 服务器管理》。
- **基于时间戳的同步** 这是一种用于识别自上次设备同步以来对统一数据库所作更改的方法。ULCustomer 和 ULOrder 表将采用基于时间戳的方法进行同步。

  请参见“基于时间戳的下载”一节 《MobiLink - 服务器管理》。
- **快照同步** 这是一种简单的同步方法，可在每次同步中下载所有行。ULProduct 表使用这种方法进行同步。

  请参见“快照同步”一节 《MobiLink - 服务器管理》。
- **通过主键池保持主键唯一** 确保主键值在整个 MobiLink 安装中的唯一性至关重要。在此应用程序中采用的主键池方法是一种确保主键唯一性的方法。

  请参见“使用主键池”一节 《MobiLink - 服务器管理》。

  有关确保主键唯一性的其它方法，请参见“维护唯一主键”一节 《MobiLink - 服务器管理》。

有关 CustDB 表的实体关系图，请参见“关于 CustDB 示例数据库”一节 《SQL Anywhere 11 - 简介》。

进一步阅读

- “UltraLite CustDB 示例” 《UltraLite - 数据库管理和参考》

CustDB 安装

本节将介绍组成 CustDB 示例应用程序及数据库的代码片段。其中包括：

- 示例 SQL 脚本，位于 *samples-dir\MobiLink\CustDB* 目录中。
- 应用程序代码，位于 *samples-dir\UltraLite\CustDB* 目录中。
- 特定于平台的用户界面代码，位于 *samples-dir\UltraLite\CustDB* 目录下分别以每个操作系统命名的子目录中。

注意
有关 *samples-dir* 的详细信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。

建立 CustDB 统一数据库

CustDB 统一数据库可以是 MobiLink 支持的任何统一数据库。

SQL Anywhere CustDB

SQL Anywhere CustDB 统一数据库在 *samples-dir\UltraLite\CustDB\custdb.db* 中提供。名为 SQL Anywhere 11 CustDB 的 DSN 会随您的安装提供。

可以使用文件 *samples-dir\UltraLite\CustDB\newdb.bat* 重建此数据库。

如果您要研究创建 CustDB 示例的方法，可查看文件 *samples-dir\MobiLink\CustDB\syncsa.sql*。

其它 RDBMS 的 CustDB

在 *samples-dir\MobiLink\CustDB* 目录中提供的以下 SQL 脚本用来构建 CustDB 统一数据库，作为上述任意一种受支持的 RDBMS：

RDBMS	Custdb 安装脚本
Adaptive Server Enterprise	*custase.sql*
SQL Server	*custmss.sql*
Oracle	*custora.sql*
DB2 LUW	*custdb2.sql*
MySQL	*custmys.sql*

以下过程为每个支持的 RDBMS 创建 CustDB 统一数据库。

有关准备数据库以用作统一数据库的详细信息，请参见“建立统一数据库”一节 《MobiLink - 服务器管理》。

♦ 建立统一数据库（Adaptive Server Enterprise、DB2 LUW、MySQL、Oracle、SQL Server）

1. 在 RDBMS 中创建数据库。
2. 运行位于 SQL Anywhere 11 安装目录的 *MobiLink\setup* 子目录中的以下 SQL 脚本之一，即可添加 MobiLink 系统表：
   - 若为 Adaptive Server Enterprise 统一数据库，运行 *syncase.sql*。
   - 若为 MySQL 统一数据库，运行 *syncmys.sql*。
   - 若为 Oracle 统一数据库，运行 *syncora.sql*。
   - 若为 SQL Server 统一数据库，运行 *syncmss.sql*。
3. 运行位于 *samples-dir\MobiLink\CustDB* 的以下 SQL 脚本之一，即可在 CustDB 数据库中添加示例用户表：
   - 若为 Adaptive Server Enterprise 统一数据库，运行 *custase.sql*。
   - 若为 MySQL 统一数据库，运行 *custmys.sql*。
   - 若为 Oracle 统一数据库，运行 *custora.sql*。
   - 若为 SQL Server 统一数据库，运行 *custmss.sql*。
4. 创建一个引用客户端计算机上的数据库且名为 CustDB 的 ODBC 数据源。
   a. 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**ODBC 管理器**]。
   b. 单击 [**添加**]。
   c. 从列表中选择相应的驱动程序。
      单击 [**完成**]。
   d. 将 ODBC 数据源命名为 CustDB。
   e. 单击 [**登录**] 选项卡。输入数据库的 [**用户 ID**] 和 [**口令**]。

♦ 建立统一数据库 (DB2 LUW)

1. 在 DB2 LUW 服务器上创建统一数据库。对于本教程，称其为 CustDB。
2. 确保缺省表空间（通常称为 USERSPACE1）使用 8 KB 页。
   如果缺省表空间不使用 8 KB 页，请完成以下步骤：
   a. 验证至少有一个缓冲池有 8 KB 页。否则，创建一个具有 8 KB 页的缓冲池。
   b. 创建一个具有 8 KB 页的新表空间和临时表空间。
      有关详细信息，请参见 DB2 LUW 文档。
3. 使用文件 *MobiLink\setup\syncdb2.sql* 将 MobiLink 系统表添加到 DB2 LUW 统一数据库中：
   a. 在文件 *syncdb2.sql* 的顶部更改连接命令。用数据库的名称（或其别名）替换 *DB2Database*。在此示例中，该数据库称为 CustDB。还可以按以下方式添加 DB2 用户名和口令：

      ```
      connect to CustDB user userid using password ~
      ```

   b. 在服务器或客户机上，打开一个 DB2 LUW 命令窗口。键入以下命令运行 *syncdb2.sql*：

```
db2 -c -ec -td~ +s -v -f syncdb2.sql
```

4. 将位于 *samples-dir\MobiLink\CustDB* 中的 *custdb2.class* 复制到您的 DB2 LUW 服务器上的 *SQLLIB\FUNCTION* 目录中。此类包含用于 CustDB 示例的过程。
5. 将数据表添加到 CustDB 数据库中：
   a. 如有必要，在 custdb2.sql 中更改连接命令。例如，可以按下面的方法添加用户名和口令。用您的用户名和口令替换 *userid* 和 *password*。

      ```
      connect to CustDB user userid using password
      ```

   b. 在服务器或客户机上，打开一个 DB2 命令窗口。
   c. 键入以下命令运行 custdb2.sql：

      ```
      db2 -c -ec -td~ +s -v -f custdb2.sql
      ```

   d. 处理完毕后，输入如下命令关闭命令窗口：

      ```
      exit
      ```

6. 创建一个名为 CustDB 的 ODBC 数据源，该数据源将引用 DB2 LUW 客户端上的 DB2 LUW 数据库。
   a. 启动 ODBC 管理器：

      选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**ODBC 管理器**]。

      即会出现 [**ODBC 数据源管理器**]。

   b. 在 [**用户 DSN**] 选项卡上，单击 [**添加**]。
   c. 在 [**创建新数据源**] 窗口中，为您的 DB2 LUW 数据库选择 ODBC 驱动程序。例如，选择 IBM DB2 UDB ODBC 驱动程序。单击 [**完成**]。

      有关如何配置 ODBC 驱动程序的信息，请参见：

      - DB2 LUW 文档
      - 推荐用于 MobiLink 的 ODBC 驱动程序

7. 按以下方式在 DB2 LUW 客户端上运行 *custdb2setuplong* Java 应用程序。该应用程序会重置数据库中的 CustDB 示例。初始设置完成后，可随时键入同一命令行来运行此应用程序，以重置 CustDB 数据库。
   - 如果使用 CustDB 之外的名称来作为数据源，则必须修改 *custdb2setuplong.java* 中的连接代码并按以下方式重新编译它。如果系统变量 *%db2tempdir%* 指定的路径中包含空格，则必须用引号将路径括起来。

     ```
     javac -g -classpath %db2tempdir%\java\jdk\lib\classes.zip;
     %db2tempdir%\java\db2java.zip;
     %db2tempdir%\java\runtime.zip custdb2setuplong.java
     ```

   - 键入以下命令行，其中 *userid* 和 *password* 是用于连接到 CustDB ODBC 数据源的用户名和口令。

     ```
     java custdb2setuplong userid password
     ```

另请参见

- “IBM DB2 LUW 统一数据库”一节 《MobiLink - 服务器管理》

建立 UltraLite 远程数据库

以下过程用于为 CustDB 创建远程数据库。CustDB 远程数据库必须为 UltraLite 数据库。

远程数据库的应用程序逻辑位于 *samples-dir\UltraLite\CustDB* 目录中。它包括以下文件：

- **嵌入式 SQL 逻辑** 文件 *custdb.sqc* 中包含在 UltraLite 数据库中查询和修改信息时所需的 SQL 语句，以及启动与统一数据库的同步时所需的调用。
- **C++ API 逻辑** 文件 *custdbcomp.cpp* 包含 C++ API 逻辑。
- **用户界面功能** 这些功能分别存储在特定于平台的 *Samples\UltraLite\CustDB* 子目录中。

通过完成以下步骤，可以将示例应用程序安装到运行 UltraLite 的远程设备上：

♦ 将示例应用程序安装到远程设备上

1. 启动统一数据库。
2. 启动 MobiLink 服务器。
3. 将示例应用程序安装到远程设备上。
4. 在远程设备上启动示例应用程序。
5. 同步示例应用程序。

示例

以下示例在运行 DB2 统一数据库的 Palm 设备上安装 CustDB 示例。

1. 确保统一数据库正在运行：

   对于 DB2 LUW 数据库，打开 DB2 命令窗口。键入以下命令，其中 *userid* 和 *password* 是用于连接 DB2 LUW 数据库的用户 ID 和口令：

   ```
   db2 connect to CustDB user userid using password
   ```

2. 启动 MobiLink 服务器：

   对于 DB2 LUW 数据库，在命令提示符下运行以下命令：

   ```
   mlsrv11 -c "DSN=CustDB" -zp
   ```

3. 将示例应用程序安装到 Palm 设备上：
   a. 在您的 PC 上启动 Palm Desktop。
   b. 在 [Palm Desktop] 工具栏上单击 [**快速安装**]。
   c. 单击 [**添加**]。浏览至位于 SQL Anywhere 11 安装的 *UltraLite\palm\68k* 子目录中的 *custdb.prc*。
   d. 单击 [**打开**]。
   e. 对 Palm 设备执行 HotSync。
4. 在 Palm 设备上启动 CustDB 示例应用程序：

a. 将 Palm 设备放入其底座中。

在第一次启动示例应用程序时，会提示您进行同步，以便下载数据的初始副本。仅当第一次启动该应用程序时才需要此步骤。在此之后，下载的数据就存储在 UltraLite 数据库中。

b. 启动示例应用程序。

在 [**应用程序**] 视图中，单击 [CustDB]。

c. 在提示符处键入雇员 ID。

如果作为教程学习，请输入值 50。示例应用程序还允许输入值 51、52 或 53，但在这些情况下，其行为略有不同。

有关每个用户 ID 的行为的详细信息，请参见“CustDB 示例中的用户”一节第 56 页。

即会出现一个窗口，通知您在继续操作前必须进行同步。

d. 同步应用程序。

使用 HotSync 获取数据的初始副本。

e. 确认数据已同步到应用程序中。

在 [**应用程序**] 视图中，单击 [CustDB] 应用程序。屏幕将显示一个含有条目的客户登记表。

5. 在远程应用程序和统一数据库之间执行同步。仅当完成对数据库的更改时才需执行此步骤。

a. 确保统一数据库和 MobiLink 服务器都在运行。

b. 将 Palm 设备放入其底座中。

c. 按 [HotSync] 按钮进行同步。

CustDB 数据库中的表

CustDB 数据库的表定义位于 *samples-dir\MobiLink\CustDB* 目录的特定于平台的文件中。（有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）

有关 CustDB 表的实体关系图，请参见“关于 CustDB 示例数据库”一节 《SQL Anywhere 11 - 简介》。

统一数据库和远程数据库都包含以下五个表，不过这些表在两个数据库中的定义稍有不同。

ULCustomer

ULCustomer 表包含一个客户列表。

在远程数据库中，ULCustomer 表包含下面几列：

- **cust_id** 保存用于标识客户的唯一整数的主键列。
- **cust_name** 长度为 30 个字符的字符串，其中包含客户名称。

在统一数据库中，ULCustomer 还包含以下附加列：

- **last_modified** 包含上次修改该行的时间的时间戳。在进行基于时间戳的同步时将用到此列。

ULProduct

ULProduct 表包含一个产品列表。

在远程数据库和统一数据库中，ULProduct 表包含以下几列：

- **prod_id** 包含用于标识产品的唯一整数的主键列。
- **price** 标识单价的整数。
- **prod_name** 长度为 30 个字符的字符串，其中包含产品名称。

ULOrder

ULOrder 表包含一个订单列表，包括有关下订单的客户、接收订单的雇员以及所订购产品的详细信息。

在远程数据库中，ULOrder 表包含下面几列：

- **order_id** 保存用于标识订单的唯一整数的主键列。
- **cust_id** 引用 ULCustomer 的外键列。
- **prod_id** 引用 ULProduct 的外键列。
- **emp_id** 引用 ULEmployee 的外键列。
- **disc** 包含订单折扣的整数。
- **quant** 包含产品定购数量的整数。
- **notes** 长度为 50 个字符的字符串，包含订单的注释。

- **status** 长度为 20 个字符的字符串，用以描述订单的状态。

在统一数据库中，ULOrder 还包含以下附加列：

- **last_modified** 包含上次修改该行的时间的时间戳。在进行基于时间戳的同步时将用到此列。

ULOrderIDPool

ULOrderIDPool 表是 ULOrder 的主键池。

在远程数据库中，ULOrderIDPool 包含下面几列：

- **pool_order_id** 保存用于标识订单 ID 的唯一整数的主键列。

在统一数据库中，ULOrderIDPool 还包含以下附加列：

- **pool_emp_id** 一个整数列，包含订单 ID 被指派到的远程数据库的所有者的雇员 ID。
- **last_modified** 包含上次修改该行的时间的时间戳。

ULCustomerIDPool

ULCustomerIDPool 表是 ULCustomer 的主键池。

在远程数据库中，ULCustomerIDPool 包含下面几列：

- **pool_cust_id** 保存用于标识客户 ID 的唯一整数的主键列。

在统一数据库中，ULCustomerIDPool 还包含以下附加列：

- **pool_emp_id** 一个整数列，包含用于在远程数据库生成的新雇员的雇员 ID。
- **last_modified** 包含上次修改该行的时间的时间戳。

下面的表仅包含在统一数据库中：

ULIdentifyEmployee_nosync

ULIdentifyEmployee_nosync 表仅存在于统一数据库中。它包含如下所示的一列：

- **emp_id** 该主键列包含一个代表雇员 ID 的整数。

ULEmployee

ULEmployee 表仅存在于统一数据库中。它包含一个销售雇员列表。

ULEmployee 表包含下面几列：

- **emp_id** 保存用于标识雇员的唯一整数的主键列。
- **emp_name** 长度为 30 个字符的字符串，其中包含雇员名称。

ULEmpCust

ULEmpCust 表控制将下载哪些客户的订单。如果雇员需要新客户的订单，则插入雇员 ID 和客户 ID 将会强制下载该客户的订单。

- **emp_id** ULEmployee.emp_id 的外键。
- **cust_id** ULCustomer.cust_id 的外键。主键包括 emp_id 和 cust_id。
- **action** 一个字符，用于确定是否应该从远程数据库中删除雇员记录。如果雇员不再需要客户的订单，则设置为 D（删除）。如果仍然需要订单，则设置为空。

  这种情况下必须使用逻辑删除，这样，统一数据库能够识别从 ULOrder 表中删除哪些行。一旦下载完删除后，也可以从统一数据库中删除该雇员的 action 字段为 D 的所有记录。
- **last_modified** 包含上次修改该行的时间的时间戳。在进行基于时间戳的同步时将用到此列。

ULOldOrder 和 ULNewOrder

这些表仅存在于统一数据库中。它们用于解决冲突，所含的列与 ULOrder 相同。在 SQL Anywhere 和 Microsoft SQL Server 中，它们是临时表。在 Adaptive Server Enterprise 中，它们是普通表和 @@spid。DB2 LUW 和 Oracle 没有临时表，因此 MobiLink 需要能够识别哪些行属于同步用户。由于这些表都是基表，因此如果有五个用户正在同步，则他们每个人都同时在这些表中拥有一行。

有关 @@spid 的详细信息，请参见“变量”一节 《SQL Anywhere 服务器 - SQL 参考》。

CustDB 示例中的用户

在 CustDB 示例中有两种类型的用户，销售人员和移动经理。二者的不同之处如下：

- **销售人员** 用户 ID 51、52 和 53 标识与销售人员相关的远程数据库。销售人员可以执行以下任务：
  - 查看客户列表和产品列表。
  - 添加新客户。
  - 添加或删除订单。
  - 滚动浏览未完成订单的列表。
  - 将所做更改同步到统一数据库。
- **移动经理** 用户 ID 50 标识与移动经理相关的远程数据库。移动经理可以执行与销售人员相同的任务。此外，移动经理还可以执行以下任务：
  - 接受或拒绝订单。

同步 CustDB

以下部分说明 CustDB 示例的同步逻辑。

同步逻辑源代码

可以使用 Sybase Central 检查统一数据库中的同步脚本。

脚本类型与事件

通过调用 ml_add_connection_script 或 ml_add_table_script，*custdb.sql* 文件会将每个同步脚本都添加到统一数据库中。

示例

custdb.sql 中的以下代码行将为 ULProduct 表添加一个表级别的脚本，该脚本将在发生 download_cursor 事件期间执行。该脚本包含一个 SELECT 语句。

```
call ml_add_table_script(
'CustDB 11.0',
'ULProduct', 'download_cursor',
'SELECT prod_id, price, prod_name FROM ULProduct' )
go
```

同步 CustDB 示例中的订单

业务规则

ULOrder 表的业务规则如下：

- 只下载未批准或状态为空的订单。
- 在统一数据库和远程数据库中都可以修改订单。
- 每个远程数据库仅包含指派给雇员的订单。

下载

可以在统一数据库中插入、删除，或更新订单。对应于这些操作的脚本如下所示：

- **download_cursor** download_cursor 脚本中的第一个参数是上次下载的时间戳。它用于确保仅下载那些自上次同步以来在远程数据库或统一数据库中进行过修改的行。第二个参数是雇员 ID。它用于确定下载哪些行。

  CustDB 的 download_cursor 脚本如下所示：

  ```
  CALL ULOrderDownload( {ml s.last_table_download}, {ml s.username} )
  ```

  CustDB 的 ULOrderDownload 过程如下所示：

  ```
  CREATE PROCEDURE ULOrderDownload ( IN LastDownload timestamp, IN
  EmployeeID integer )
  BEGIN`
  ```

```
  SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant,
o.notes, o.status
    FROM ULOrder o, ULEmpCust ec
    WHERE o.cust_id = ec.cust_id
    AND ec.emp_id = EmployeeID
    AND ( o.last_modified >= LastDownload
    OR ec.last_modified >= LastDownload)
    AND ( o.status IS NULL  OR  o.status != 'Approved' )
    AND ( ec.action IS NULL )
END
```

- **download_delete_cursor** CustDB 的 download_delete_cursor 脚本如下所示：

```
SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant,
o.notes, o.status
    FROM ULOrder o, dba.ULEmpCust ec
  WHERE o.cust_id = ec.cust_id
    AND ( ( o.status = ''Approved'' AND o.last_modified >= {ml
s.last_table_download} )
    OR ( ec.action = ''D''  )  )
    AND ec.emp_id = {ml s.username}
```

上载

可以在远程数据库中插入、删除或更新订单。对应于这些操作的脚本如下所示：

- **upload_insert** CustDB 的 upload_insert 脚本如下所示：

```
INSERT INTO ULOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

- **upload_update** CustDB 的 upload_update 脚本如下所示：

```
UPDATE ULOrder
  SET cust_id = {ml r.cust_id},
      prod_id = {ml r.prod_id},
      emp_id = {ml r.emp_id},
      disc = {ml r.disc},
      quant = {ml r.quant},
      notes = {ml r.notes},
      status = {ml r.status}
    WHERE order_id = {ml r.order_id}
```

- **upload_delete** CustDB 的 upload_delete 脚本如下所示：

```
DELETE FROM ULOrder WHERE order_id = {ml r.order_id}
```

- **upload_fetch** CustDB 的 upload_fetch 脚本如下所示：

```
SELECT order_id, cust_id, prod_id, emp_id, disc, quant, notes, status
   FROM ULOrder WHERE order_id = {ml r.order_id}
```

- **upload_old_row_insert** CustDB 的 upload_old_row_insert 脚本如下所示：

```
INSERT INTO ULOldOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

- **upload_new_row_insert** CustDB 的 upload_new_row_insert 脚本如下所示：


```
INSERT INTO ULNewOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

冲突解决

- **resolve_conflict** CustDB 的 resolve_conflict 脚本如下所示：

  ```
  CALL ULResolveOrderConflict
  ```

  CustDB 的 ULResolveOrderConflict 过程如下所示：

  ```
  CREATE PROCEDURE ULResolveOrderConflict()
  BEGIN
    -- approval overrides denial
    IF 'Approved' = (SELECT status FROM ULNewOrder) THEN
      UPDATE ULOrder o
      SET o.status = n.status, o.notes = n.notes
      FROM ULNewOrder n
      WHERE o.order_id = n.order_id;
    END IF;
    DELETE FROM ULOldOrder;
    DELETE FROM ULNewOrder;
  END
  ```

同步 CustDB 示例中的客户

业务规则

用于管理客户的业务规则如下：

- 在统一数据库和远程数据库中都可以修改客户信息。
- 远程数据库和统一数据库中都包含一个完整的客户列表。

下载

可以在统一数据库中插入或更新客户信息。对应于这些操作的脚本如下所示：

- **download_cursor** 以下 download_cursor 脚本下载自用户上次下载信息以来其信息发生更改的所有客户。

  ```
  SELECT cust_id, cust_name FROM ULCustomer WHERE last_modified >= {ml
  s.last_table_download}
  ```

上载

您可以在远程数据库中插入、更新或删除客户信息。对应于这些操作的脚本如下所示：

- **upload_insert** CustDB 的 upload_insert 脚本如下所示：

  ```
  INSERT INTO ULCustomer( cust_id, cust_name )
     VALUES( {ml r.cust_id, r.cust_name } )
  ```

- **upload_update** CustDB 的 upload_update 脚本如下所示：

```
UPDATE ULCustomer SET cust_name = {ml r.cust_name}
   WHERE cust_id = {ml r.cust_id}
```

不对此表执行冲突检测。

- **upload_delete** CustDB 的 upload_delete 脚本如下所示：

```
DELETE FROM ULCustomer WHERE cust_id = {ml r.cust_id}
```

同步 CustDB 示例中的产品

业务规则

为 ULProduct 下载所有行—这称为快照同步。

请参见“快照同步”一节 《MobiLink - 服务器管理》。

ULProduct 表的业务规则如下：

- 只能在统一数据库中修改产品。
- 每个远程数据库中都包含所有产品。

下载

可以在统一数据库中插入、删除或更新产品信息。对应于这些操作的脚本如下所示：

- **download_cursor** 以下 download_cursor 脚本用于在每次同步时下载 ULProduct 表的所有行和列：

```
SELECT prod_id, price, prod_name FROM ULProduct
```


维护客户和订单的主键池

CustDB 示例数据库使用主键池来维护 ULCustomer 和 ULOrder 表中的唯一主键。主键池是 ULCustomerIDPool 和 ULOrderIDPool 表。

ULCustomerIDPool

以下脚本在 ULCustomerIDPool 表中定义：

下载

- **download_cursor**

```
SELECT pool_cust_id FROM ULCustomerIDPool
   WHERE last_modified >= {ml s.last_table_download}
     AND pool_emp_id = {ml s.username}
```

上载

- **upload_insert** CustDB 的 upload_insert 脚本如下所示：

```
INSERT INTO ULCustomerIDPool ( pool_cust_id )
   VALUES( {ml r.pool_cust_id} )
```

- **upload_delete** CustDB 的 upload_delete 脚本如下所示：

```
DELETE FROM ULCustomerIDPool
   WHERE pool_cust_id = {ml r.pool_cust_id}
```

- **end_upload** 以下 end_upload 脚本用于确保每次上载后客户 ID 池中保留 20 个客户 ID：

```
CALL ULOrderIDPool_maintain( {ml s.username} )
```

CustDB 的 UL_CustomerIDPool_maintain 过程如下所示：

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
  DECLARE pool_count INTEGER;
  -- Determine how many ids to add to the pool
  SELECT COUNT(*) INTO pool_count
    FROM ULCustomerIDPool
    WHERE pool_emp_id = syncuser_id;
  -- Top up the pool with new ids
  WHILE pool_count < 20 LOOP
    INSERT INTO ULCustomerIDPool ( pool_emp_id )
      VALUES ( syncuser_id );
    SET pool_count = pool_count + 1;
  END LOOP;
END
```

ULOrderIDPool

以下脚本在 ULOrderIDPool 表中定义：

下载

- **download_cursor** CustDB 的 download_cursor 脚本如下所示：

```
SELECT pool_order_id FROM ULOrderIDPool
  WHERE last_modified >= {ml s.last_table_download}
    AND pool_emp_id = {ml s.username}
```

上载

- **end_upload** 以下 end_upload 脚本用于确保每次上载后订单 ID 池中保留 20 个订单 ID。

```
CALL ULOrderIDPool_maintain( {ml s.username} )
```

CustDB 的 UL_OrderIDPool_maintain 过程如下所示：

```
ALTER PROCEDURE ULOrderIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
  DECLARE pool_count INTEGER;
  -- Determine how many ids to add to the pool
  SELECT COUNT(*) INTO pool_count
    FROM ULOrderIDPool
    WHERE pool_emp_id = syncuser_id;
  -- Top up the pool with new ids
  WHILE pool_count < 20 LOOP
    INSERT INTO ULOrderIDPool ( pool_emp_id )
      VALUES ( syncuser_id );
    SET pool_count = pool_count + 1;
  END LOOP;
END
```

- **upload_insert** CustDB 的 upload_insert 脚本如下所示：

```
INSERT INTO ULOrderIDPool ( pool_order_id )
   VALUES( {ml r.pool_order_id}
```

- **upload_delete** CustDB 的 upload_delete 脚本如下所示：

```
DELETE FROM ULOrderIDPool
   WHERE pool_order_id = {ml r.pool_order_id}
```

清除

要重新启动示例，请重置 CustDB 数据库中的数据。

♦ **重置 CustDB 数据库中的数据**

1. 在您的设备上安装 ULDBUtil：
   a. 对于 Palm 设备，在您的 PC 上启动 Palm Desktop。
   b. 在 [Palm Desktop] 工具栏上单击 [**安装**]。
   c. 单击 [**添加**]。浏览至位于 SQL Anywhere 11 安装的 *UltraLite\palm\68k* 子目录中的 *uldbutil.prc*。
   d. 单击 [**完成**]。
   e. 对 Palm 设备执行 HotSync。
2. 使用 ULDBUtil 删除数据：
   a. 对于 Palm 设备，单击 [ULDBUtil] 图标。
   b. 选择 [CustDB] 并单击 [**删除数据**]。
   c. 对 Palm 设备执行 HotSync。

进一步阅读

有关脚本类型的详细信息，请参见“脚本类型”一节 《MobiLink - 服务器管理》。

有关参考资料（包括每个脚本及其参数的详细信息），请参见“同步事件” 《MobiLink - 服务器管理》。

研究 MobiLink Contact 示例

目录

Contact 示例教程简介

Contact 示例是 MobiLink 开发人员的宝贵资源。它以示例的形式说明如何实现开发 MobiLink 应用程序所需的多种技术。

Contact 示例应用程序中包含一个 SQL Anywhere 统一数据库和两个 SQL Anywhere 远程数据库。它阐释了几种常用的同步技术。为了从本章学到更多知识，请在阅读本章的同时学习该示例应用程序。

尽管统一数据库属于 SQL Anywhere 数据库，但同步脚本由 SQL 语句组成，而且这些语句应仅需做少量更改即可应用于其它数据库管理系统。

Contact 示例位于 *samples-dir\MobiLink\Contact* 目录中。有关概述，请参见同一位置的自述文件。（有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。）

同步设计

Contact 示例应用程序中的同步设计使用以下功能：

- **列子集** 统一数据库 Customer、Product、SalesRep 和 Contact 表中各列的子集与远程数据库共享。
- **行子集** 统一数据库中 SalesRep 表的所有列（但只有一行）与每个远程数据库共享。请参见“在远程数据库之间对行进行分区”一节 《MobiLink - 服务器管理》。
- **基于时间戳的同步** 这是一种用于识别自上次设备同步以来对统一数据库所作更改的方法。Customer、Contact 和 Product 表采用基于时间戳的方法进行同步。请参见“基于时间戳的下载”一节 《MobiLink - 服务器管理》。

Contact 示例安装

您可以使用名为 *build.bat* 的 Windows 批处理文件来构建 Contact 示例数据库。在 Unix 系统上，该文件是 *build.sh*。您最好检查该批处理文件的内容。它执行以下操作：

- 为统一数据库和每个远程数据库创建 ODBC 数据源定义。
- 创建一个名为 *consol.db* 的统一数据库，并将 MobiLink 系统表、数据库模式、一些数据、同步脚本及 MobiLink 用户名装载到该数据库中。
- 在名为 *remote_1* 和 *remote_2* 的子目录下各创建一个远程数据库，都命名为 *remote.db*。装载两个数据库的共同信息，并应用自定义设置。这些自定义设置包括一个全局数据库标识符、一个 MobiLink 用户名及两个发布的预订。

♦ 构建 Contact 示例

1. 在命令提示符下，转到 *samples-dir\MobiLink\Contact*。
2. 运行 *build.bat* (Windows) 或 *build.sh* (Unix)。

有关 *samples-dir* 的信息，请参见“示例目录”一节 《SQL Anywhere 服务器 - 数据库管理》。

运行 Contact 示例

Contact 示例中包含用于执行初始同步和演示 MobiLink 服务器及 dbmlsync 命令行的批处理文件。可以在文本编辑器中检查 *samples-dir\MobiLink\Contact* 中以下批处理文件的内容：

- *step1.bat*
- *step2.bat*
- *step3.bat*

♦ 运行 Contact 示例

1. 启动 MobiLink 服务器。
   a. 在命令提示符下，转到 *samples-dir\MobiLink\Contact*。
   b. 运行以下命令：

      ```
      step1
      ```

      该命令用于运行以详细模式启动 MobiLink 服务器的批处理文件。这种模式在开发或故障排除过程中非常有用，但由于其对性能影响很大，因此在日常生产环境中一般不采用。
2. 同步两个远程数据库。
   a. 在命令提示符下，转到 *samples-dir\MobiLink\Contact*。
   b. 运行以下命令：

      ```
      step2
      ```

这是用于同步两个远程数据库的批处理文件。

3. 关闭 MobiLink 服务器。
   a. 在命令提示符下，转到 *samples-dir\MobiLink\Contact*。
   b. 运行以下命令：

      ```
      step3
      ```

      这是用于关闭 MobiLink 服务器的批处理文件。

要了解 Contact 示例中的同步方式，可以使用 Interactive SQL 修改远程数据库和统一数据库中的数据，并使用批处理文件进行同步。

Contact 数据库中的表

Contact 数据库的表定义位于以下文件中（所有文件均位于示例目录下）：

- *MobiLink\Contact\build_consol.sql*
- *MobiLink\Contact\build_remote.sql*

统一数据库和远程数据库都包含以下三个表，不过这些表在两个数据库中的定义稍有不同。

SalesRep

每个销售代表由 SalesRep 表中的一行来表示。每个远程数据库属于一个销售代表。

在每个远程数据库中，SalesRep 包含下面几列：

- **rep_id** 包含销售代表标识号的主键列。
- **name** 销售代表的姓名。

在统一数据库中（远程数据库中没有），还有一个用于保存销售代表的 MobiLink 用户名的 ml_username 列。

Customer

此表为每个客户保存一行信息。每个客户即是一个与某一销售代表有业务关系的公司。在表 SalesRep 和 Customer 之间存在一对多的关系。

在每个远程数据库中，Customer 包含下面几列：

- **cust_id** 用于保存客户标识号的主键列。
- **name** 客户名。此名称为公司名称。
- **rep_id** 引用 SalesRep 表的外键列。标识指派给该客户的销售代表。

在统一数据库中，有两个附加列：last_modified 和 active：

- **last_modified** 行的最近一次修改时间。在进行基于时间戳的同步时将用到此列。
- **active** 用于说明客户当前状态（处于活动状态 (1) 或不再与公司有业务往来 (0)）的 BIT 列。如果此列标记为非活动状态 (0)，则所有与此客户相关的行将从远程数据库中删除。

Contact

此表为每个联系人保存一行信息。联系人是客户公司的员工。表 Customer 和 Contact 之间存在一对多的关系。

在每个远程数据库中，Contact 包含下面几列：

- **contact_id** 用于保存联系人标识号的主键列。
- **name** 各联系人的姓名。
- **cust_id** 联系人所属客户的标识符。

在统一数据库中，该表还包含以下列：

- **last_modified** 行的最近一次修改时间。在进行基于时间戳的同步时将用到此列。
- **active** 用于说明联系人当前状态（处于活动状态 (1) 或不再与公司有业务往来 (0)）的 BIT 列。如果此列标记为非活动状态 (0)，则与此联系人相关的行将从远程数据库中删除。

Product

在 Product 表中，公司所销售的每种产品占一行。Product 表在单独的发布中保存，因此远程数据库可以单独对该表进行同步。

在每个远程数据库中，Product 包含下面几列：

- **id** 包含产品标识号的主键列。
- **name** 产品的名称。
- **size** 产品的大小。
- **quantity** 产品的库存数量。在销售代表获得订单时，此列会被更新。
- **unit_price** 产品的单价。

在统一数据库中，Product 表还包含以下附加列：

- **supplier** 产品的制造商。
- **last_modified** 行的最近一次修改时间。在进行基于时间戳的同步时将用到此列。
- **active** 用于说明产品当前是否处于活动状态 (1) 的 BIT 列。如果此列标记为非活动状态 (0)，则与此产品相关的行将从远程数据库中删除。

除上述这些表以外，在统一数据库中还将创建一组表。其中包括 product_conflict 表（在解决冲突时使用的临时表）和一组属于用户 mlmaint 的用于监控 MobiLink 活动的表。用来创建 MobiLink 监控表的脚本位于 *samples-dir\MobiLink\Contact\mlmaint.sql* 文件中。

Contact 示例中的用户

Contact 示例中包含多个不同的数据库用户 ID 和 MobiLink 用户名。

数据库用户 ID

两个远程数据库已分别指派给销售代表 Samuel Singer (rep_id 856) 和 Pamela Savarino (rep_id 949)。

在连接其远程数据库时，两个用户都使用缺省 SQL Anywhere 用户 ID **dba** 及口令 **SQL**。

每个远程数据库中还有一个用户 ID **sync_user**，其口令为 **sync_user**。此用户 ID 只在 dbmlsync 命令行中使用。**sync_user** 拥有 REMOTE DBA 权限，从 dbmlsync 连接时可以执行任何操作，但是从其它应用程序连接时则没有任何权限。所以使用 **sync_user** ID 和口令应该没有问题。

在统一数据库中，有一个名为 **mlmaint** 的用户，该用户拥有用于监控 MobiLink 同步统计数据及错误的表。但此 **mlmaint** 用户没有进行连接的权限。这种将表指派给单独用户 ID 的做法仅仅是为了在模式中将某些对象与其它对象分开，以便在 Sybase Central 及其它实用程序中进行管理。

MobiLink 用户名

MobiLink 用户名与数据库用户 ID 不同。除了在连接数据库时所使用的用户 ID 以外，每个远程设备还拥有一个 MobiLink 用户名。Samuel Singer 的 MobiLink 用户名为 SSinger。Pamela Savarino 的 MobiLink 用户名为 PSavarino。MobiLink 用户名在以下位置储存或使用：

- 在远程数据库中，使用 CREATE SYNCHRONIZATION USER 语句添加 MobiLink 用户名。
- 在统一数据库中，使用 mluser 实用程序添加 MobiLink 用户名及口令。
- 在同步过程中，正在连接的用户的 MobiLink 口令在 *MobiLink\Contact\step2.bat* 所列 dbmlsync 命令行中提供。
- MobiLink 服务器在同步过程中将 MobiLink 用户名作为参数提供给许多脚本。
- 统一数据库的 SalesRep 表中包含一个 ml_username 列。同步脚本会将 MobiLink 用户名参数与此列中的值相比较。

同步 Contact 示例

下面的部分说明 Contact 示例的同步逻辑。

同步 Contact 示例中的销售代表

SalesRep 表的同步脚本演示了何为**快照同步**。销售代表的信息无论是否发生更改都要进行下载。

请参见“快照同步”一节 《MobiLink - 服务器管理》。

业务规则

SalesRep 表的业务规则如下：

- 不能在远程数据库中修改该表。
- 不能更改销售代表的 MobiLink 用户名及 rep_id 值。
- 每个远程数据库只包含 SalesRep 表中与远程数据库所有者的 MobiLink 用户名相对应的一行。

下载

- **download_cursor** 在每个远程数据库中，SalesRep 表包含单独一行。下载单独一行需要的开销非常小，因此使用一个简单的快照 download_cursor 脚本：

```
SELECT rep_id, name
FROM SalesRep
WHERE ? IS NOT NULL
AND ml_username = ?
```

脚本中的第一个参数为最后一次下载时间戳（此处未使用）。IS NOT NULL 表达式是一个虚表达式，用以使用该参数。第二个参数为 MobiLink 用户名。

上载

此表不应在远程数据库中更新，所以没有针对此表的上载脚本。

同步 Contact 示例中的客户

Customer 表的同步脚本演示了**基于时间戳的同步**及行的分区。这两种技术既维护了表数据的一致性，又最大程度减小了同步期间所传送的数据量。

请参见：

- “基于时间戳的下载”一节 《MobiLink - 服务器管理》
- “在远程数据库之间对行进行分区”一节 《MobiLink - 服务器管理》

业务规则

用于管理客户的业务规则如下：

- 在远程数据库和统一数据库中都可以修改客户信息。
- 客户将定期在销售代表之间重新指派。此过程通常称为地域调整。
- 每个远程数据库中仅包含所指派的客户。

下载

- **download_cursor** 以下 download_cursor 脚本只下载自上次成功下载以来信息已经发生更改的活动客户。它还可以按销售代表过滤客户。

```
SELECT cust_id, Customer.name, Customer.rep_id
FROM Customer key join SalesRep
WHERE Customer.last_modified >= ?
AND SalesRep.ml_username = ?
AND Customer.active = 1
```

- **download_delete_cursor** 以下 download_delete_cursor 脚本只下载自上次成功下载以来信息已经发生更改的客户。它会删除标记为非活动和未指派给销售代表的所有客户。

```
SELECT cust_id
FROM Customer key join SalesRep
WHERE Customer.last_modified >= ?
AND ( SalesRep.ml_username != ? OR Customer.active = 0 )
```

  如果从统一数据库 Customer 表中删除了某些行，它们将不会出现在此结果集中，因此也不会从远程数据库中删除。但这些客户会被标记为非活动状态。

  进行地域调整后，此脚本将删除不再指派给该销售代表的那些客户。它还会删除转让给其他销售代表的客户。这些附加删除将被标志为 SQLCODE 100，但并不妨碍同步过程。可以编写更复杂的脚本来只识别那些从当前销售代表转让出去的客户。

  MobiLink 客户端将在远程数据库上执行级联删除，因此该脚本还将删除所有为指派给其他销售代表的客户工作的联系人。

上载

您可以在远程数据库中插入、更新或删除客户信息。与这些操作对应的脚本如下：

- **upload_insert** 以下 upload_insert 脚本将在 Customer 表中添加一行，并将客户标记为活动：

```
INSERT INTO Customer(
 cust_id, name, rep_id, active )
VALUES ( ?, ?, ?, 1 )
```

- **upload_update** 以下 upload_update 脚本将在统一数据库中修改客户信息。不对此表执行冲突检测。

```
UPDATE Customer
SET name = ?, rep_id = ?
WHERE cust_id = ?
```

- **upload_delete** 以下 upload_delete 脚本在统一数据库中将客户标记为非活动。但它不删除行。

```
UPDATE Customer
SET active = 0
WHERE cust_id = ?
```

同步 Contact 示例中的联系人

Contact 表中包含在客户公司工作的员工的姓名、指向该客户的外键和一个用于标识该联系人的唯一整型值。它还包含一个 last_modified 时间戳和用于指示该联系人是否为活动状态的标记。

业务规则

此表的业务规则如下：

- 在统一数据库和远程数据库中都可以修改联系人信息。
- 每个远程数据库只包含为所指派客户工作的联系人。
- 在销售代表间重新指派客户时，联系人也必须重新指派。

触发器

Customer 表中的触发器用于确保在客户信息发生更改时联系人也执行相应的更改。在客户发生更改时，触发器将显式地更改其每个联系人的 last_modified 列：

```
CREATE TRIGGER UpdateCustomerForContact
AFTER UPDATE OF rep_id ORDER 1
ON DBA.Customer
REFERENCING OLD AS old_cust NEW as new_cust
FOR EACH ROW
BEGIN
  UPDATE Contact
  SET Contact.last_modified = new_cust.last_modified
  FROM Contact
  WHERE Contact.cust_id = new_cust.cust_id
END
```

触发器在客户发生修改时会更新所有联系人记录，使客户与其相关联的联系人结合在一起。只要客户发生修改，所有关联联系人也会随之修改，且客户和关联联系人会在下一次同步过程中一同下载。

下载

- **download_cursor** Contact 的 download_cursor 脚本如下所示：

  ```
  SELECT contact_id, contact.name, contact.cust_id
  FROM ( contact JOIN customer ) JOIN salesrep
  ON contact.cust_id = customer.cust_id
   AND customer.rep_id = salesrep.rep_id
  WHERE Contact.last_modified >= ?
   AND salesrep.ml_username = ?
   AND Contact.active = 1
  ```

  此脚本将检索指派给此代表的、自销售代表上次下载（显式下载或因相应客户的修改而引发的下载）以来发生了更改的所有活动联系人。表 Customer 和 SalesRep 之间需要有一个连接来识别与该销售代表相关联的联系人。

- **download_delete_cursor** Contact 的 download_delete_cursor 脚本如下所示：

  ```
  SELECT contact_id
  FROM ( Contact JOIN Customer ) JOIN SalesRep
  ON Contact.cust_id = Customer.cust_id
   AND Customer.rep_id = SalesRep.rep_id
  ```

```
WHERE Contact.last_modified >= ?
 AND Contact.active = 0
```

由于 MobiLink 客户端将自动使用级联参照完整性，因此在从远程数据库中删除客户时，相应的联系人也将删除。所以 download_delete_cursor 脚本必须仅删除标记为非活动的那些联系人。

上载

可以在远程数据库中插入、更新或删除联系人信息。与这些操作对应的脚本如下：

- **upload_insert** 以下 upload_insert 脚本将在 Contact 表中添加一行，并将联系人标记为活动：

```
INSERT INTO Contact (
 contact_id, name, cust_id, active )
VALUES ( ?, ?, ?, 1 )
```

- **upload_update** 以下 upload_update 脚本将在统一数据库中修改联系人信息：

```
UPDATE Contact
SET name = ?, cust_id = ?
WHERE contact_id = ?
```

不对此表执行冲突检测。

- **upload_delete** 以下 upload_delete 脚本在统一数据库中将联系人标记为非活动。但它不删除行。

```
UPDATE Contact
SET active = 0
WHERE contact_id = ?
```

同步 Contact 示例中的产品

Product 表的脚本演示了冲突检测及其解决办法。

Product 表与其它表分别保存在不同的发布中，因此它可以单独下载。例如：如果价格发生变化而销售代表正在通过速度缓慢的链接进行同步，他们无需上载自己客户和联系人的更改即可下载相关产品的更改。

业务规则

远程数据库中可以进行的唯一更改是在获取订单时更改数量列。

下载

- **download_cursor** 以下 download_cursor 脚本将下载自从上次远程数据库同步后发生更改的所有行：

```
SELECT id, name, size, quantity, unit_price
FROM product
WHERE last_modified >= ?
AND active = 1
```

- **download_delete_cursor** 以下 download_delete_cursor 脚本删除公司停止销售的所有产品。这些产品在统一数据库中被标记为非活动状态。

```
SELECT id, name, size, quantity, unit_price
FROM product
WHERE last_modified >= ?
AND active = 0
```

上载

从远程数据库只上载 UPDATE 操作。这些上载脚本的主要功能是冲突检测与解决过程。

如果两个销售代表获取订单并执行同步，则每个订单都将从 Product 表的数量列中减去。例如，如果 Samuel Singer 获得了一个 20 套棒球帽（产品 ID 400）的订单，他会将数量从 90 更改为 70；如果 Pamela Savarino 在接收到此更改之前获得了一个 10 套棒球帽的订单，她会将自己数据库中的数量列从 90 更改为 80。

因此在 Samuel Singer 同步他的更改时，统一数据库中的数量列由 90 变为 70，而当 Pamela Savarino 同步她的更改时，正确的操作应该是将值设置为 60。而这一设置就是由冲突检测完成的。

冲突检测模式中包含以下脚本：

- **upload_update** 以下 upload_update 脚本是统一数据库中的一个简单的 UPDATE：

  ```
  UPDATE product
  SET name = ?, size = ?, quantity = ?, unit_price = ?
  WHERE product.id = ?
  ```

- **upload_fetch** 以下 upload_fetch 脚本从 Product 表中读取一个单独的行，并与已上载行的旧值进行比较。如果这两行的值不同，则说明检测到冲突。

  ```
  SELECT id, name, size, quantity, unit_price
  FROM Product
  WHERE id = ?
  ```

- **upload_old_row_insert** 如果检测到冲突，则将旧值放入 product_conflict 表中，以供 resolve_conflict 脚本使用。添加该行时将在 row_type 列写入 O（表示原值）。

  ```
  INSERT INTO DBA.product_conflict(
   id, name, size, quantity, unit_price, row_type )
  VALUES( ?, ?, ?, ?, ?, 'O' )' )
  ```

- **upload_new_row_insert** 以下脚本将在 product_conflict 表中添加已上载的行的新值，以供 resolve_conflict 脚本使用：

  ```
  INSERT INTO DBA.product_conflict(
   id, name, size, quantity, unit_price, row_type )
  VALUES( ?, ?, ?, ?, ?, 'N' )
  ```

冲突解决

- **resolve_conflict** 以下脚本通过将旧行和新行之间的差加到统一数据库的数量值中来解决冲突：

  ```
  UPDATE Product
  SET p.quantity = p.quantity
                 - old_row.quantity
                 + new_row.quantity
  FROM Product p,
       DBA.product_conflict old_row,
       DBA.product_conflict new_row
  WHERE p.id = old_row.id
      AND p.id = new_row.id
  ```



```
AND old_row.row_type = 'O'
AND new_row.row_type = 'N'
```

在 Contact 示例中监控统计信息和错误

Contact 示例中包含一些简单的错误报告及监控脚本。用来创建这些脚本的 SQL 语句位于 *MobiLink \Contact\mlmaint.sql* 文件中。

这些脚本将行插入用于保存这些值的表中。为了方便起见，这些表属于特别的用户 mlmaint。

MobiLink 教程

本节提供的教程介绍如何设置和使用 MobiLink 技术。其涵盖范围包括面向新用户的介绍性教程到如何使用高级功能的说明。

教程：MobiLink 简介

目录

MobiLink 教程简介

此教程说明如何开发 MobiLink 应用程序，包括建立统一数据库、创建 ODBC 数据源和远程数据库以及配置数据库对象来调整应用程序。

必需的软件

- SQL Anywhere 11

能力和经验

基本计算机技能。

目标

您将了解：

- MobiLink 的基本概念
- 如何开发 MobiLink 应用程序

建议阅读的背景知识

有关 MobiLink 的介绍，请参见：

- “了解 MobiLink 同步”第 3 页
- “MobiLink 统一数据库” 《MobiLink - 服务器管理》
- “MobiLink 模型”第 23 页
- “同步技术” 《MobiLink - 服务器管理》

第 1 课：确保主键的唯一性

本教程使用尚未设置 MobiLink 同步的示例数据库 Demo。它的表都有主键，但主键仅在 Demo 数据库中是唯一的。由于 Demo 即将成为同步系统中的统一数据库，因此其主键在整个系统（包括所有远程数据库）中必须唯一。

第 2 课：运行 [创建同步模型向导]

♦ **运行 [创建同步模型向导]**

1. 在 Sybase Central 的左窗格中，右击 [**MobiLink**] » [**新建**] » [**同步模型**]
2. 完成以下字段：
   a. 在 [**您要给新同步模型指定什么名称**] 字段中键入 **MLDemo**。
   b. 在 [**您希望新同步模型文件使用哪一个文件夹**] 字段中键入 *C:\temp*。
   c. 单击 [**下一步**]。
3. 将三个复选框全部选中。单击 [**下一步**]。
4. 单击 [**选择统一数据库**]。
5. 指定统一数据库的连接参数：
   a. 单击 [**ODBC 数据源名称**]。
   b. 在 [**ODBC 数据源名称**] 列表中，单击 [**SQL Anywhere 11 Demo**]。
   c. 单击 [**确定**]。
   d. 如果出现提示询问您是否想要安装系统设置，则单击 [**是**]。
6. 单击 [**下一步**]。
7. 单击 [**否，新建一个远程数据库模式**]。单击 [**下一步**]。
8. 单击 [**全选**]。单击 [**下一步**]。
9. 在 [**下载类型**]、[**时间戳下载选项**]、[**下载删除**] 和 [**下载子集**] 页面上单击 [**下一步**] 以接受缺省设置。
10. 单击 [**基于行的冲突检测**]。单击 [**下一步**]。
11. 单击 [**统一数据库优先**]。单击 [**下一步**]。
12. 单击 [**下一步**]。
13. 单击 [**完成**]。

教程：编写脚本和监控同步

目录

简介

目标

本教程的目的是使您胜任和熟悉以下任务：

- 将统一数据库模式迁移到远程数据库。
- 使用 Sybase Central 或 Interactive SQL 编写同步需要的基本脚本并将其存储在统一数据库中。
- 编写冲突检测和解决脚本。
- 使用日志文件和 MobiLink 监控器监控同步。

建议阅读的背景知识

有关 MobiLink 体系结构的详细信息，请参见“了解 MobiLink 同步”第 3 页。

有关同步脚本的详细信息，请参见“同步脚本介绍”一节 《MobiLink - 服务器管理》。

有关 Interactive SQL 的详细信息，请参见“使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关 Sybase Central 的详细信息，请参见“使用 Sybase Central”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关同步的简介，请参见“教程：MobiLink 简介”第 81 页。

有关 MobiLink 冲突解决的详细信息，请参见“冲突处理”一节 《MobiLink - 服务器管理》。

有关 MobiLink 监控器的详细信息，请参见“MobiLink 监控器” 《MobiLink - 服务器管理》。

第 1 课：建立 SQL Anywhere 统一数据库

本课指导您通过以下步骤建立 SQL Anywhere 统一数据库：

1. 创建统一数据库和模式。
2. 运行 MobiLink 安装脚本。
3. 为统一数据库定义 ODBC 数据源。

创建统一数据库

在此过程中，您将使用 Sybase Central 的 [创建数据库向导] 来创建统一数据库。

♦ 创建 SQL Anywhere 数据库

1. 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 选择 [**工具**] » [**SQL Anywhere 11**] » [**创建数据库**]。
3. 在 [**欢迎**] 页面上，单击 [**下一步**]。
4. 单击 [**下一步**]。
5. 在 [**将主数据库文件保存到以下文件**] 字段中键入 *C:\MLmon\cons.db*。单击 [**下一步**]。
6. 请按照 [**创建数据库向导**] 中的说明进行操作并接受缺省值。
7. 单击 [**完成**]。

生成统一数据库模式

统一数据库模式包括 Product 表，其中存储硬件产品的名称和数量。

♦ 将 Product 表添加到统一模式

1. 在 Sybase Central 的左窗格中，右击 [**cons - DBA**]，然后选择 [**文件**] » [**打开 Interactive SQL**]。
2. 安装 Product 表。
   - 在 Interactive SQL 中执行以下命令。

```
/* the Product table */
create table Product (
    name    varchar(128) not null primary key,
    quantity      integer,
    last_modified timestamp default timestamp
)
go

insert into Product(name, quantity)
  values ( 'Screwmaster Drill', 10);

insert into Product(name, quantity)
  values ( 'Drywall Screws 10lb', 30);

insert into Product(name, quantity)
  values ( 'Putty Knife x25', 12);
```

```
go
```

3. 安装用于冲突解决的临时表。

   在“第 6 课：创建冲突检测和解决脚本”一节第 97 页中,您将编写用于在发生冲突时将值插入这些表的脚本。

```
/* the Product_old table */
create table Product_old (
    name     varchar(128) not null primary key,
    quantity      integer,
    last_modified timestamp default timestamp
)
go

/* the Product_new table */
create table Product_new (
    name     varchar(128) not null primary key,
    quantity      integer,
    last_modified timestamp default timestamp
)
go
```

4. 检验每个表是否都成功创建。

   - 例如，在 Interactive SQL 中执行以下命令来检验 Product 表的内容。

```
select * from Product
```

5. 运行 MobiLink 安装脚本来添加 MobiLink 需要的系统对象。

   - 在 [**SQL 语句**] 窗格中，键入：

```
read "C:\Program Files\SQL Anywhere 11\MobiLink\setup\syncsa.sql"
```

为统一数据库定义 ODBC 数据源。

使用 SQL Anywhere 11 驱动程序为 cons 数据库定义 ODBC 数据源。

♦ **为统一数据库定义 ODBC 数据源**

1. 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**ODBC 管理器**]。
2. 单击 [**用户 DSN**] 选项卡，然后单击 [**添加**]。
3. 在 [**名称**] 列表中，单击 [**SQL Anywhere 11**]。单击 [**完成**]。
4. 在 [**SQL Anywhere 11 的 ODBC 配置**] 窗口中，进行以下操作：
   a. 单击 [**ODBC**] 选项卡。
   b. 在 [**数据源名**] 字段中，键入 **sa_cons**。
   c. 单击 [**登录**] 选项卡。
   d. 在 [**用户 ID**] 字段键入 **DBA**。
   e. 在 [**口令**] 字段中键入 **sql**。
   f. 单击 [**数据库**] 选项卡。

g. 在 [**服务器名**] 字段中键入 **cons**。

h. 在 [**数据库文件**] 字段中键入 *C:\MLmon\cons.db*。

i. 单击 [**确定**]。

5. 单击 [**确定**]。

进一步阅读

有关使用 dbinit 命令行实用程序创建统一数据库的信息，请参见“初始化实用程序 (dbinit)”一节《SQL Anywhere 服务器 - 数据库管理》或“第 3 课：为同步创建脚本”一节第 93 页。

有关统一数据库（包括非 SQL Anywhere RDBMS）的详细信息，请参见“MobiLink 统一数据库”《MobiLink - 服务器管理》。

有关 Interactive SQL 的详细信息，请参见“使用 Interactive SQL”一节 《SQL Anywhere 服务器 - 数据库管理》。

第 2 课：建立远程 SQL Anywhere 数据库

MobiLink 用于进行涉及统一数据库服务器和多个移动数据库的同步。在本节中，您将创建两个远程数据库。对于每个数据库，您将：

- 迁移统一模式的选定部分。
- 创建同步发布、用户和预订。

创建 SQL Anywhere 数据库

在第 1 课中，使用了 Sybase Central 创建数据库。在本教程中，您将使用命令行实用程序。这两个工具产生的结果完全相同。

♦ 创建和启动 SQL Anywhere 远程数据库

1. 在命令提示符处，浏览到要创建远程数据库的目录。
2. 键入以下命令创建数据库：

   ```
   dbinit -p 4096 remote1.db
   ```

   对于 remote2，键入：

   ```
   dbinit -p 4096 remote2.db
   ```

   -p 选项定义了 4K 页面大小，该大小被证实可以提高许多环境下的性能。

   有关 dbinit 选项的详细信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。
3. 现在，要启动数据库，请键入：

   ```
   dbeng11 remote1.db
   ```

   对于 remote2，键入：

   ```
   dbeng11 remote2.db
   ```

迁移统一数据库模式的子集

迁移统一数据库模式的子集涉及到以下任务：

- 连接到远程数据库。
- 创建远程服务器和外部登录。
- 使用 Sybase Central 的 [迁移数据库向导]。

♦ 将统一数据库模式的子集迁移到 remote1

1. 单击 [开始] » [程序] » [SQL Anywhere 11] » [Sybase Central]。
2. 连接到远程数据库：
   a. 在左窗格中，单击 [SQL Anywhere 11]。

b. 单击 [**文件**] » [**连接**]。

c. 单击 [**标识**] 选项卡。

d. 在 [**用户 ID**] 字段键入 **DBA**。

e. 在 [**口令**] 字段中键入 **sql**。

f. 单击 [**数据库**] 选项卡。

g. 在 [**服务器名**] 字段中键入 **remote1**。

h. 单击 [**确定**]。

3. 创建远程服务器：

a. 在左窗格中，右击 [**远程服务器**] 并选择 [**新建**] » [**远程服务器**]。

b. 在 [**您要给新远程服务器指定什么名称**] 字段中键入 **my_sa**。单击 [**下一步**]。

c. 单击 [**SQL Anywhere**]。单击 [**下一步**]。

d. 在 [**连接信息是什么**] 字段中键入 **sa_cons**。单击 [**下一步**]。

e. 单击 [**下一步**]。

f. 单击 [**为当前用户创建外部登录**]。

g. 在 [**登录名**] 字段中键入 **DBA**。

h. 在 [**口令**] 字段中键入 **sql**。

i. 在 [**确认口令**] 字段中键入 **sql**。

j. 单击 [**完成**]。

4. 迁移统一数据库模式：

a. 单击 [**工具**] » [**SQL Anywhere 11**] » [**迁移数据库**]。

b. 单击 [**下一步**]。

c. 在 [**要迁移到哪个数据库**] 列表中，选择 **remote1**。单击 [**下一步**]。

d. 在 [**要从哪个远程服务器迁移**] 列表中，选择 **my_sa**。单击 [**下一步**]。

e. 在 [可用表] 列表中，选择 [**Product**]，然后单击 [**添加**]。单击 [**下一步**]。

f. 单击 [**DBA**]。单击 [**下一步**]。

g. 清除 [**迁移数据**]。单击 [**完成**]。

h. 单击 [**关闭**]。

5. 使用 **remote2** 数据库重复步骤 2 至 4。

同步预订和发布

发布用于标识远程数据库上要同步的表和列。这些表和列称为 **articles**。同步预订会为 MobiLink 用户预订发布。

同步预订和发布存储在远程数据库中。

♦ **创建远程同步发布、同步用户和同步预订**

1. 在 Sybase Central 的左窗格中，右击 [**remote1 - DBA**]，然后选择 [**文件**] » [**打开 Interactive SQL**]。
2. 输入 remote1 的同步信息：
   - 在 Interactive SQL 中执行以下语句：

```
CREATE PUBLICATION pub_1 (TABLE Product);
CREATE SYNCHRONIZATION USER user_1;
CREATE SYNCHRONIZATION SUBSCRIPTION TO pub_1
 FOR user_1 TYPE TCPIP ADDRESS 'host=localhost'
 OPTION scriptversion='ver1';
```

3. 启动 Interactive SQL 并连接到 remote2。
4. 输入 remote2 的同步信息。
   - 在 Interactive SQL 中执行以下语句：

```
CREATE PUBLICATION pub_2 (TABLE Product);
CREATE SYNCHRONIZATION USER user_2;
CREATE SYNCHRONIZATION SUBSCRIPTION TO pub_2
 FOR user_2 TYPE TCPIP ADDRESS 'host=localhost'
 OPTION scriptversion='ver1';
```

现在您已准备了远程数据库和统一数据库。在下一课中，您将编写同步脚本。在第 4 课中，您将执行同步。

进一步阅读

有关定义发布和预订的详细信息，请参见“发布数据”一节 《MobiLink - 客户端管理》。

第 3 课：为同步创建脚本

可以使用 Sybase Central 查看、编写和修改同步脚本。在本节中，您将编写以下同步脚本：

- **upload_insert** 定义如何将插入到远程数据库的数据应用于统一数据库。
- **download_cursor** 定义应从统一数据库下载哪些数据。

每个脚本都属于指定的脚本版本。在添加脚本之前必须将脚本版本添加到统一数据库。

♦ 添加脚本版本

1. 使用 Sybase Central 的 MobiLink 插件连接到 cons 数据库：
   a. 在 Sybase Central 的左窗格中，选择 [**MobiLink 11**]。
   b. 选择 [**模式**] » [**管理**]。
   c. 选择 [**文件**] » [**连接**]。
   d. 单击 [**标识**] 选项卡。
   e. 单击 [**ODBC 数据源名称**]，然后键入 **sa_cons**。单击 [**确定**]。
2. 添加脚本版本 ver1：
   a. 在左窗格中，右击 [**版本**] 并选择 [**新建**] » [**版本**]。
   b. 在 [**您要给新脚本版本指定什么名称**] 字段中键入 **ver1**。
   c. 单击 [**完成**]。

♦ 将已同步表添加到统一数据库

1. 在左窗格的 [**MobiLink 11**] 列表中，右击 [**同步表**] » [**新建**] » [**同步表**]。
2. 单击 [**在统一数据库中选择与远程表相同名称的表**]。
3. 在 [**哪个用户拥有要同步的表**] 列表中，单击 [**DBA**]。
4. 在 [**您要同步哪个表**] 列表中，单击 [**产品**]。
5. 单击 [**完成**]。
6. 为统一数据库的每个上载和下载都添加一个新表脚本

♦ 添加 Product 表的表脚本

1. 在左窗格的 [**MobiLink 11**] 列表中，展开 [**同步表**]。
2. 右击 [**产品**] 表，然后选择 [**新建**] » [**表脚本**]。
3. 在 [**您要为哪个版本创建表脚本**] 列表中，单击 **ver1**。
4. 在 [**哪个事件应导致执行表脚本**] 列表中，单击 [**upload_insert**]。单击 [**下一步**]。
5. 单击 [**完成**]。

6. 在 Sybase Central 的右窗格中，键入以下 SQL 语句：

```
INSERT INTO Product( name, quantity, last_modified )
  VALUES( ?, ?, ? )
```

upload_insert 事件决定应如何将插入到远程数据库的数据应用于统一数据库。有关 upload_insert 的详细信息，请参见“upload_insert 表事件”一节 《MobiLink - 服务器管理》。

7. 选择 [**文件**] » [**保存**]。

8. 使用以下 SQL 语句对 **download_cursor** 事件重复步骤 1 至 5：

```
SELECT name, quantity, last_modified
  FROM Product where last_modified >= ?
```

download_cursor 脚本定义一个用于选择在远程数据库中下载和插入或更新的统一数据库行的游标。有关 download_cursor 的详细信息，请参见“download_cursor 表事件”一节 《MobiLink - 服务器管理》。

进一步阅读

有关您刚创建的脚本的详细信息，请参见“upload_insert 表事件”一节 《MobiLink - 服务器管理》和“download_cursor 表事件”一节 《MobiLink - 服务器管理》。

有关脚本版本的详细信息，请参见“脚本版本”一节 《MobiLink - 服务器管理》。

有关添加脚本的详细信息，请参见“添加和删除脚本”一节 《MobiLink - 服务器管理》。

有关编写表脚本的详细信息，请参见“表脚本”一节 《MobiLink - 服务器管理》。

有关编写同步脚本的详细信息，请参见“编写同步脚本” 《MobiLink - 服务器管理》。

有关可以编程以自定义同步的事件的完整列表，请参见“同步事件” 《MobiLink - 服务器管理》。

第 4 课：执行 MobiLink 同步

♦ 执行 remote1 和 remote2 的同步

1. 通过运行 MobiLink 服务器实用程序 (mlsrv11) 来启动 MobiLink 服务器：
   - 在命令提示符处，键入以下命令：

     ```
     mlsrv11 -c "dsn=sa_cons" -o mlserver.mls -v+ -dl -zu+ -x tcpip
     ```

   有关 mlsrv11 选项的信息，请参见“MobiLink 服务器选项”《MobiLink - 服务器管理》。

   在 [**MobiLink 服务器**] 窗口中出现一条消息，指明 MobiLink 服务器已准备好处理请求。
2. 通过运行 MobiLink 同步客户端实用程序 (dbmlsync) 来启动同步。
   - 要同步 remote1，请在命令提示符处，以单行键入以下命令：

     ```
     dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -o rem1.txt -v+
     ```

   - 要同步 remote2，请在命令提示符处，以单行键入以下命令：

     ```
     dbmlsync -c "eng=remote2;uid=DBA;pwd=sql" -o rem2.txt -v+
     ```

   有关 dbmlsync 选项的完整列表，请参见“MobiLink SQL Anywhere 客户端实用程序 (dbmlsync)”《MobiLink - 客户端管理》。

   MobiLink 同步客户端启动后，DBMLSync 窗口中出现一条消息，指明 MobiLink 同步已成功完成。

进一步阅读

有关 MobiLink 服务器的详细信息，请参见“MobiLink 服务器”《MobiLink - 服务器管理》。

有关 mlsrv11 选项的完整列表，请参见“MobiLink 服务器选项”《MobiLink - 服务器管理》。

有关 dbmlsync 的详细信息，请参见“SQL Anywhere 客户端”《MobiLink - 客户端管理》。

有关 dbmlsync 选项的完整列表，请参见“MobiLink SQL Anywhere 客户端实用程序 (dbmlsync)”《MobiLink - 客户端管理》。

第 5 课：使用日志文件监控 MobiLink 同步

同步表之后，可以使用通过每个命令行创建的消息日志文件（*mlserver.mls*、*rem1.txt* 和 *rem2.txt*）来查看同步的进度。这些文件的缺省位置是命令运行时所在的目录。

♦ 在 MobiLink 同步日志文件中查找错误

1. 在文本编辑器中打开日志文件。对于本教程，日志文件为 *mlserver.mls*。
2. 在该文件中搜索字符串 [MobiLink 服务器已启动]
3. 向下扫描文件左侧。以 **I.** 开头的行包含信息性消息，以 **E.** 开头的行包含错误消息。
4. 请注意，在本示例的 **E.** 旁有以下文本：

   ```
   04/27 16:01:01. <Main>: Error: Unable to initialize communications stream
   1: tcpip.
   ```

   此消息指明上载和下载前的错误。在同步预订或发布定义中可能存在错误。
5. 查找以如下形式开头的子句：

   ```
   SQL Anywhere Synchronization request from:
   ```

   该子句表示已建立同步请求。
6. 查找开始 [Working on a request] 的子句。这表示客户端和服务器正在通信。如果指定了高详细程度，则可能得到此消息。

♦ 在 MobiLink 同步客户端日志文件中检测错误

1. 在文本编辑器中打开客户端日志文件 *rem1.txt*。
2. 在该文件中搜索字符串 [COMMIT]。如果此字符串出现，则说明同步成功。
3. 在该文件中搜索字符串 [ROLLBACK]。如果事务回退，则存在阻止事务完成的错误。
4. 向下扫描文件左侧。如果看到 **E.**，则存在错误。如果没有任何错误，则说明同步已成功完成。

进一步阅读

有关 MobiLink 服务器日志文件的详细信息，请参见“记录 MobiLink 服务器操作”一节 《MobiLink - 服务器管理》。

第 6 课：创建冲突检测和解决脚本

在向统一数据库上载行时会发生冲突。如果两个用户修改不同远程数据库中相同的行，则当这两行中的第二行到达 MobiLink 服务器时会检测到冲突。使用同步脚本可以检测和解决冲突。

有关 MobiLink 冲突解决的详细信息，请参见“冲突处理”一节 《MobiLink - 服务器管理》。

库存示例

假设在实地有两个销售员的场景。Salesman1 开始时库存有 10 个货品，后来卖出 3 个。他将其远程数据库 remote1 的库存更新为 3 个货品。Salesman2 卖出 4 个货品并将其库存（在 Remote2 上）更新为 6。

当 remote1 使用 MobiLink 同步客户端实用程序进行同步时，统一数据库更新为 7。当 remote2 同步时，将检测到冲突，因为统一数据库中的库存值已更改。

若要以编程方式解决这一冲突，需要三个行值：

1. 统一数据库中的当前值。

   remote1 同步后，统一数据库中的值为 7。

2. Remote2 上载的新行值。

3. Remote2 在上一次同步期间获取的旧行值。

在这种情况下，可使用以下业务逻辑来计算新的库存值并解决冲突：

```
current consolidated - (old remote - new remote)
that is, 7 - (10-6) = 3
```

表达式（旧远程数据库 - 新远程数据库）提供了 Salesman2 卖出的货品数，而非绝对库存值。

用于冲突检测和解决的同步脚本

要检测和解决冲突，添加以下脚本：

- **upload_update** upload_update 事件决定应如何将插入到远程数据库的数据应用于统一数据库。也可以使用 upload_update 的扩展原型检测更新冲突。

  有关使用 upload_update 检测冲突的详细信息，请参见“检测冲突”一节 《MobiLink - 服务器管理》。

  有关 upload_update 的详细信息，请参见“upload_update 表事件”一节 《MobiLink - 服务器管理》。

- **upload_old_row_insert** 使用此脚本处理远程数据库在上一次同步时获取的旧行值。

  有关 upload_old_row_insert 的详细信息，请参见“upload_old_row_insert 表事件”一节 《MobiLink - 服务器管理》。

- **upload_new_row_insert** 使用此脚本处理新行值（远程数据库上的更新值）。

  有关 upload_new_row_insert 的详细信息，请参见“upload_new_row_insert 表事件”一节 《MobiLink - 服务器管理》。

- **resolve_conflict** 解决冲突脚本应用业务逻辑来解决冲突。

有关 resolve_conflict 的详细信息，请参见“resolve_conflict 表事件”一节 《MobiLink - 服务器管理》。

♦ 安装冲突检测和解决脚本

1. 启动 Interactive SQL。
   a. 在命令提示符下，键入 **dbisql**。
   b. 单击 [**标识**] 选项卡。
   c. 在 [**用户 ID**] 字段键入 **DBA**。
   d. 在 [**口令**] 字段中键入 **sql**。
   e. 单击 [**数据库**] 选项卡。
   f. 在 [**服务器名**] 字段中键入 **cons**。
   g. 单击 [**确定**]。
2. 安装冲突检测和解决脚本：

   在 Interactive SQL 中执行以下语句：

```
/* upload_update */
call ml_add_table_script( 'ver1', 'Product',
 'upload_update',
 'UPDATE Product
   SET quantity = ?, last_modified = ?
    WHERE name = ?
    AND quantity=? AND last_modified=?' )
go

/* upload_old_row_insert */
call ml_add_table_script( 'ver1', 'Product',
 'upload_old_row_insert',
 'INSERT INTO Product_old (name,quantity,last_modified)
   values (?,?,?)')
go

/* upload_new_row_insert */
call ml_add_table_script( 'ver1', 'Product',
 'upload_new_row_insert',
 'INSERT INTO Product_new (name,quantity,last_modified)
   values (?,?,?)')
go

/* resolve_conflict */
call ml_add_table_script( 'ver1', 'Product',
 'resolve_conflict',
 'declare @product_name varchar(128);
  declare @old_rem_val integer;
  declare @new_rem_val integer;
  declare @curr_cons_val integer;
  declare @resolved_value integer;

  // obtain the product name
  SELECT name INTO @product_name
     FROM Product_old;

  // obtain the old remote value
```



```
SELECT quantity INTO @old_rem_val
  FROM Product_old;

//obtain the new remote value
SELECT quantity INTO @new_rem_val
  FROM Product_new;

// obtain the current value in cons
SELECT quantity INTO @curr_cons_val
 FROM Product WHERE name = @product_name;

// determine the resolved value
SET @resolved_value =
  @curr_cons_val- (@old_rem_val - @new_rem_val);

// update cons with the resolved value
UPDATE Product
 SET quantity = @resolved_value
 WHERE name = @product_name;

// clear the old and new row tables
delete from Product_new;
delete from Product_old
')
```

SQL Anywhere 统一数据库的设置已完成。

进一步阅读

有关 MobiLink 冲突检测和解决的详细信息，请参见“冲突处理”一节 《MobiLink - 服务器管理》。

有关 MobiLink 统一数据库的详细信息，请参见“MobiLink 统一数据库” 《MobiLink - 服务器管理》。

第 7 课：使用 MobiLink 监控器检测更新冲突

可以使用 MobiLink 监控器收集同步发生时的相关统计信息。MobiLink 监控器的图表在垂直轴上显示任务，在水平轴上显示时间进度。

使用 MobiLink 监控器可以快速识别导致错误或满足某些条件的同步。由于 MobiLink 监控器对性能的影响不大，建议将其同时用于开发和生产。

在本节中，您将：

- 启动和配置 MobiLink 监控器，以可视方式区分涉及更新冲突的同步。
- 通过在 remote1 和 remote2 上更新同一行来生成冲突。
- 使用 MobiLink 监控器检测冲突。

♦ 配置 MobiLink 监控器检测更新冲突

1. 单击 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**MobiLink 监控器**]。
2. 连接到 MobiLink 服务器：
   a. 单击 [**标识**] 选项卡。
   b. 在 [**用户**] 字段中键入 **monitor_user**。使用 -zu+ 选项启动 MobiLink 服务器时，将自动添加此用户。
   c. 在 [**口令**] 字段中，键入口令或将此字段留空。
   d. 单击 [**数据库**] 选项卡。
   e. 在 [**服务器名**] 字段中键入 **cons**。
   f. 单击 [**确定**]。
3. 单击 [**工具**] » [**监视项目管理器**] 以启动 MobiLink 监控器监视项目管理器。
4. 对更新冲突添加新监视项目：
   a. 单击 [**新建**]。
   b. 在 [**名称**] 字段中键入 **conflict_detected**。
   c. 在 [**属性**] 列表中，单击 [**conflicted_updates**]。

      conflicted_updates 统计属性指明被检测到冲突的已上载更新的数目。

      有关 MobiLink 监控器统计属性的详细信息，请参见“MobiLink 统计信息属性”一节《MobiLink - 服务器管理》。
   d. 在 [**操作员**] 列表中，单击 [**is greater than**]。
   e. 在 [**值**] 字段中键入 **0**。
   f. 单击 [**添加**]。
   g. 在 [**图表模式**] 列表中，为 [**图表**] 窗格设置模式。[图表] 窗格是 MobiLink 监控器的中间窗格。

h. 在 [**一览表颜色**] 列表中，为 [**一览表**] 窗格设置颜色。[一览表] 窗格是 MobiLink 监控器的底部窗格。

5. 单击 [**确定**]。
6. 单击 [**确定**]。

♦ 生成更新冲突

1. 更新 remote1 库存值。

   Salesman1 开始时 Screwmaster Drill 库存有 10 个货品，后来卖出 3 个。他将其远程数据库 remote1 的库存更新为 3 个货品。执行更新：

   a. 启动 Interactive SQL 并连接到 remote1（如果尚未连接）。

   b. 在 Interactive SQL 中执行以下语句，将 Screwmaster Drill 库存更新为 7 个货品。

   ```
   UPDATE Product SET quantity = 7
    WHERE name ='Screwmaster Drill'
   COMMIT
   ```

2. 同步 remote1。

   在命令提示符处，键入以下命令启动 MobiLink 同步客户端：

   ```
   dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -v+
   ```

   同步之后，统一数据库 Screwmaster Drill 库存为 7 个货品。

3. 更新 remote2 库存值。

   Salesman2 卖出 4 个货品并将其库存（在 Remote2 上）更新为 6。当 remote2 同步时，将检测到冲突，因为统一数据库中的库存值已更改。执行更新：

   a. 启动 Interactive SQL 并连接到 remote2。

   在命令提示符处，键入以下命令：

   ```
   dbisql
   ```

   b. 单击 [**标识**] 选项卡。

   c. 在 [**用户 ID**] 字段键入 **DBA**。

   d. 在 [**口令**] 字段中键入 **sql**。

   e. 单击 [**数据库**] 选项卡。

   f. 在 [**服务器名**] 字段中键入 **remote2**。

   g. 单击 [**确定**]。

   h. 在 Interactive SQL 中执行以下语句，将 Screwmaster Drill 库存更新为 6 个货品：

   ```
   UPDATE Product SET quantity = 6
    WHERE name ='Screwmaster Drill'
   COMMIT
   ```

4. 同步 remote2。

   - 启动 MobiLink 同步客户端。

在命令提示符处，键入以下命令：

```
dbmlsync -c "eng=remote2;uid=DBA;pwd=sql" -v+
```

现在可以切换到 MobiLink 监控器并查看同步的结果。

♦ 使用 MobiLink 监控器检测更新冲突

1. 暂停图表滚动。

   单击 [**文件**] » [**监控器**] » [**暂停图表滚动**]。

2. 使用 MobiLink 监控器的 [**一览表**] 窗格、[**图表**] 窗格和 [**详细信息**] 表，查看有关同步的统计信息。

   a. 在监控器的 [**一览表**] 窗格（MobiLink 监控器的底部窗格）中找到同步项。生成更新冲突的 remote2 同步以红色显示：

   b. 要在 [**图表**] 窗格中查看 remote2 同步，请单击 [**一览表**] 窗格中的同步对象并在其上方拖动鼠标：

      该同步对象以您为 conflict_detected 监视项目所选的模式显示。

   c. 使用缩放工具查看同步详细信息。

      从 [**视图**] 菜单中选择 [**放大**]。

   d. 要查看同步属性，请双击同步对象或详细信息表中的对应行。选择 [**上载**] 选项卡，查看发生冲突的更新数。

进一步阅读

有关 MobiLink 冲突解决的详细信息，请参见“冲突处理”一节 《MobiLink - 服务器管理》。

有关 MobiLink 监控器的详细信息，请参见“MobiLink 监控器” 《MobiLink - 服务器管理》。

有关 MobiLink 监控器统计属性的详细信息，请参见“MobiLink 统计信息属性”一节 《MobiLink - 服务器管理》。

清除

从计算机中删除教程资料。

♦ 从计算机中删除教程资料

1. 关闭以下应用程序的所有实例：
   - MobiLink 监控器
   - Sybase Central
   - Interactive SQL
2. 关闭 SQL Anywhere 窗口、MobiLink 窗口和同步客户端窗口，方法是右击各个任务栏项并选择 [**关闭**]。
3. 删除所有与教程相关的数据源：
   a. 启动 ODBC 管理器。
   b. 从 [**开始**] 菜单中选择 [**程序**] » [**SQL Anywhere 11**] » [**ODBC 管理器**]。
   c. 在 [**用户数据源**] 列表中，选择 [**sa_cons**]，然后单击 [**删除**]。
4. 删除统一数据库和远程数据库。
   a. 导航到统一数据库和远程数据库所在的目录。
   b. 删除 *cons.db*、*cons.log*、*remote1.db*、*remote1.log*、*remote2.db*、*remote2.log*。

进一步阅读

有关运行 MobiLink 服务器的详细信息，请参见“MobiLink 服务器”《MobiLink - 服务器管理》。

有关同步脚本编写的详细信息，请参见“编写同步脚本”《MobiLink - 服务器管理》和“同步事件”《MobiLink - 服务器管理》。

有关其它同步方法（如基于时间戳的同步）的简介，请参见“同步技术”《MobiLink - 服务器管理》。

有关 MobiLink 监控器的详细信息，请参见“MobiLink 监控器”《MobiLink - 服务器管理》。

有关 MobiLink 冲突检测和解决的详细信息，请参见“冲突处理”一节 《MobiLink - 服务器管理》。

教程：将 MobiLink 用于 Oracle 10g 统一数据库

目录

Oracle 教程简介

本教程介绍如何使用 MobiLink 协调 Oracle 10g 数据库。它设置 Oracle 10g 统一数据库和 SQL Anywhere 远程数据库之间的同步。还设置 UltraLite 远程数据库。

本教程的目的是协调与销售团队相关的数据。在这种情况下，每个销售人员均是远程同步客户端。每个销售人员都有一个本地 SQL Anywhere 数据库，使用 MobiLink 与位于总部的公司 Oracle 数据库同步。每个销售人员使用膝上型电脑或手提设备访问公司数据和操纵远程数据库的数据。

必需的软件

本教程假定，在运行本教程的本地计算机上安装了完整的 SQL Anywhere（包括 MobiLink）。本教程使用 Oracle 数据库 10g 发行版本 2 创建。本教程可以用于其它版本 Oracle，但不能保证正常运行。本教程还假设，本地计算机上安装了 Oracle 数据库 10g，而且还可以远程访问。

本教程假设您执行 Oracle 数据库 10g 的基本安装，创建名为 orcl 的 starter 数据库。orcl 数据库有订单输入 (OE) 和人事关系 (HR) 示例模式。此外，还可以手动安装示例模式，或使用 Oracle Database Configuration Assistant 进行安装。有关安装这两个示例模式的详细信息，请参见 http://www.oracle.com/technology/obe/obe1013jdev/common/OBEConnection.htm。

本教程假定，以具有 SYSDBA 权限的 SYS 用户身份连接到 Oracle。这是在第 7 课中授予 V_$TRANSACTION Oracle 系统视图的权限时的一个要求。SYS 用户的口令在安装 Oracle 数据库期间设置。

概述

此教程将介绍如何：

- 在设计远程模式时作出某些重要的判断，如远程表的同步方向。
- 向统一数据库和远程数据库添加唯一主键。
- 创建 ODBC 数据源，将 MobiLink 连接到 Oracle 10g 数据库。
- 使用 [**创建同步模型向导**] 在统一数据库和远程数据库之间建立同步。
- 使用 [模型] 模式自定义同步模型。
- 使用 [**部署同步模型向导**] 部署统一数据库和远程数据库。
- 将远程客户端与统一数据库同步。

建议阅读的背景知识

有关 MobiLink 同步的概述，请参见“了解 MobiLink 同步”第 3 页。

有关 MobiLink 模型的概述，请参见“MobiLink 模型”第 23 页。

第 1 课：设计模式

本教程假设，您已安装订单输入 (OE) 和人事关系 (HR) 示例模式。OE 模式用作统一数据库。它封装有关雇员、订单、客户和产品的信息。对于本教程，您主要对 OE 模式感兴趣。但是，您必须引用 HR 模式中的 EMPLOYEES 表才能获取有关各个销售人员的信息。这里简单介绍 OE 模式中的相关表：

表	说明
CUSTOMERS	其信息记录备案的客户。
INVENTORIES	每个产品在每个仓库中的存储量。
ORDER_ITEMS	每个订单中包括的产品列表。
ORDERS	在特定日期销售人员与客户之间的销售记录。
PRODUCT_DESCRIPTIONS	用不同语言描述每个产品。
PRODUCT_INFORMATION	每个产品在系统中的记录。

设计远程模式

第一步是设计远程模式。每个销售人员都使用统一数据库的完整副本既没有必要，同时效率也十分低。将设计远程模式，以便其中仅包含与特定销售人员相关的信息。为实现此目标，按以下方式设计远程模式：

统一表	远程表
CUSTOMERS	包括所有行。
INVENTORIES	远程表中不包含。
ORDER_ITEMS	按 sales_rep_id 过滤。
ORDERS	包括所有行。
PRODUCT_DESCRIPTIONS	远程表中不包含。
PRODUCT_INFORMATION	包括所有行。

每个销售人员需要保存所有客户和产品的记录，以便可以将任何产品销售给任何客户。本教程假设销售人员说的语言始终与客户相同，所以不需要 PRODUCT_DESCRIPTIONS 表。每个销售人员需要订单相关信息，但不是与其他销售人员相关的订单。此目标可通过按销售人员标识符过滤行来实现。

下一步是选择每个表的同步方向。您应分别考虑远程数据库需要读取、创建、更改或删除的信息。在本示例中，特定销售人员需要产品和客户列表，但从不向系统输入新产品。您可以制定限制，始终从总部的统一数据库向系统输入产品和客户。但是，销售人员需要能够定期记录新订单。这些因素导致关于每个表中同步的以下决定：

表	同步
CUSTOMERS	仅下载到远程表。
ORDER_ITEMS	下载和上载。
ORDER	下载和上载。
PRODUCT_INFORMATION	仅下载到远程表。

第 2 课：准备统一数据库

本教程假定，您已安装订单输入 (OE) 示例数据库。有关安装示例模式的信息，可在 Oracle 文档中找到，也可在 http://www.oracle.com/technology/obe/obe1013jdev/common/OBEConnection.htm 在线查看。

OE 数据库需要更改才能与 MobiLink 一起使用。将删除一些列，因为这些列曾作为用户定义类型创建。可以将这些用户定义类型转换为 SQL Anywhere 识别的类型，但完成此任务与本教程无关。接着，必须授予 OE 用户创建触发器的权限，因为 MobiLink 需要使用 OE 的凭据创建几个触发器。

♦ 准备统一数据库

1. 使用 Oracle SQL Plus 应用程序以具有 SYSDBA 权限的 SYS 用户身份进行连接。在命令提示符处，运行以下命令：

```
sqlplus SYS/your password for sys as SYSDBA
```

2. 要删除作为用户定义类型创建的列，运行以下命令：

```
ALTER TABLE OE.CUSTOMERS DROP COLUMN CUST_ADDRESS;
ALTER TABLE OE.CUSTOMERS DROP COLUMN PHONE_NUMBERS;
ALTER TABLE OE.CUSTOMERS DROP COLUMN CUST_GEO_LOCATION;
ALTER TABLE OE.PRODUCT_INFORMATION DROP COLUMN WARRANTY_PERIOD;
```

3. 解锁 OE 用户，并将口令设置为 sql。运行以下命令：

```
ALTER USER OE IDENTIFIED BY sql ACCOUNT UNLOCK;
```

4. 要允许 OE 用户创建触发器，运行以下命令：

```
GRANT CREATE ANY TRIGGER TO OE;
```

5. 要删除 orders_customer 外键和创建引用 customers 表中的 customer_id 的新外键，运行以下命令：

```
ALTER TABLE OE.ORDERS DROP CONSTRAINT ORDERS_CUSTOMER_ID_FK;
ALTER TABLE OE.ORDERS ADD CONSTRAINT ORDERS_CUSTOMER_ID_FK
  FOREIGN KEY (CUSTOMER_ID) REFERENCES OE.CUSTOMERS (CUSTOMER_ID);
```

添加唯一主键

在同步系统中，表的主键是在不同数据库中唯一地标识行的唯一方式，也是检测冲突的唯一方式。正在协调的每一个表都必须具有一个主键。切勿对主键进行更新。还需要确保已插入到一个数据库中的主键值不被插入到另一数据库。

可通过多种方法来生成唯一主键。为简便起见，本教程使用了复合主键方法。此方法使用多个在所有统一和远程数据库中唯一的列来生成主键。

♦ 向统一数据库添加唯一主键

1. 使用 Oracle SQL Plus 应用程序以具有 SYSDBA 权限的 SYS 用户身份进行连接。在命令提示符处，运行以下命令：

```
sqlplus SYS/your password for sys as SYSDBA
```

2. 添加到 SALES_REP_ID 的值必须在 HR.EMPLOYEES 表中。ORDERS_SALES_REP_FK 外键实施此规则。为简单起见，通过运行以下命令删除此外键：

```
ALTER TABLE OE.ORDERS
DROP CONSTRAINT ORDERS_SALES_REP_FK;
```

3. SALES_REP_ID 列不能添加为主键，因为它包含空值。就本教程而言，用值 1 替换空值。运行以下命令：

```
UPDATE OE.ORDERS
SET SALES_REP_ID = 1
WHERE SALES_REP_ID IS NULL;
```

4. ORDER_ID 列是 ORDERS 表的当前主键。要删除当前主键，运行以下命令：

```
ALTER TABLE OE.ORDERS
DROP PRIMARY KEY CASCADE;
```

5. 组合主键由 SALES_REP_ID 列和 ORDER_ID 列组成。要添加组合主键，运行以下命令：

```
ALTER TABLE OE.ORDERS
ADD CONSTRAINT salesrep_order_pk PRIMARY KEY (sales_rep_id, order_id);
```

运行了以上命令后，MobiLink 服务器将顺利地连接到统一数据库，并将其设置为可与任意数量的远程表进行同步。

添加外键

所有数据库中的唯一主键

在第 4 课中，将根据统一模式创建远程模式。这意味着远程模式和统一模式具有相同的主键。

列经过特别的选择，以确保对于所有数据库存在唯一主键。对于 ORDERS 表，主键由 SALES_REP_ID 和 ORDER_ID 列组成。任何插入到远程 sales 表的值都必须具有唯一订单号（而 SALES_REP_ID 值始终不变）。这确保每个远程 ORDERS 表的唯一性。统一 ORDERS 表中的主键防止多个销售人员上载数据时发生冲突。由于销售人员的 SALES_REP_ID 值不同，因而来自一个销售人员的每个上载对于另一个销售人员都是唯一的。

进一步阅读

有关兼容 RDBMS 的详细信息，请参见“统一数据库简介”一节 《MobiLink - 服务器管理》。

有关将 Oracle 设置为统一数据库的详细信息，请参见“Oracle 统一数据库”一节 《MobiLink - 服务器管理》。

有关生成唯一主键不同方式的详细信息，请参见“维护唯一主键”一节 《MobiLink - 服务器管理》。

第 3 课：连接 MobiLink

本课将创建一个 ODBC 数据源，用于将 MobiLink 连接到统一数据库。

♦ 将 MobiLink 连接到统一数据库

1. 创建 ODBC 数据源。

   您应使用随 SQL Anywhere 11 提供的用于 Oracle 的 iAnywhere 驱动程序。使用以下配置设置：

ODBC 选项卡字段	值
数据源名	oracle_cons
用户 ID	OE
口令	sql
SID	orcl
过程返回结果	未选。
数组大小	60000

   本教程假设您执行 Oracle 数据库 10g 的基本安装，创建名为 orcl 的 starter 数据库。订单输入 (OE) 模式自动安装在 orcl 上。如果您将 OE 模式安装在另一个数据库上，则使用该数据库的名称作为 SID 值。

2. 要测试 ODBC 连接，单击 [**测试连接**]。

配置 ODBC 数据源后，就可使用 MobiLink 插件连接到 Oracle 数据库并创建同步模型。

进一步阅读

有关推荐用于 MobiLink 的 ODBC 驱动程序的详细信息，请参见推荐用于 MobiLink 的 ODBC 驱动程序。

有关 iAnywhere Solutions Oracle 驱动程序的配置选项的详细信息，请参见“iAnywhere Solutions Oracle 驱动程序”一节 《MobiLink - 服务器管理》 。

第 4 课：创建同步模型

[**创建同步模型向导**] 会引导您逐步完成在统一数据库和远程数据库之间设置同步的过程。

♦ 创建同步模型

1. 启动 Sybase Central。
2. 单击 [**工具**] » [**MobiLink 11**] » [**设置 MobiLink 同步**]。
3. 将新同步模型命名为 **sync_oracle** 并键入新模型的位置。
4. 在 [**主键需求**] 页面上，单击全部三个复选框。
5. 在 [**统一数据库模式**] 页面上，连接到统一数据库：
   a. 单击 [**选择统一数据库**]。
   b. 选择 [**ODBC 数据源名称**]，然后选择 **oracle_cons**。单击 [**确定**]。
   c. 如果这是 MobiLink 第一次使用统一数据库，将出现一条消息，询问您是否安装 MobiLink 系统设置。安装 MobiLink 系统设置会添加 MobiLink 系统表和过程。单击 [**是**]。

      如果出现错误消息，请确保数据源的配置正确。
   d. 将 MobiLink 系统表和过程添加到统一数据库后，页面中将出现名称、用户、产品和版本号。单击 [**下一步**]。

      将装载来自统一数据库模式的表。
6. 创建远程模式：
   a. 在 [**远程数据库模式**] 页面上，选择 [**否，新建一个远程数据库模式**]，然后单击 [**下一步**]。
   b. 在 [**新的远程数据库模式**] 页面上，选中以下表的复选框以将其包括在远程模式中，然后单击 [**下一步**]。
      - CUSTOMERS
      - ORDERS
      - ORDER_ITEMS
      - PRODUCT_INFORMATION
7. 选择并配置下载类型：
   a. 在 [**下载类型**] 页面上，选择 [**基于时间戳的下载**]。

      选择基于时间戳的下载可以使所传送的数据量最小，因为此下载类型仅传输自上次下载后经过更新的数据。
   b. 在 [**时间戳下载选项**] 页面上，选择 [**使用影子表来保存时间戳列**]。

      最好选择使用影子表，因为这样不会对现有的表进行任何更改。
8. 在 [**下载删除**] 页面上，指定将删除记录传播到远程设备的方式：
   a. 如果要指示远程数据库下载删除，请选择 [**是**]。
   b. 选择 [**使用影子表来记录删除**]。

      MobiLink 将在统一数据库中创建影子表，以执行删除。

9. 在 [**下载子集**] 页面上，指定远程数据库仅从统一数据库下载某一数据子集：

   a. 选择 [**是，向每个远程数据库下载相同的数据**]。（在 [模型] 模式一课的步骤 2 中，将指定如何使用自定义逻辑向远程数据库下载特定的数据。）

10. 在 [**上载冲突检测**] 页面上，选择 [**无冲突检测**]。

    许多应用程序都需要进行冲突检测，但本教程未使用任何冲突检测。

11. 在 [**发布、脚本版本和说明**] 页面上，分别键入 **sync_oracle_publication** 和 **sync_oracle_scriptversion** 作为发布和脚本版本。

    发布是远程数据库上的对象，用于指定要同步的数据。MobiLink 服务器脚本定义了如何将远程数据库上载的数据应用到统一数据库，同时也对脚本版本组的脚本进行了定义。可以针对不同的应用程序使用不同的脚本版本，从而使您只需维护单一 MobiLink 服务器就可同时同步多个不同的应用程序。

12. 单击 [**完成**]。

    模型将以 [模型] 模式出现。

模型模式

1. 要指定数据的同步方向，在 [**映射方向**] 列中按以下方式设置方向：

   - ORDERS 和 ORDER_ITEMS 是双向（既能上载又可下载）。
   - 其余的表为仅下载同步。

2. 按远程 ID 过滤下载到远程数据库的行：

   a. 对于 ORDERS 表，将 [**下载子集**] 更改为 [**自定义**]。

   b. 在屏幕底部打开 [**下载子集**] 选项卡。

   c. 远程 ID 可唯一地标识远程数据库。要按远程 ID 过滤行，需要对 download_cursor 脚本的 WHERE 子句添加限制。要执行以上操作，在 [**要在下载游标的 WHERE 子句中使用的 SQL 表达式**] 文本框中键入以下搜索条件：

   ```
   "OE"."ORDERS"."SALES_REP_ID" = {ml s.remote_id}
   ```

   下载游标脚本指定了要从各个表中下载到远程数据库的行和列。搜索条件确保仅下载有关一个销售人员（即标识符等于数据库的远程 ID 的销售代表）的信息。

3. 保存同步模型。

   至此，同步模型创建完成，接下来您可以部署该模型。

进一步阅读

- “建立统一数据库”一节 《MobiLink - 服务器管理》
- “MobiLink 服务器系统表” 《MobiLink - 服务器管理》
- “MobiLink 系统过程”一节 《MobiLink - 服务器管理》
- “编写 download_delete_cursor 脚本”一节 《MobiLink - 服务器管理》
- “冲突处理”一节 《MobiLink - 服务器管理》
- “解决冲突”一节 《MobiLink - 服务器管理》
- “发布数据”一节 《MobiLink - 客户端管理》
- “模型模式”一节第 30 页
- “修改下载类型”一节第 32 页
- “修改冲突检测和冲突解决”一节第 35 页
- “修改表映射和列映射”一节第 30 页

第 5 课：部署同步模型

此过程使用 [**部署同步模型向导**] 部署统一数据库和远程数据库。可以分别部署统一数据库和远程数据库，也可以同时部署两者。

♦ **部署同步模型**

1. 在 Sybase Central 的左窗格中，右击同步模型，然后选择 [**部署**]。
2. 在 [**欢迎**] 页面上，单击 [**为下列一项或多项内容指定部署详细信息**] 并选择 [**统一数据库**]、[**远程数据库和同步客户端**] 和 [**MobiLink 服务器**]。单击 [**下一步**]。
3. 在 [**统一数据库部署目标**] 页面上，执行以下操作：
   a. 选择 [**将更改保存到以下 SQL 文件中**]。在本教程中，请使用缺省位置。
   b. 单击 [**连接到统一数据库以便直接应用更改**] 将更改立即应用于统一数据库。
   c. 单击 [**选择统一数据库**]。
   d. 单击 [**ODBC 数据源名称**]，然后选择 **oracle_cons**。
   e. 单击 [**确定**]。
   f. 单击 [**下一步**]。

   将出现一条消息，询问是否要创建 *consolidated* 目录。单击 [**是**]。
4. 在 [**远程数据库部署**] 页面上，选择 [**新 SQL Anywhere 数据库**]。单击 [**下一步**]。
5. 在 [**新 SQL Anywhere 远程数据库**] 页面上执行以下操作：
   a. 单击 [**建立带有可创建数据库命令的命令文件和 SQL 文件**]。
   b. 在 [**SQL 文件**] 字段中，接受 SQL 文件的缺省位置。
   c. 单击 [**创建远程 SQL Anywhere 数据库**]。
   d. 单击 [**下一步**]。

   将出现一条消息，询问是否要创建 *remote* 目录。单击 [**是**]。
6. 在 [**MobiLink 用户**] 页面上执行以下操作：
   a. 在 [**您要使用什么用户名连接到 MobiLink 服务器**] 字段中键入 **oracle_remote**。
   b. 在 [**您要使用什么口令**] 字段中键入 **oracle_pass**。
   c. 单击 [**下一步**]。
7. 在 [**同步流参数**] 页面中单击 [**TCP/IP**]，并且在 [**端口**] 字段中键入 **2439**。单击 [**下一步**]。
8. 在 [**客户端流参数**] 页面的 [**主机**] 字段中键入 **localhost**。单击 [**下一步**]。
9. 要接受缺省设置，在 [**MobiLink 服务器流参数**] 和 [**MobiLink 服务器的详细程度**] 页面上单击 [**下一步**]。
10. 在 [**MobiLink 服务器的选项**] 页面的 [**您要给 MobiLink 服务器指定什么名称**] 字段中键入 **oracle_mlsrv**。

    将出现一条消息，询问是否要创建 *mlsrv* 目录。单击 [**是**]。

11. 在 [**SQL Anywhere 远程同步客户端的详细程度**] 页面上，选择远程同步客户端的详细程度，并使用远程数据库日志文件的缺省文件名。单击 [**完成**]。

12. 单击 [**关闭**]。

现在，统一数据库已完成配置，可与众多远程客户端进行同步，并且您已成功部署了一个远程客户端。如果要部署其它远程客户端，可再次运行此向导，但此时需要创建新的 MobiLink 用户并跳过对统一数据库和 MobiLink 服务器的部署。由于它们已部署，因此您只需部署其它远程同步客户端。

进一步阅读

- “部署模型”一节第 40 页
- “创建远程数据库”一节 《MobiLink - 客户端管理》
- “MobiLink 用户简介”一节 《MobiLink - 客户端管理》

第 6 课：启动服务器和客户端

在本课中，您将启动 MobiLink 服务器和远程数据库。

在前几课中，您修改了下载游标脚本以下载有关一个销售人员的信息。在本课中，您通过将远程 ID 设置为销售人员标识符来指定销售人员。

启动 MobiLink 服务器

缺省情况下，MobiLink 在上载和下载时使用快照/READ COMMITTED 隔离级别。为使 MobiLink 服务器最有效地使用快照隔离，MobiLink 服务器使用的 Oracle 帐户必须拥有 V_$TRANSACTION Oracle 系统视图的访问权限。如果未授予访问权限，将会发出警告，而且下载时可能会丢失行。

♦ 授予 OE 用户访问权限

1. 使用 Oracle SQL Plus 应用程序以具有 SYSDBA 权限的 SYS 用户身份进行连接。在命令提示符处，运行以下命令：

   ```
   sqlplus SYS/your password for sys as SYSDBA
   ```

2. 要授予 V_$TRANSACTION Oracle 系统视图的访问权限，运行以下命令：

   ```
   GRANT SELECT ON SYS.V_$TRANSACTION TO OE;
   ```

3. 要向 V_$SESSION Oracle 系统视图授予访问权限，请运行以下命令：

   ```
   GRANT SELECT ON SYS.V_$SESSION TO OE;
   ```

♦ 启动 MobiLink 服务器

1. 在命令提示符处，导航到您创建同步模型的目录。（此为您在 [**创建同步模型向导**] 的第一步中选择的根目录。）

   如果您使用建议的目录名称，则以下目录应位于根目录中：*sync_oracle\mlsrv*。

2. 从 mlsrv 目录运行以下命令：

   ```
   sync_oracle_mlsrv.bat "DSN=oracle_cons;UID=OE;PWD=sql;"
   ```

   - **sync_oracle_mlsrv.bat**　用于启动 MobiLink 服务器所创建的命令文件。
   - **DSN**　ODBC 数据源名称。
   - **UID**　用于连接到统一数据库的用户名。
   - **PWD**　用于连接到统一数据库的口令。

   此命令成功运行后，在 [**MobiLink 服务器消息**] 窗口中将显示消息 [`MobiLink 服务器已启动`]。

   如果 MobiLink 服务器启动失败，请检查统一数据库的连接信息。

♦ 启动远程数据库

1. 在命令提示符处，浏览到 [**部署同步模型向导**] 创建的远程数据库所在的文件夹。

   如果您使用建议的目录名称，则以下目录应位于根目录中：*sync_oracle\remote*。

2. 使用以下命令启动远程 SQL Anywhere 数据库：

```
dbeng11 -n remote_eng sync_oracle_remote.db -n remote_db
```

- **dbeng11** 用于启动 SQL Anywhere 数据库的数据库服务器。
- **remote_eng** 数据库服务器名称。
- **sync_oracle_remote.db** 在 remote_eng 上启动的数据库文件。
- **remote_db** remote_eng 上的数据库名称。

上述命令成功运行后，名为 remote_eng 的 SQL Anywhere 数据库服务器将启动，并加载名为 remote_db 的数据库。

设置远程 ID

在远程模式中，每个远程数据库代表一个销售人员。您编写的同步脚本包含了指示 MobiLink 服务器根据远程数据库的远程 ID 下载数据子集的逻辑。必须将数据库的远程 ID 设置为有效的销售人员标识符值。

在第一次同步之前完成此步骤非常重要，因为当远程设备第一次同步时，将下载与所选择的销售人员相关的所有信息。

♦ 将远程 ID 设置为有效的销售人员标识符

1. 选择一个有效的销售人员标识符：
   a. 使用 Oracle SQL Plus 应用程序以具有 SYSDBA 权限的 SYS 用户身份进行连接。在命令提示符处，运行以下命令：

```
sqlplus SYS/your password for sys as SYSDBA
```

   b. 要查看 ORDERS 表中有效销售人员标识符的列表，请执行以下语句：

```
SELECT COUNT( SALES_REP_ID ), SALES_REP_ID
FROM OE.ORDERS GROUP BY SALES_REP_ID;
```

   在本示例中，远程数据库代表 SALES_REP_ID 为 154 的销售人员。

   c. 要退出 Oracle，运行以下命令：

```
exit
```

2. 要将数据库的远程 ID 设置为值 154，运行以下命令：

```
dbisql
-c "ENG=remote_eng;DBN=remote_db;UID=DBA;PWD=sql"
"SET OPTION PUBLIC.ml_remote_id='154'"
```

- **dbisql** 用于对 SQL Anywhere 数据库执行 SQL 命令的应用程序。
- **ENG** 将数据库服务器名指定为 remote_eng。
- **DBN** 将数据库名指定为 remote_db。
- **UID** 用于连接到远程数据库的用户名。
- **PWD** 用于连接到远程数据库的口令。

- **SET OPTION PUBLIC.ml_remote_id='154'** 用于将远程 ID 的值设置为 154 的 SQL 命令。

进一步阅读

- “SQL Anywhere 数据库服务器”一节 《SQL Anywhere 服务器 - 数据库管理》
- “同步已部署的模型”一节第 42 页
- “运行 MobiLink 服务器”一节 《MobiLink - 服务器管理》
- “远程 ID”一节 《MobiLink - 客户端管理》

第 7 课：同步

现在您可以进行第一次远程客户端同步了。可以使用 MobiLink 客户端程序 dbmlsync 来完成这一过程。Dbmlsync 将连接到远程数据库、通过 MobiLink 服务器对自身进行验证，并根据远程数据库中的发布执行要同步远程数据库和统一数据库所必需的所有上载和下载。

♦ 同步远程客户端

- 在命令提示符处，运行以下命令：

```
dbmlsync
-c "ENG=remote_eng;DBN=remote_db;UID=DBA;PWD=sql;"
-n sync_oracle_publication
-u oracle_remote -mp oracle_pass
```

  - **dbmlsync**　同步应用程序。
  - **ENG**　指定远程数据库服务器的名称。
  - **DBN**　指定远程数据库的名称。
  - **UID**　用于连接到远程数据库的用户名。
  - **PWD**　用于连接到远程数据库的口令。
  - **sync_oracle_publication**　远程设备上的发布，此发布用于执行同步。（此发布由 [**创建同步模型向导**] 创建。）
  - **oracle_remote**　使用 MobiLink 服务器进行验证时所用的用户名。
  - **oracle_pass**　使用 MobiLink 服务器进行验证时所用的口令。

同步的进度会出现在 [**SQL Anywhere MobiLink 客户端消息**] 窗口中。上述命令成功运行后，dbmlsync 应用程序将使用统一数据库中的信息子集填充远程数据库。

如果同步失败，则检查传递给 dbmlsync 应用程序的连接信息以及 MobiLink 用户名和口令。 如果没有问题，请检查所使用的发布名，并确保统一数据库和 MobiLink 服务器正在运行。您也可以检查同步日志的内容（服务器和客户端）。

> **注意**
> 如果在与 MobiLink 服务器不同的另一台计算机上运行 dbmlsync 应用程序，则必须传递指定 MobiLink 服务器位置的参数。

查看数据

使用 MobiLink 服务器成功地同步远程客户端和统一数据库后，远程数据现应该包含有关一个销售人员的信息。在 Sybase Central 中，您可以使用 SQL Anywhere 11 插件来验证这一点。

♦ 查看远程数据库中的数据

1. 启动 Sybase Central
2. 连接到远程数据库：
   a. 在左窗格中，右击 [**SQL Anywhere 11**] 并选择 [**连接**]。

b. 键入 **DBA** 作为 [**用户 ID**]，相应的 [**口令**] 为 **sql**。

c. 在 [**数据库**] 选项卡上，键入 **remote_eng** 作为 [**服务器名**]，键入 **remote_db** 作为 [**数据库文件**]。

d. 单击 [**确定**]。

3. 选择 ORDERS 表，然后在右窗格中单击 [**数据**] 选项卡。

   在 ORDERS 表中，所有记录均为标识符为 154 的销售人员的数据。此特定销售人员不关心其他销售人员的销售信息。为此，需要设置同步脚本按照远程 ID 来过滤行，并需要将此数据库的远程 ID 的值设置为特定销售人员的标识符。这样可使该特定销售人员的数据库所占的空间更小，并且同步时间也更少。正是由于远程数据库的大小始终维持在最低水平，某些经常执行的操作（如输入新的销售记录或处理之前销售的退款）才能运行得更快，效率更高。

进一步阅读

- “同步过程”一节第 16 页
- “dbmlsync 语法”一节 《MobiLink - 客户端管理》

清除

重新生成订单输入数据库，然后从计算机上删除所有教程资料。

♦ 重新生成订单输入数据库

- 运行 *oe_main.sql*，删除当前 OE 模式，安装新 OE 模式。*oe_main.sql* 文件位于 *$ORACLE_HOME/demo/schema/order_entry* 中。

也可以使用 Oracle Database Configuration Assistant 用新安装的示例模式创建新数据库。

♦ 删除同步模型

1. 启动 Sybase Central。
2. 在右窗格中，双击 [**MobiLink 11**]。

   sync_oracle 模型将出现在右窗格中。
3. 右击 **sync_oracle** 并单击 [**删除**]。
4. 在 [**确认删除**] 窗口中，单击 [**删除**]。

♦ 消除远程数据库

- 要消除远程数据库，请使用 dberase 实用程序。运行以下命令：

```
dberase sync_oracle\remote\sync_oracle_remote.db
```


进一步阅读

有关运行 MobiLink 服务器的详细信息，请参见“MobiLink 服务器”《MobiLink - 服务器管理》。

有关编写同步脚本的详细信息，请参见“编写同步脚本”《MobiLink - 服务器管理》和“同步事件”《MobiLink - 服务器管理》。

有关其它同步方法（如时间戳）的介绍，请参见“同步技术”《MobiLink - 服务器管理》。

教程：将 MobiLink 与 Adaptive Server Enterprise 统一数据库结合使用

目录

Adaptive Server Enterprise 教程简介

本教程介绍如何使用 MobiLink 协调 Adaptive Server Enterpris 数据库。教程将在 Adaptive Server Enterprise 统一数据库和 SQL Anywhere 远程数据库之间建立同步。也可以使用 UltraLite 客户端。

本教程的目的是协调某一连锁书店的数据。在本方案中，每个书店都是一种远程同步环境。每个书店都有一个本地 SQL Anywhere 数据库，该数据库与位于总部的 Adaptive Server Enterprise 数据库同步。各个书店都可以拥有数台计算机，用于访问和操作来自远程数据库的数据。

必需的软件

本教程假定您运行教程的本地计算机上已完整安装了 SQL Anywhere（包括 MobiLink）。本教程是使用 Adaptive Server Enterprise 15.0 创建的。本教程可能也适用于其它版本的 Adaptive Server Enterprise，但并不能保证这一点。本教程还假定您已经在本地计算机上安装了 Adaptive Server Enterprise 15.0。您也可以使用 Sybase Open Client 远程访问 Adaptive Server Enterprise 15.0。

教程假定 Adaptive Server Enterprise 服务器上安装了 pubs2 示例模式。pubs2 示例模式随 Adaptive Server Enterprise 15.0 共同提供，在安装时可以选择安装该模式。本教程将其用作统一数据库。可以在 Adaptive Server Enterprise 的文档中找到关于此示例的信息，也可以在 http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sqlug/html/sqlug/sqlug894.htm 上在线查看关于此示例的信息。

本教程使用缺省的 **sa** 帐户。刚安装 Adaptive Server Enterprise 时，**sa** 帐户的口令为空。本教程假定您已将空口令更改为有效口令。有关在 Adaptive Server Enterprise 中更改空口令的详细信息，请参见 http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sqlug/html/sqlug/sqlug894.htm。

概述

此教程将介绍如何：

- 在设计远程模式时提供重要的注意事项，如远程表的同步方向。
- 向统一数据库和远程数据库添加唯一主键。
- 创建用于将 MobiLink 连接到 Adaptive Server Enterprise 数据库的 ODBC 数据源。
- 使用 [**创建同步模型向导**] 在统一数据库和远程数据库之间建立同步。
- 使用 [模型] 模式自定义同步模型。
- 使用 [**部署同步模型向导**] 部署统一数据库和远程数据库。
- 将远程客户端与统一数据库同步。

建议阅读的背景知识

有关 MobiLink 同步的概述，请参见“了解 MobiLink 同步”第 3 页。

第 1 课：设计模式

教程假定 Adaptive Server Enterprise 服务器上安装了 pubs2 示例模式。可以将 Adaptive Server Enterprise 服务器安装在本地计算机上，也可以使用 Sybase Open Client 远程访问 Adaptive Server Enterprise 服务器。

此处将 pubs2 示例模式用作统一数据库模式。它包含有关书店、标题、作者、出版商和销售等方面的信息。下表对 Adaptive Server Enterprise 数据库中的各个表进行说明：

表	说明
au_pix	作者照片。
authors	系统中不同 TITLES 的作者。
discounts	特定 STORES 的各种折扣记录。
sales	每条销售记录都是某个特定书店完成的一次销售。
salesdetail	包含有关某一特定销售中的不同 TITLES 的信息。
stores	每条书店记录都表示系统中的一家书店或分店。
titleauthor	包含有关 TITLES 和 AUTHORS 之间的创作对应关系的信息。
titles	系统中所有不同书籍的记录。
blurbs、publishers 和 roysched	包含此演示中不需要的信息。

设计远程模式

如果每个书店都拥有统一数据库的完整副本，既没有必要，又会造成效率低下。而远程模式虽然使用相同的表名称，但仅包含与特定书店相关的信息。为了实现这一点，可按以下方式将远程模式设计成统一数据库的子集：

统一表	远程表
au_pix	包括所有行。
authors	包括所有行。
discounts	按 stor_id 过滤。
sales	按 stor_id 过滤。
salesdetail	按 stor_id 过滤。
stores	按 stor_id 过滤。

统一表	远程表
titleauthor	包括所有行。
titles	包括所有行。
blurbs	远程表中不包含该行。
publishers	远程表中不包含该行。
roysched	远程表中不包含该行。

各家书店都需要保存所有标题和作者的记录，以便顾客可以搜索书店的库存情况。但是，书店不需要保留有关出版商和版税的信息，所以这些信息不对各个书店同步。各家书店需要保存有关销售和折扣的信息，但是不需要与其它书店相关联的此类信息。这一点可以通过根据书店标识符来过滤行而实现。

注意
如果远程数据库不需要某些列，也可以从表中挑出一个列子集。

下一步是选择每个表的同步方向。您应分别考虑远程数据库需要读取哪些信息以及远程数据库需要创建、更改或删除哪些信息。在本例中，书店只需具有对作者和标题列表的访问权限，而从不会向系统输入新作者。这样就限制了始终需要从位于总部的统一数据库将作者和标题输入系统。但是，书店需要能够定期记录新的销售情况。这些因素决定了各表具有以下的同步方向：

表	同步
titleauthor	仅下载到远程表。
authors	仅下载到远程表。
au_pix	仅下载到远程表。
titles	仅下载到远程表。
stores	仅下载到远程表。
discounts	仅下载到远程表。
sales	下载和上载。
salesdetail	下载和上载。

第 2 课：准备统一数据库

本教程使用缺省 **sa** 帐户连接到 pubs2 数据库。刚安装 Adaptive Server Enterprise 时，**sa** 帐户的口令为空。本教程假定您已将空口令更改为有效口令。有关在 Adaptive Server Enterprise 中更改空口令的详细信息，请参见 http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sqlug/html/sqlug/sqlug894.htm。

在本课中，将增加统一数据库的大小以进行 MobiLink 同步，同时创建唯一主键。

增加统一数据库的大小

MobiLink 需要向 pubs2 数据库添加系统表和其它对象，以便进行同步。为此，必须增大 pubs2 数据库的大小。

♦ 增加统一数据库的大小

1. 在 Adaptive Server Enterprise 中使用 isql 实用程序，以 **sa** 身份连接到 pubs2 数据库。在命令提示符处，运行以下命令（全部内容都输入到一行上）：

   ```
   isql
   -U sa
   -P Your password for sa account
   -D pubs2
   ```

   如果正远程访问 Adaptive Server Enterprise，请使用 -S 参数指定服务器名称。

2. 要获得增加数据库大小的适当权限，必须访问主数据库。在 isql 中运行以下命令：

   ```
   use master
   ```

3. 在 Adaptive Server Enterprise 中，数据库存储在磁盘或磁盘的一个部分中。要增大 pubs2 数据库，请运行以下命令（必须指定存储 pubs2 的磁盘）：

   ```
   ALTER DATABASE pubs2 ON disk name = 33
   ```

添加唯一主键

在同步系统中，表的主键是在不同数据库中唯一地标识行的唯一方式，也是检测冲突的唯一方式。正在协调的每一个表都必须具有一个主键。切勿对主键进行更新。同时必须确保已插入到一个数据库中的主键值不被插入到另一数据库。

可通过多种方法来生成唯一主键。为简便起见，本教程使用了复合主键方法。此方法使用在所有统一和远程数据库中唯一的多个列来生成主键。

♦ 向统一数据库添加唯一主键

1. 在 Adaptive Server Enterprise 中使用 isql 实用程序，以 **sa** 身份连接到 pubs2 数据库。在命令提示符处，运行以下命令（全部内容都输入到一行上）：

   ```
   isql
   -U sa
   -P Your password for sa account
   -D pubs2
   ```

   如果正远程访问 Adaptive Server Enterprise，请使用 -S 参数指定服务器名称。

2. 根据在步骤 5 中为 salesdetail 表创建的复合主键，以下几行并不是唯一的。为简化起见，请运行以下命令删除这些行：

```
DELETE FROM salesdetail
WHERE stor_id = '5023'
AND ord_num = 'NF-123-ADS-642-9G3'
AND title_id = 'PC8888'

DELETE FROM salesdetail
WHERE stor_id = '5023'
AND ord_num = 'ZS-645-CAT-415-1B2'
AND title_id = 'BU2075'
```

3. 以下索引会妨碍在步骤 5 中创建主键。要删除这些索引，请运行以下命令：

```
DROP INDEX authors.auidind
DROP INDEX titleauthor.taind
DROP INDEX titles.titleidind
DROP INDEX sales.salesind
```

4. 添加唯一主键。

```
ALTER TABLE au_pix ADD PRIMARY KEY (au_id)
ALTER TABLE authors ADD PRIMARY KEY (au_id)
ALTER TABLE titleauthor ADD PRIMARY KEY (au_id, title_id)
ALTER TABLE titles ADD PRIMARY KEY (title_id)
ALTER TABLE discounts ADD PRIMARY KEY (discounttype)
ALTER TABLE stores ADD PRIMARY KEY (stor_id)
ALTER TABLE sales ADD PRIMARY KEY (stor_id, ord_num)
ALTER TABLE salesdetail ADD PRIMARY KEY (stor_id, ord_num, title_id)
```

运行以上命令后，MobiLink 服务器将顺利地连接到统一数据库，并将其设置为可与任意数量的远程表进行同步。

注意
也可以与没有主键的统一数据库同步数据。但此时您必须记录您自己的作用于影子表的同步事件，以使用这些事件在其它表中唯一地标识行。

所有数据库中的唯一主键

在第 4 课中，将根据统一模式创建远程模式。这意味着远程模式和统一模式具有相同的主键。

列经过特别的选择，以确保对于所有数据库存在唯一主键。对于 sales 表，其主键由 stor_id 和 ord_num 两列构成。任何插入到远程 sales 表的值都必须具有唯一订单号（而 stor_id 值始终不变）。这样可以确保各个远程 sales 表的唯一性。如果需要从多个书店上载数据，则统一 sales 表中的主键可防止发生冲突。由于各个书店的 stor_id 值各不相同，因而来自一家书店的上载始终是唯一的。

对于 salesdetail 表，其主键由 stor_id、ord_num 和 title_id 列构成。一个订单中可能包含若干书籍标题。对于远程 sales 表，不同的行可能具有相同的 stor_id 和 ord_num 值，但是其 title_id 值必须不同。这可以确保每个远程 salesdetail 表的唯一性。与 sales 表类似，从某一书店向统一数据库上载的每个项对于另一个书店都是唯一的，因为它们的 stor_id 值不同。

进一步阅读

有关 Adaptive Server Enterprise 问题的详细信息，请参见“Adaptive Server Enterprise 统一数据库”一节 《MobiLink - 服务器管理》。

有关生成唯一主键不同方式的详细信息，请参见“维护唯一主键”一节 《MobiLink - 服务器管理》。

第 3 课：连接 MobiLink

本课将创建一个 ODBC 数据源，用于将 MobiLink 连接到统一数据库。

♦ 将 MobiLink 连接到统一数据库

1. 创建 ODBC 数据源。

   您应使用 Adaptive Server Enterprise 所提供的 ODBC 驱动程序。对于本教程，使用以下配置设置：

[常规] 选项卡字段	值
数据源名	ase_cons
说明	
服务器名（ASE 主机名）	localhost
服务器端口	5000
数据库名	pubs2
登录 ID	sa
使用游标	未选

[Transaction] 选项卡字段	值
Server Initiated Transactions	未选

2. 测试 ODBC 连接：
   a. 在 [**常规**] 选项卡上，单击 [**测试连接**].
      将出现 Adaptive Server Enterprise 登录屏幕。
   b. 输入 **sa** 帐户的口令。
      将出现 [**Logon Succeeded**] 消息。

配置 ODBC 数据源后，就可使用 MobiLink 插件连接到统一数据库并创建同步模型。

进一步阅读

有关推荐用于 MobiLink 的 ODBC 驱动程序的详细信息，请参见推荐用于 MobiLink 的 ODBC 驱动程序。

第 4 课：创建同步模型

[**创建同步模型向导**] 提供了在统一数据库和远程数据库之间设置同步的分步说明。

♦ 创建同步模型

1. 启动 Sybase Central。
2. 选择 [**工具**] » [**MobiLink 11**] » [**设置 MobiLink 同步**]。
3. 在 [**欢迎**] 页面上的 [**您要给新同步模型指定什么名称**] 字段中键入 **sync_ase**，并指定新模型的位置。单击 [**下一步**]。
4. 在 [**主键需求**] 页面中，将三个复选框全部选中（要保证第 2 课中所述的唯一主键）。单击 [**下一步**]。
5. 在 [**统一数据库模式**] 页面上执行以下操作：
   a. 单击 [**选择统一数据库**]。
   b. 单击 [**ODBC 数据源名称**]，然后选择 **ase_cons**。
   c. 在 [**用户 ID**] 字段键入 **sa**。
   d. 在 [**口令**] 字段中键入 **sa**。
   e. 单击 [**确定**]。
      如果这是 MobiLink 第一次使用统一数据库，将出现一条消息，询问您是否安装 MobiLink 系统设置。单击 [**是**]。
   f. 单击 [**下一步**]。
6. 在 [**远程数据库模式**] 页面上，单击 [**否，新建一个远程数据库模式**]。单击 [**下一步**]。
7. 在 [**新的远程数据库模式**] 页面上的 [**您要在远程数据库中使用哪些统一数据库表**] 列表中选择下列表：
   - au_pix
   - authors
   - discounts
   - sales
   - salesdetail
   - stores
   - titleauthor
   - titles
8. 单击 [**下一步**]。
9. 在 [**下载类型**] 页面上，单击 [**基于时间戳的下载**]。单击 [**下一步**]。
   选择基于时间戳的下载可以使所传送的数据量最小，因为此下载类型仅传输自上次下载后经过更新的数据。
10. 在 [**时间戳下载选项**] 页面上，选择 [**使用影子表来保存时间戳列**]。单击 [**下一步**]。
    最好选择使用影子表，因为这样不会对现有的表进行任何更改。

11. 在 [**下载删除**] 页面上执行以下操作：
    a. 在 [**是否要在远程数据库中删除那些在统一数据库中被删除的数据？**] 字段中，单击 [**是**]。
    b. 单击 [**使用影子表来记录删除**]。
       MobiLink 将在统一数据库中创建影子表，以执行删除。
    c. 单击 [**下一步**]。
12. 在 [**下载子集**] 页面上，单击 [**是，向每个远程数据库下载相同的数据**]。单击 [**下一步**]。
    在 [模型] 模式一课的步骤 2 中，将指定如何使用自定义逻辑向远程数据库下载特定的数据。
13. 在 [**上载冲突检测**] 页面上，单击 [**无冲突检测**]。单击 [**下一步**]。
    尽管本教程未指定任何冲突检测，但许多应用程序都需要进行冲突检测。
14. 在 [**发布、脚本版本和说明**] 页面上执行以下操作：
    a. 在 [**您要给此发布指定什么名称**] 字段中键入 **sync_ase_publication**。
    b. 在 [**您要给此脚本版本指定什么名称**] 字段中键入 **sync_ase_scriptversion**。
       发布是远程数据库上的对象，用于指定要同步的数据。MobiLink 服务器脚本定义了应如何将来自远程数据库的上载数据应用到统一数据库，同时也对脚本版本组的脚本进行了定义。可以针对不同的应用程序使用不同的脚本版本，从而使您只需维护单一 MobiLink 服务器就可同时同步多个不同的应用程序。
    c. 单击 [**完成**]。
       模型将以 [模型] 模式出现。

[模型] 模式

1. 要指定数据的同步方向，在 [**映射方向**] 列中按以下方式设置方向：
   - Sales 和 salesdetail 表为双向同步（既能上载又可下载）。
   - 其余的表为仅下载同步。
2. 按远程 ID 过滤下载到远程数据库的行：
   a. 对于 stores 表，将 [**下载子集**] 更改为 [**自定义**]。
   b. 在屏幕底部打开 [**下载子集**] 选项卡。
   c. 远程 ID 可唯一地标识远程数据库。要按远程 ID 过滤行，需要对 download_cursor 脚本的 WHERE 子句添加限制。要执行以上操作，在 [**要在下载游标的 WHERE 子句中使用的 SQL 表达式**] 文本框中键入以下搜索条件：

      ```
      "dbo"."stores"."stor_id" = {ml s.remote_id}
      ```

      下载游标脚本指定了要从各个表中下载到远程数据库的行和列。上述搜索条件可确保仅下载有关一家书店（即标识符与数据库的远程 ID 相同的那家书店）的信息。
   d. 按上述步骤对 sales、salesdetail 和 discounts 表进行操作。您需要在表达式中相应地重命名这些表。
3. 保存同步模型。

至此，同步模型创建完成，接下来您可以部署该模型。

进一步阅读

- “建立统一数据库”一节 《MobiLink - 服务器管理》
- “MobiLink 服务器系统表” 《MobiLink - 服务器管理》
- “MobiLink 系统过程”一节 《MobiLink - 服务器管理》
- “编写 download_delete_cursor 脚本”一节 《MobiLink - 服务器管理》
- “冲突处理”一节 《MobiLink - 服务器管理》
- “解决冲突”一节 《MobiLink - 服务器管理》
- “发布数据”一节 《MobiLink - 客户端管理》
- “模型模式”一节第 30 页
- “修改下载类型”一节第 32 页
- “修改冲突检测和冲突解决”一节第 35 页
- “修改表映射和列映射”一节第 30 页

第 5 课：部署同步模型

[**部署同步模型向导**] 允许您部署统一数据库和远程数据库。可以分别部署这两个数据库，也可以同时部署两者。[**部署同步模型向导**] 会引导您逐步完成对部署选项的配置。

♦ 部署同步模型

1. 在 Sybase Central 的左窗格中，右击同步模型，然后选择 [**部署**]。
2. 单击 [**为下列一项或多项内容指定部署详细信息**] 并选择 [**统一数据库**]、[**远程数据库和同步客户端**] 和 [**MobiLink 服务器**]。单击 [**下一步**]。
3. 在 [**统一数据库部署目标**] 页面上完成以下字段：
   a. 单击 [**将更改保存到以下 SQL 文件**] 并接受文件的缺省位置。

      MobiLink 将生成对统一数据库进行修改的 *.sql* 文件来设置同步。可以稍后检查 *.sql* 文件并进行自己的更改。这时，您必须亲自运行 *.sql* 文件。
   b. 要立即将更改应用到统一数据库，请单击 [**连接到统一数据库以便直接应用更改**]。
   c. 单击 [**选择统一数据库**]。
   d. 选择 [**ODBC 数据源名称**]，然后选择 **ase_cons**。
   e. 在 [**用户 Id**] 字段中键入 **sa**。
   f. 在 [**口令**] 字段中键入 **sa**。
   g. 单击 [**确定**]。
   h. 单击 [**下一步**]。

      将出现一个提示，询问是否要创建 *consolidated* 目录。单击 [**是**]。
4. 在 [**远程数据库部署**] 页面上，单击 [**新 SQL Anywhere 数据库**]。单击 [**下一步**]。
5. 在 [**新 SQL Anywhere 远程数据库**] 页面上完成以下字段：
   a. 单击 [**建立带有可创建数据库命令的命令文件和 SQL 文件**]。

      MobiLink 将生成另一个 *.sql* 文件，该文件带有可使用所有模式和同步信息设置远程数据库的命令。
   b. 在 [**SQL 文件**] 字段中，接受 SQL 文件的缺省位置。
   c. 单击 [**创建远程 SQL Anywhere 数据库**]。

      如果选择不进行上述操作，那么必须生成一个新的远程数据库然后对其运行 *.sql* 文件。这允许您稍后检查 *.sql* 文件，然后进行自定义修改。
   d. 在 [**SQL Anywhere 数据库文件**] 字段中，接受远程 SQL Anywhere 数据库的缺省位置。
   e. 单击 [**下一步**]。

      将出现一个提示，询问是否要创建 *remote* 目录。单击 [**是**]。
6. 在 [**MobiLink 用户**] 页面上完成以下字段：
   a. 在 [**您要使用什么用户名连接到 MobiLink 服务器**] 字段中键入 **ase_remote**。
   b. 在 [**您要使用什么口令**] 字段中键入 **ase_pass**。

c. 单击 [**下一步**]。

7. 在 [**同步流参数**] 页面中单击 [**TCP/IP**]，并且在 [**端口**] 字段中键入 **2439**。单击 [**下一步**]。
8. 在 [**客户端流参数**] 页面的 [**主机**] 字段中键入 **localhost**。单击 [**下一步**]。

   可以选择键入您的计算机名或 IP 地址、要使用的其它网络服务器名或 IP 地址，或其它客户端流选项。
9. 要接受缺省设置，在 [**MobiLink 服务器流参数**] 和 [**MobiLink 服务器的详细程度**] 页面上单击 [**下一步**]。
10. 在 [**MobiLink 服务器的选项**] 页面的 [**您要给 MobiLink 服务器指定什么名称**] 字段中键入 **ase_mlsrv**。单击 [**下一步**]。

    将出现一条消息，询问是否要创建 *mlsrv* 目录。单击 [**是**]。
11. 选择远程同步客户端的详细程度，并使用远程数据库日志文件的缺省文件名。单击 [**下一步**]。
12. 单击 [**完成**]。
13. 单击 [**关闭**]。

现在，统一数据库已完成配置，可与众多远程客户端进行同步，并且您已成功部署了一个远程客户端。如果要部署其它远程客户端，可再次运行此向导，但此时需要创建新的 MobiLink 用户并跳过对统一数据库和 MobiLink 服务器的部署。由于它们已部署，因此您只需部署其它远程同步客户端。

进一步阅读

- “部署模型”一节第 40 页
- “创建远程数据库”一节 《MobiLink - 客户端管理》
- “MobiLink 用户简介”一节 《MobiLink - 客户端管理》

第 6 课：启动服务器和客户端

在本课中，您将启动 MobiLink 服务器和远程数据库。

在前几课中，您修改了下载游标脚本以下载有关某一书店的信息。而在本课中，将通过把远程 ID 设置为书店标识符的方法来指定书店。

♦ 启动 MobiLink 服务器

1. 在命令提示符处，浏览到创建的同步模型所在的文件夹。（此文件夹是在 [**Synchronization Model Wizard**] 的第一步中所选择的根目录。）

   如果使用了推荐的目录名，则应浏览到以下目录：*sync_ase\mlsrv*。

2. 要启动 MobiLink 服务器，请运行以下命令：

   ```
   sync_ase_mlsrv.bat "dsn=ase_cons;uid=sa;pwd=your password for sa account;"
   ```

   - **sync_ase_mlsrv.bat**　是用于启动 MobiLink 服务器的命令文件。
   - **dsn**　是 ODBC 数据源名称。
   - **uid**　是用于连接到统一数据库的用户名（Adaptive Server Enterprise 的缺省用户名为 **sa**）。
   - **pwd**　是以 **sa** 进行连接时所使用的口令。

   此命令成功运行后，在 [**MobiLink 服务器消息**] 窗口中将显示消息 [MobiLink 服务器已启动]。

   如果 MobiLink 服务器启动失败，请检查统一数据库的连接信息。

♦ 启动远程数据库

1. 在命令提示符处，浏览到 [**部署同步模型向导**] 创建的远程数据库所在的目录。

   如果使用了推荐的目录名，则应浏览到以下目录：*sync_ase\remote*。

2. 要启动远程 SQL Anywhere 数据库，请运行以下命令：

   ```
   dbeng11 -n remote_eng sync_ase_remote.db -n remote_db
   ```

   - **dbeng11**　是用于启动 SQL Anywhere 数据库的数据库服务器。
   - **remote_eng**　是数据库服务器名称。
   - **sync_ase_remote.db**　是在 remote_eng 上启动的数据库文件。
   - **remote_db**　是 remote_eng 上的数据库名称。

   上述命令成功运行后，名为 remote_eng 的 SQL Anywhere 数据库服务器将启动，并加载名为 remote_db 的数据库。

设置远程 ID

在远程模式中，每个远程数据库都代表一家书店。您编写的同步脚本包含了指示 MobiLink 服务器根据远程数据库的远程 ID 下载数据子集的逻辑。您必须将数据库的远程 ID 设置为有效的书店标识符值。

您应在第一次同步之前完成上述步骤，因为当远程设备首次同步时，它将下载与该书店（在本例中为 Thoreau Reading 连锁折扣书店）相关的所有信息。

♦ 将远程 ID 设置为有效的书店标识符

1. 选择一个有效的书店标识符。
   a. 在 Adaptive Server Enterprise 中使用 isql 实用程序，以 **sa** 身份连接到 pubs2 数据库。在命令提示符处，运行以下命令（全部内容都输入到一行上）：

      ```
      isql
      -U sa
      -P Your password for sa account
      -D pubs2
      ```

      如果正远程访问 Adaptive Server Enterprise，请使用 -S 参数指定服务器名称。
   b. 要查看 stores 表中有效书店标识符的列表，请执行以下语句：

      ```
      SELECT * FROM stores
      ```

      在本教程中，以远程数据库代表 Thoreau Reading 连锁折扣书店，该书店的标识符值为 5023。
   c. 要退出 Adaptive Server Enterprise，请运行以下命令：

      ```
      exit
      ```
2. 要将数据库的远程 ID 设置为 5023，请运行以下命令（全部内容都输入到一行上）：

   ```
   dbisql
      -c "eng=remote_eng;dbn=remote_db;uid=DBA;pwd=sql"
      "SET OPTION PUBLIC.ml_remote_id='5023'"
   ```

   - **dbisql** 是用于对 SQL Anywhere 数据库执行 SQL 命令的应用程序。
   - **eng** 用于将数据库服务器名指定为 remote_eng。
   - **dbn** 用于将数据库名指定为 remote_db。
   - **uid** 是用于连接到远程数据库的用户名。
   - **pwd** 是用于连接到远程数据库的口令。
   - **SET OPTION PUBLIC.ml_remote_id='5023'** 是用于将远程 ID 设置为 5023 的 SQL 命令。

进一步阅读

- “SQL Anywhere 数据库服务器”一节 《SQL Anywhere 服务器 - 数据库管理》
- “同步已部署的模型”一节第 42 页
- “运行 MobiLink 服务器”一节 《MobiLink - 服务器管理》
- “远程 ID”一节 《MobiLink - 客户端管理》

第 7 课：同步

现在您可以进行第一次远程客户端同步了。可以使用 MobiLink 客户端程序 dbmlsync 来完成这一过程。Dbmlsync 将连接到远程数据库、通过 MobiLink 服务器对自身进行验证，并根据远程数据库中的发布执行要同步远程数据库和统一数据库所必需的所有上载和下载。

♦ 同步远程客户端

- 在命令提示符处，运行以下命令（全部内容都输入到一行上）：

```
dbmlsync
   -c "eng=remote_eng;dbn=remote_db;uid=DBA;pwd=sql;"
   -n sync_ase_publication
   -u ase_remote -mp ase_pass
```

  - **dbmlsync** 是同步应用程序。
  - **eng=remote_eng** 用于指定远程数据库服务器的名称。
  - **dbn=remote_db** 用于指定远程数据库的名称。
  - **uid** 是用于连接到远程数据库的用户名。
  - **pwd** 是用于连接到远程数据库的口令。
  - **sync_ase_publication** 是远程设备上的发布名称，此发布用于执行同步。（此发布由 [**创建同步模型向导**] 创建。）
  - **ase_remote** 是使用 MobiLink 服务器进行验证时所用的用户名。
  - **ase_pass** 是使用 MobiLink 服务器进行验证时所用的口令。

  同步的进度显示在 [**SQL Anywhere MobiLink 客户端消息**] 窗口中。上述命令成功运行后，dbmlsync 应用程序将使用统一数据库中的信息子集填充远程数据库。

  如果同步失败，则检查传递给 dbmlsync 应用程序的连接信息以及 MobiLink 用户名和口令。如果问题仍存在，则检查所使用的发布名，并确保统一数据库和 MobiLink 服务器正在运行。您也可以检查同步日志的内容（服务器和客户端）。

> **注意**
> 如果要在 MobiLink 服务器以外的计算机上运行 dbmlsync 应用程序，还必须传递用于指定 MobiLink 服务器位置的参数。

查看数据

在使用 MobiLink 服务器将远程客户端与统一数据库成功同步后，远程数据现在应该包含有关某一书店的信息。在 Sybase Central 中，您可以使用 SQL Anywhere 11 插件来验证这一点。

♦ 查看远程数据库中的数据

1. 启动 Sybase Central。
2. 连接到远程数据库：
   a. 在左窗格中，右击 [**SQL Anywhere 11**] 并选择 [**连接**]。

b. 键入 **DBA** 作为 [**用户 ID**]，相应的 [**口令**] 为 **sql**。

c. 在 [**标识**] 选项卡中，键入 **remote_eng** 作为 [**服务器名**]，而 [**数据库名**] 为 **remote_db**。

d. 单击 [**确定**]。

3. 如果从统一数据库创建的表不可见，请执行以下步骤：

a. 右击 *remote_db*，然后单击 [**配置所有者过滤**]。

b. 选择 [**dbo**]，然后单击 [**确定**]。

从统一数据库创建的表将出现在左窗格中。dbo 对这些表的所有权保存在远程数据库中。

4. 选择任意远程表，然后在右窗格中单击 [**数据**] 选项卡。

在 sales、salesdetail 和 stores 表中，所有记录都是关于标识符为 5023 的书店的。此特定书店不关注其它书店的销售信息。为此，需要设置同步脚本按照远程 ID 来过滤出相应的行，并需要将此数据库的远程 ID 设置为某特定书店的标识符值。这样可使该特定书店的数据库所占的空间更小，并且同步时间也更少。正是由于远程数据库的大小始终维持在最低水平，某些经常执行的操作（如输入新的销售记录或处理之前销售的退款）才能运行得更快，效率更高。

进一步阅读

- “同步过程”一节第 16 页
- “dbmlsync 语法”一节 《MobiLink - 客户端管理》

清除

重新生成 pubs2 数据库并从计算机中删除所有教程资料。

♦ 重新生成 pubs2 数据库

- 要运行用于安装 pubs2 数据库的脚本，请运行以下命令：

```
isql
-U sa
-P Your password for sa account
-i %SYBASE%\%SYBASE_ASE%\scripts\instpbs2
```

如果正远程访问 Adaptive Server Enterprise，请使用 -S 参数指定服务器名称。还必须以本地方式将 instpbs2 文件复制到计算机上。需要更新 -i 参数以便指定 instpbs2 文件的新位置。

♦ 删除同步模型

1. 启动 Sybase Central。
2. 在右窗格中双击 [**MobiLink 11**]。

   sync_ase 模型出现。
3. 右击 **sync_ase** 并选择 [**删除**]。

♦ 消除远程数据库

- 要消除远程数据库，请使用 dberase 实用程序。运行以下命令：

```
dberase sync_ase\remote\sync_ase_remote.db
```

教程：使用 Java 同步逻辑

目录

Java 同步教程简介

本教程将指导您完成使用 Java 同步逻辑的基本步骤。可使用 CustDB 示例作为 SQL Anywhere 统一数据库，为 MobiLink 表级别事件指定简单的类方法。该过程还涉及使用用来设置编译的 Java 类路径的选项来运行 MobiLink 服务器 (mlsrv11)。

必需的软件

- SQL Anywhere 11
- Java 软件开发工具包

能力和经验

您需要：

- 熟悉 Java
- 了解 MobiLink 事件脚本的基本知识

目标

您将掌握并熟悉以下内容：

- 使用简单的 Java 类方法编写 MobiLink 表级别事件

重要概念

本节使用以下步骤通过 MobiLink CustDB 示例数据库实现基本的基于 Java 的同步：

- 使用 MobiLink 服务器 API 引用编译源文件
- 为特定表级别事件指定类方法
- 用 -sl java 选项运行 MobiLink 服务器 (mlsrv11)
- 使用示例 Windows 客户端应用程序测试同步

建议阅读的背景知识

有关同步脚本的详细信息，请参见“同步脚本介绍”一节 《MobiLink - 服务器管理》。

第 1 课：编译 CustdbScripts Java 类

Java 类在方法中封装同步逻辑。

在本课中，您将编译一个与 CustDB 示例数据库关联的类。

MobiLink 数据库示例

SQL Anywhere 随附一个 SQL Anywhere 示例数据库 (CustDB)，该数据库已进行了同步设置，其中包括同步所需的 SQL 脚本。例如，CustDB ULCustomer 表是一个同步表，它支持多种表级别事件。

CustDB 的设计用途是作为 UltraLite 和 SQL Anywhere 客户端的统一数据库服务器。CustDB 数据库有一个名为 SQL Anywhere 11 CustDB 的 DSN。

CustdbScripts 类

本节将创建一个名为 CustdbScripts 的 Java 类，其中含有用于处理 ULCustomer upload_insert 和 download_cursor 事件的逻辑。可在文本编辑器中输入 CustdbScripts 代码，并将文件另存为 *CustdbScripts.java*。

- **创建 CustdbScripts.java：**

1. 为 Java 类和程序集创建一个目录。

   本教程假定路径为 *c:\mljava*。

2. 使用文本编辑器输入 CustdbScripts 代码：

```
public class CustdbScripts {
    public static String UploadInsert() {
        return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)");
    }
    public String DownloadCursor(java.sql.Timestamp ts,String user ) {
        return("SELECT cust_id, cust_name FROM ULCustomer where
last_modified >= ' " + ts +" ' ");
    }
}
```

> **注意**
> 当创建您自己的自定义脚本时，确保类和关联的方法定义为 **public**。

3. 将文件在 *c:\mljava* 中另存为 *CustdbScripts.java*。

编译 Java 源代码

要执行 Java 同步逻辑，MobiLink 服务器必须能够访问 *mlscript.jar* 中的类。此 JAR 文件包含要在 Java 方法中使用的 MobiLink 服务器 API 类的存储库。

为 MobiLink 编译 Java 源代码时，必须包含 *mlscript.jar* 才能使用 MobiLink 服务器 API。本节中使用 javac 实用程序的 -classpath 选项来为 CustdbScripts 类指定 *mlscript.jar*。

- **编译 Java 源代码 (Windows)**

1. 在命令提示符下，导航到包含 *CustdbScripts.java* 的文件夹 (*c:\mljava*)。

2. 运行以下命令。

```
javac custdbscripts.java -classpath "%sqlany11%\java\mlscript.jar"
```

将生成 *CustdbScripts.class* 文件。

进一步阅读

有关面向 Java 的 MobiLink 服务器 API 的详细信息，请参见“用于 Java 的 MobiLink 服务器 API 参考”一节 《MobiLink - 服务器管理》。

有关 Java 方法的详细信息，请参见“方法”一节 《MobiLink - 服务器管理》。

有关 CustDB 示例数据库和使用备用 RDBMS 服务器的详细信息，请参见“建立 CustDB 统一数据库”一节第 48 页。

第 2 课：指定类方法来处理事件

上一课中创建的 *CustdbScripts.class* 封装了 UploadInsert 和 DownloadCursor 方法。这些方法分别包含 ULCustomer upload_insert 和 download_cursor 事件的实现。

在本节中，使用两种方法为表级别事件指定类方法：

- 使用 Sybase Central 中的 [MobiLink 管理] 模式：

  使用 Sybase Central 连接到 CustDB 数据库，将 upload_insert 脚本的语言更改为 Java，然后指定 CustdbScripts.UploadInsert 来处理事件。

- 使用 ml_add_java_table_script 存储过程：

  使用 Interactive SQL 连接到 CustDB 数据库并执行 ml_add_java_table_script，指定 CustdbScripts.DownloadCursor 来处理 download_cursor 事件。

♦ 指定 CustdbScripts.UploadInsert 来处理 ULCustomer upload_insert 事件

1. 使用 Sybase Central 的 [MobiLink 管理] 模式连接到示例数据库：
   a. 启动 Sybase Central。
   b. 单击 [**视图**] » [**文件夹**]。
   c. 单击 [**连接**] » [**使用 MobiLink 11 连接**]。
   d. 单击 [**标识**] 选项卡。
   e. 单击 [**ODBC 数据源名称**]，然后键入 **SQL Anywhere 11 CustDB**。
   f. 单击 [**确定**]。

   Sybase Central 在 MobiLink 11 插件下显示 CustDB 数据源。
2. 将 ULCustomer 表的现有 upload_insert 事件删除：
   a. 在左窗格中，双击 [**同步表**] 文件夹，然后选择 ULCustomer 表。在右窗格中，出现一个表级别脚本列表。
   b. 选择与 custdb 11.0 upload_insert 事件关联的表脚本。单击 [**编辑**] » [**删除**]。
   c. 在 [**确认删除**] 窗口中，单击 [**是**]。custdb 11.0 upload_insert 事件即从 ULCustomer 表中删除。
3. 为 ULCustomer 表创建一个新的 upload_insert 事件：
   a. 在 [**同步表**] 文件夹中选择 ULCustomer 表，然后单击 [**文件**] » [**新建**] » [**表脚本**]。
   b. 选择 custdb 11.0 作为脚本版本。
   c. 选择 upload_insert 作为要创建的事件，然后单击 [**下一步**]。
   d. 选择 [**创建新脚本定义**]，然后选择 [**Java**]。
   e. 单击 [**完成**]。
4. 指示 MobiLink 服务器对 upload_insert 事件运行 **CustdbScripts.UploadInsert** 方法。
   a. 选择 custDB 11.0 - upload_insert 脚本。
   b. 在右窗格中，输入以下代码：

```
CustdbScripts.UploadInsert
```

c. 单击 [**文件**] » [**保存**]。

5. 退出 Sybase Central。

此步骤使用 Sybase Central 将 Java 方法指定为 ULCustomer upload_insert 事件的脚本。

或者，也可以使用 ml_add_java_connection_script 和 ml_add_java_table_script 存储过程。如果使用大量 Java 方法处理同步事件，则使用这些存储过程效率更高。

请参见“ml_add_java_connection_script 系统过程”一节 《MobiLink - 服务器管理》和“ml_add_java_table_script 系统过程”一节 《MobiLink - 服务器管理》。

♦ 指定 CustdbScripts.DownloadCursor() 来处理 ULCustomer download_cursor 事件

1. 使用 Interactive SQL 连接到示例数据库。
   a. 单击 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Interactive SQL**]，或运行以下命令：

```
dbisql
```

   b. 单击 [**标识**] 选项卡。
   c. 单击 [**ODBC 数据源名称**]，然后键入 **SQL Anywhere 11 CustDB**。
   d. 单击 [**确定**]。
2. 在 Interactive SQL 中运行以下命令：

```
CALL ml_add_java_table_script(
'custdb 11.0',
'ULCustomer',
'download_cursor',
'CustdbScripts.DownloadCursor');
COMMIT;
```

以下是对每个参数的说明：

参数	说明
custdb 11.0	脚本版本。
ULCustomer	同步表。
download_cursor	事件名称。
CustdbScripts.DownloadCursor	完全限定的 Java 方法。

3. 退出 Interactive SQL。

在本课中，您指定了 Java 方法来处理 ULCustomer 表级别事件。下一课将确保 MobiLink 服务器装载合适的类文件和 MobiLink 服务器 API。

进一步阅读

有关使用存储过程来添加脚本的详细信息，请参见“ml_add_java_connection_script 系统过程”一节 《MobiLink - 服务器管理》和“ml_add_java_table_script 系统过程”一节 《MobiLink - 服务器管理》。

有关添加和删除同步脚本的一般信息，请参见“添加和删除脚本”一节 《MobiLink - 服务器管理》。

第 3 课：用 -sl java 运行 MobiLink 服务器

用 -sl java -cp 选项运行 MobiLink 服务器可以指定一组目录来搜索其中的类文件，并强制在服务器启动时装载 Java 虚拟机。

♦ **启动 MobiLink 服务器 (mlsrv11) 并装载 Java 程序集**

● 在命令提示符处，运行以下命令：

```
mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -sl java (-cp c:\mljava)
```

显示一条消息，指出服务器已启动。现在，Java 方法会在同步过程中 ULCustomer 表的 upload_insert 事件被触发时执行。

进一步阅读

有关详细信息，请参见“-sl java 选项”一节 《MobiLink - 服务器管理》。

第 4 课：测试同步

UltraLite 随附一个示例 Windows 客户端，当用户发出同步请求时，该客户端自动调用 dbmlsync 实用程序。它是一个简单的销售状态应用程序，您可以对上一课中启动的 CustDB 统一数据库运行该程序。

启动应用程序 (Windows)

♦ **启动并同步示例应用程序**

1. 单击 [开始] » [程序] » [**SQL Anywhere 11**] » [**UltraLite**] » [**Windows 示例应用程序**]。
2. 输入一个职员 ID。

   键入值 **50** 并单击 [**OK**]。

应用程序会自动同步，同时会将一组客户、产品和订单从 CustDB 统一数据库下载到应用程序中。

添加订单 (Windows)

♦ **添加订单：**

1. 单击 [**Order**] » [**New**]。
2. 输入新的客户名称。例如，输入 **Frank Javac**。
3. 选择一种产品，然后输入数量和折扣。
4. 单击 [**OK**] 添加新订单。

您现在已经修改了本地 UltraLite 数据库中的数据。在进行同步之前，此数据没有与统一数据库共享。

♦ **与统一数据库进行同步并触发 upload_insert 事件**

● 单击 [**File**] » [**Synchronize**]。

将出现一条消息，指明已将一个插入操作成功上载到了统一数据库。

进一步阅读

有关 CustDB Windows 应用程序的详细信息，请参见“研究 MobiLink 的 CustDB 示例”第 45 页。

清除

从计算机中删除教程资料。

♦ 从计算机中删除教程资料

1. 将 ULCustomer 表的 upload_insert 和 download_cursor 脚本返回到其原始 SQL 逻辑。
   a. 单击 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Interactive SQL**]，或运行以下命令：

      ```
      dbisql
      ```

   b. 单击 [**标识**] 选项卡。
   c. 单击 [**ODBC 数据源名称**]，然后键入 **SQL Anywhere 11 CustDB**。
   d. 单击 [**确定**]。
   e. 在 Interactive SQL 中运行以下命令：

      ```
      CALL ml_add_table_script( 'custdb 11.0',
         'ULCustomer',
         'upload_insert',
         'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

      CALL ml_add_table_script( 'custdb 11.0',
         'ULCustomer',
         'download_cursor',
         'SELECT "cust_id", "cust_name"
          FROM "ULCustomer" WHERE "last_modified" >= ?' );
      COMMIT;
      ```

2. 要关闭 SQL Anywhere 窗口、MobiLink 窗口和同步客户端窗口，右击各个任务栏项并选择 [**关闭**]。
3. 删除所有与教程相关的 Java 文件。

   删除包含 *CustdbScripts.java* 和 *CustdbScripts.class* 文件的文件夹 (*c:\mljava*)。

   注意
   确保 *c:\mljava* 中没有任何重要文件。

4. 为 Windows 示例应用程序重置数据库。

   从 *samples-dir\UltraLite\CustDB* 目录运行以下命令：

   ```
   newdb.bat
   ```


进一步阅读

有关使用 Java 编写 MobiLink 同步脚本的详细信息，请参见“设置 Java 同步逻辑”一节 《MobiLink - 服务器管理》。

有关说明自定义验证的 Java 同步脚本用法的示例，请参见“Java 同步示例”一节 《MobiLink - 服务器管理》。

有关同步脚本编写的详细信息，请参见“编写同步脚本” 《MobiLink - 服务器管理》和“同步事件” 《MobiLink - 服务器管理》。

有关其它同步方法（如时间戳）的介绍，请参见“同步技术” 《MobiLink - 服务器管理》。

教程：使用 .NET 同步逻辑

目录

.NET 同步教程简介

本教程将指导您完成使用 .NET 同步逻辑的基本步骤。通过使用 CustDB 示例作为 SQL Anywhere 统一数据库，可以为 MobiLink 表级事件指定简单的类方法。该过程还涉及使用用于设置 .NET 程序集路径的选项来运行 MobiLink 服务器 (mlsrv11)。

必需的软件

- SQL Anywhere 11
- Microsoft .NET Framework SDK

能力和经验

您应该：

- 熟悉 .NET
- 了解 MobiLink 事件脚本的基本知识

目标

您将掌握并熟悉以下内容：

- 利用 .NET 类方法编写 MobiLink 表级事件脚本

重要概念

本节通过以下步骤使用 MobiLink CustDB 示例数据库实现基本 .NET 同步：

- 用 MobiLink 参考编译 *CustdbScripts.dll* 私有程序集
- 为表级事件指定类方法
- 用 -sl dnet 选项运行 MobiLink 服务器 (mlsrv11)
- 使用示例 Windows 客户端应用程序测试同步

建议阅读的背景知识

有关同步脚本的详细信息，请参见“同步脚本介绍”一节 《MobiLink - 服务器管理》 。

第 1 课：用 MobiLink 参考编译 CustdbScripts.dll 程序集

.NET 类将同步逻辑封装在方法中。

在本课中，您将编译一个与 CustDB 示例数据库关联的类。

MobiLink 数据库示例

SQL Anywhere 随附一个 SQL Anywhere 示例数据库 (CustDB)，该数据库已设置为可以进行同步，其中包括驱动同步所需的 SQL 脚本。例如，CustDB ULCustomer 表是一个支持多种表级别事件的同步表。

CustDB 的设计用途是作为 UltraLite 和 SQL Anywhere 客户端的统一数据库服务器。CustDB 数据库有一个名为 SQL Anywhere 11 CustDB 的 DSN。

CustdbScripts 程序集

在本节中，您将创建一个名为 CustdbScripts 的 .NET 类，其中包含用于处理 ULCustomer upload_insert 和 download_cursor 事件的逻辑。

MobiLink 服务器 API

要执行 .NET 同步逻辑，MobiLink 服务器必须能够访问 *iAnywhere.MobiLink.Script.dll* 中的类。*iAnywhere.MobiLink.Script.dll* 包含要在 .NET 方法中使用的面向 .NET 的 MobiLink 服务器 API 类的存储库。

有关用于 .NET 的 MobiLink 服务器 API 的详细信息，请参见“用于 .NET 参考的 MobiLink 服务器 API”一节 《MobiLink - 服务器管理》。

编译 CustdbScripts 类时，必须包含此程序集才能使用该 API。可以使用 Visual Studio 或在命令提示符处编译类。

- 在 Visual Studio 中，创建一个新的类库，并输入 CustdbScripts 代码。链接 *iAnywhere.MobiLink.Script.dll*，并构建类的程序集。
- 将 CustdbScripts 代码写入文本文件，并将该文件保存为 *CustdbScripts.cs*（对于 Visual Basic .NET，保存为 *CustdbScripts.vb*）。使用命令行编译器，引用 *iAnywhere.MobiLink.Script.dll* 并构建类的程序集。

♦ 使用 Visual Studio 创建 CustdbScripts 程序集

1. 启动一个新的 Visual C# 或 Visual Basic .NET 类库项目。

   使用 CustdbScripts 作为项目名，并输入相应的路径。本教程假定路径为 *c:\mldnet*。

2. 输入 CustdbScripts 代码。

   如果是 C#，请键入：

```
namespace MLExample
{
class CustdbScripts
{
```

```
  public static string UploadInsert()
 {
   return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)");
 }
 public static string DownloadCursor(System.DateTime ts, string user )
 {
   return("SELECT cust_id, cust_name
           FROM ULCustomer
            WHERE last_modified >= '" + ts.ToString("yyyy-MM-dd
hh:mm:ss.fff") +"'");
 }
}
}
```

如果是 Visual Basic .NET，请键入：

```
Namespace MLExample

 Class CustdbScripts

  Public Shared Function UploadInsert() As String
    Return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)")
  End Function

  Public Shared Function DownloadCursor(ByVal ts As System.DateTime, ByVal
user As String) As String
   Return("SELECT cust_id, cust_name FROM ULCustomer " +
    "WHERE last_modified >= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff")
+"'")
  End Function

 End Class

End Namespace
```

3. 添加对 MobiLink 服务器 API 的引用。
   - 从 Visual Studio 的 [**Project**] 菜单中，选择 [**Add Existing Item**]。
   - 在 *install-dir\Assembly\v2* 中选择 *iAnywhere.MobiLink.Script.dll*。在 Visual Studio 的 [**Open**] 菜单中，选择 [**Link File**]。在 Visual Studio 2005 的 [**Add**] 菜单中选择 [**Add Link**]。
4. 右击 CustdbScripts 项目并选择表 [**Properties**] » [**General**]。对于 Visual Basic .NET，请确保 [**Root Namespace**] 文本字段中没有任何文本。
5. 构建 *CustdbScripts.dll*。

   在 [**Build**] 菜单中，选择 [**Build CustdbScripts**]。

   将在 *C:\mldnet\CustdbScripts\CustdbScripts\bin\Debug* 目录中创建 *CustdbScripts.dll*。

♦ 在命令提示符中创建 CustdbScripts 程序集

1. 为 .NET 类和程序集创建一个目录。

   本教程假定路径为 *c:\mldnet*。
2. 使用文本编辑器输入 CustdbScripts 代码。

   如果是 C#，请键入：


```
namespace MLExample
{
class CustdbScripts
{
  public static string UploadInsert()
 {
   return("INSERT INTO ulcustomer(cust_id,cust_name) values (?,?)");
 }
 public static string DownloadCursor(System.DateTime ts, string user )
 {
   return("SELECT cust_id, cust_name FROM ULCustomer where last_modified
>= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff") +"'");
 }
}
}
```

如果是 Visual Basic .NET，请键入：

```
Namespace MLExample

 Class CustdbScripts

  Public Shared Function UploadInsert() As String
    Return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)")
  End Function

  Public Shared Function DownloadCursor(ByVal ts As System.DateTime, ByVal
user As String) As String
   Return("SELECT cust_id, cust_name FROM ULCustomer " + _
    "WHERE last_modified >= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff")
+"'")
  End Function

 End Class

End Namespace
```

3. 在 *c:\mldnet* 中，将文件保存为 *CustdbScripts.cs*（对于 Visual Basic .NET，保存为 *CustdbScripts.vb*）。
4. 使用以下命令编译该文件。

   如果是 C#，请键入：

   ```
   csc /out:c:\mldnet\custdbscripts.dll /target:library /
   reference:"%sqlany11%\Assembly\v2\iAnywhere.MobiLink.Script.dll" c:\mldnet
   \CustdbScripts.cs
   ```

   如果是 Visual Basic .NET，请键入：

   ```
   vbc /out:c:\mldnet\custdbscripts.dll /target:library /
   reference:"%sqlany11%\Assembly\v2\iAnywhere.MobiLink.Script.dll" c:\mldnet
   \CustdbScripts.vb
   ```

   将生成 *CustdbScripts.dll* 程序集。

进一步阅读

有关用于 .NET 的 MobiLink 服务器 API 的详细信息，请参见“用于 .NET 参考的 MobiLink 服务器 API”一节 《MobiLink - 服务器管理》。

有关 .NET 方法的详细信息，请参见“方法”一节 《MobiLink - 服务器管理》。

第 2 课：为事件指定类方法

有关 CustDB 示例数据库的详细信息，请参见“建立 CustDB 统一数据库”一节第 48 页。

上一课中创建的 *CustdbScripts.dll* 封装了 UploadInsert() 和 DownloadCursor() 方法。这些方法分别包含 ULCustomer upload_insert 和 download_cursor 事件的实现。

在本节中，可以用两种方法为表级事件指定类方法：

1. 使用 MobiLink 同步插件。

   使用 Sybase Central 连接到 CustDB 数据库，将 upload_insert 脚本的语言更改为 .NET，然后指定 MLExample.CustdbScripts.UploadInsert 来处理事件。

2. 使用 ml_add_dnet_table_script 存储过程。

   通过 Interactive SQL 连接到 CustDB 数据库并执行 ml_add_dnet_table_script，从而指定 MLExample.CustdbScripts.DownloadCursor 来处理 download_cursor 事件。

♦ 为 CustdbScripts.uploadInsert() 预订 ULCustomer 表的 upload_insert 事件

1. 使用 MobiLink 同步插件连接到示例数据库：
   a. 启动 Sybase Central。
   b. 在 [**视图**] 菜单中，确保已选择 [**文件夹**]。
   c. 在 [**连接**] 菜单中，选择 [**使用 MobiLink 11 连接**]。
   d. 在 [**标识**] 选项卡上，选择 [**ODBC 数据源名称**] SQL Anywhere 11 CustDB。
   e. 单击 [**确定**] 进行连接。
   f. 现在，Sybase Central 应在 MobiLink 11 插件下显示 CustDB 数据源。
2. 将 ULCustomer 表 upload_insert 事件的语言更改为 .NET：
   a. 在左窗格中，打开 [**同步表**] 文件夹，然后选择 ULCustomer 表。在右窗格中，出现一个表级别脚本列表。
   b. 在右窗格中，打开 custdb 11.0 upload_insert 表脚本。在 [**文件**] 菜单中，选择 [**语言**]，并将语言更改为 .NET。
3. 输入用于 custdb 11.0 upload_insert 表脚本的 .NET 方法名称。
   a. 删除 upload_insert 表脚本的内容，以确保右窗格显示为空白。
   b. 在右窗格中键入以下内容：

      ```
      MLExample.CustdbScripts.UploadInsert
      ```

   c. 从 [**文件**] 菜单中选择 [**保存**] 以保存脚本。
4. 退出 Sybase Central。

此步骤使用 Sybase Central 将 .NET 方法指定为 ULCustomer upload_insert 事件的脚本。

也可以使用 ml_add_dnet_connection_script 和 ml_add_dnet_table_script 存储过程。使用这些存储过程效率更高，特别是在使用大量 .NET 方法处理同步事件时。

请参见“ml_add_dnet_connection_script 系统过程”一节 《MobiLink - 服务器管理》和“ml_add_dnet_table_script 系统过程”一节 《MobiLink - 服务器管理》。

在下一节中，您将通过 Interactive SQL 连接到 CustDB 并执行 ml_add_dnet_table_script，从而将 MLExample.CustdbScripts.DownloadCursor 指派给 download_cursor 事件。

♦ 为 ULCustomer download_cursor 事件指定 MLExample.CustdbScripts.DownloadCursor

1. 使用 Interactive SQL 连接到示例数据库。
   a. 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Interactive SQL**]，或运行以下命令：

      ```
      dbisql
      ```

   b. 单击 [**标识**] 选项卡。
   c. 单击 [**ODBC 数据源名称**]，然后键入 SQL Anywhere 11 CustDB。
   d. 单击 [**确定**] 进行连接。
2. 在 Interactive SQL 中执行以下命令：

   ```
   CALL ml_add_dnet_table_script(
   'custdb 11.0',
   'ULCustomer',
   'download_cursor',
   'MLExample.CustdbScripts.DownloadCursor');
   COMMIT;
   ```

   以下是对每个参数的说明：

参数	说明
custdb 11.0	脚本版本。
ULCustomer	同步表。
download_cursor	事件名。
MLExample.CustdbScripts.DownloadCursor	完全限定 .NET 方法。

3. 退出 Interactive SQL。

在本课中，您指定了 .NET 方法来处理 ULCustomer 表级别事件。下一课将确保 MobiLink 服务器装载正确的类文件和 MobiLink 服务器 API。

进一步阅读

有关添加和删除同步脚本的详细信息，请参见“添加和删除脚本”一节 《MobiLink - 服务器管理》。

有关本课所用脚本的详细信息，请参见“ml_add_dnet_connection_script 系统过程”一节 《MobiLink - 服务器管理》和“ml_add_dnet_table_script 系统过程”一节 《MobiLink - 服务器管理》。

第 3 课：带 -sl dnet 运行 MobiLink

带 -sl dnet 选项运行 MobiLink 服务器可以指定 .NET 程序集的位置，并强制在服务器启动时装载 CLR。

如果使用 Visual Studio 进行编译，则 *CustdbScripts.dll* 的位置是 *c:\mldnet\CustdbScripts\CustdbScripts\bin\Debug*。如果在命令提示符中进行编译，则 *CustdbScripts.dll* 的位置是 *c:\mldnet*。

♦ 启动 MobiLink 服务器 (mlsrv11) 并装载 .NET 程序集

- 带 -sl dnet 选项启动 MobiLink 服务器。

  如果使用 Visual Studio 编译程序集：

  在命令提示符中，在一行上键入以下命令：

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -dl -o c:\mldnet\cons1.txt -v+ -sl
  dnet(-MLAutoLoadPath=c:\mldnet\CustdbScripts\CustdbScripts\bin\Debug)
  ```

  如果在命令提示符中编译程序集：

  在命令提示符中，在一行上键入以下命令：

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -dl -o c:\mldnet\cons1.txt -v+ -sl
  dnet(-MLAutoLoadPath=c:\mldnet)
  ```

  将出现一条消息，指明服务器已准备好处理请求。现在，如果同步过程中触发了 upload_insert 事件，便会执行 .NET 方法。

进一步阅读

有关详细信息，请参见“-sl dnet 选项”一节 《MobiLink - 服务器管理》。

第 4 课：测试同步

UltraLite 随附一个示例 Windows 客户端，当用户发出同步请求时，该客户端自动调用 dbmlsync 实用程序。它是一个简单的销售状态应用程序，您可以对上一课中启动的 CustDB 统一数据库运行该程序。

启动应用程序

♦ **启动并同步示例应用程序**

1. 选择 [开始] » [程序] » [**SQL Anywhere 11**] » [**UltraLite**] » [**Windows 示例应用程序**]。
2. 输入一个职员 ID。

   输入值 **50** 并按 Enter 键。

应用程序会自动进行同步，将一组客户、产品和订单从 CustDB 统一数据库下载到应用程序中。

在下一节中，您将输入新的客户名和订单详细信息。在随后进行的同步期间，这些信息将会上载到 CustDB 统一数据库中，并会触发 ULCustomer 表的 upload_insert 和 download_cursor 事件。

添加订单

♦ **添加订单：**

1. 从 [**Order**] 菜单中，选择 [**New**]。
2. 输入新的客户名称。例如，输入 **Frank DotNET**。
3. 选择一种产品，然后输入数量和折扣。
4. 按 Enter 键添加新订单。

您现在已经修改了本地 UltraLite 数据库中的数据。在进行同步之前，此数据没有与统一数据库共享。

♦ **与统一数据库进行同步并触发 upload_insert 事件**

● 从 [**File**] 菜单中选择 [**Synchronize**]。

  将出现一条消息，指明已将插入操作成功上载到了统一数据库。

进一步阅读

有关 CustDB Windows 应用程序的详细信息，请参见“研究 MobiLink 的 CustDB 示例”第 45 页。

清除

从计算机中删除教程资料。

♦ 从计算机中删除教程资料

1. 将 ULCustomer 表的 upload_insert 和 download_cursor 脚本恢复为其原始 SQL 逻辑。
   a. 打开 Interactive SQL。
      选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Interactive SQL**]，或运行以下命令：

      ```
      dbisql
      ```

   b. 单击 [**标识**] 选项卡。
   c. 单击 [**ODBC 数据源名称**]，然后键入 **SQL Anywhere 11 CustDB**。
   d. 单击 [**确定**] 进行连接。
   e. 在 Interactive SQL 中执行以下命令：

      ```
      CALL ml_add_table_script( 'custdb 11.0',
         'ULCustomer',
         'upload_insert',
         'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

      CALL ml_add_table_script( 'custdb 11.0',
         'ULCustomer',
         'download_cursor',
         'SELECT "cust_id", "cust_name"
          FROM "ULCustomer"
          WHERE "last_modified" >= ?' );
      ```

2. 关闭 SQL Anywhere 窗口、MobiLink 窗口和同步客户端窗口，方法是右击各个任务栏项并选择 [**关闭**]。
3. 删除所有与教程有关的 .NET 源文件。
   删除包含 *CustdbScripts.cs* 和 *CustdbScripts.dll* 文件的文件夹 (*c:\mldnet*)。

   注意
   确保 *c:\mldnet* 中没有其它重要文件。

4. 为 Windows 示例应用程序重置数据库。
   从 *samples-dir\UltraLite\CustDB* 目录运行以下命令：

   ```
   newdb.bat
   ```

进一步阅读

有关使用 .NET 编写 MobiLink 同步脚本的详细信息，请参见“设置 .NET 同步逻辑”一节 《MobiLink - 服务器管理》。

有关调试 .NET 同步逻辑的信息，请参见“调试 .NET 同步逻辑”一节 《MobiLink - 服务器管理》。

有关说明使用 .NET 同步脚本进行自定义验证的详细示例，请参见“.NET 同步示例”一节 《MobiLink - 服务器管理》。

有关同步脚本编写的详细信息，请参见“编写同步脚本” 《MobiLink - 服务器管理》和“同步事件” 《MobiLink - 服务器管理》。

有关其它同步方法（如时间戳）的介绍，请参见“同步技术” 《MobiLink - 服务器管理》。

教程：使用 .NET 和 Java 进行自定义验证

目录

MobiLink 自定义验证简介

您可以使用 SQL、Java 或 .NET 编写 MobiLink 同步脚本。您可以使用 Java 或 .NET 在同步的任意时刻添加自定义操作。

在本教程内，您将为 authenticate_user 连接事件添加 Java 或 .NET 方法。authenticate_user 事件允许您指定一个自定义验证方案并替换 MobiLink 内置客户端验证。

必需的软件

- SQL Anywhere 11
- Java 软件开发工具包

能力和经验

您需要：

- 熟悉 Java
- 了解 MobiLink 事件脚本的基本知识

目标

您将掌握并熟悉以下内容：

- MobiLink 自定义验证

重要概念

本节使用以下步骤通过 MobiLink CustDB 示例数据库实现基本的基于 Java 的同步：

- 使用 MobiLink 服务器 API 引用编译源文件
- 为特定表级别事件指定类方法
- 用 -sl java 选项运行 MobiLink 服务器 (mlsrv11)
- 使用示例 Windows 客户端应用程序测试同步

建议阅读的背景知识

有关验证 MobiLink 客户端的详细信息，请参见“选择用户验证机制”一节 《MobiLink - 客户端管理》。

有关集成 POP3、IMAP 或 LDAP 验证的详细信息，请参见“向外部服务器验证”一节 《MobiLink - 客户端管理》。

有关 .NET 或 Java 同步脚本的详细信息，请参见“使用 .NET 编写同步脚本” 《MobiLink - 服务器管理》或“使用 Java 语言编写同步脚本” 《MobiLink - 服务器管理》。

第 1 课：创建用于自定义验证的 Java 或 .NET 类（服务器端）

在本课中，您将编译用于自定义验证的包含 Java 或 .NET 逻辑的类。

面向 .NET 的 MobiLink 服务器 API

要执行 .NET 同步逻辑，MobiLink 服务器必须能够访问 *iAnywhere.MobiLink.Script.dll* 中的类。*iAnywhere.MobiLink.Script.dll* 包含要在 .NET 方法中使用的 MobiLink .NET 服务器 API 类的存储库。编译 .NET 类时，将引用 *iAnywhere.MobiLink.Script.dll*。

面向 Java 的 MobiLink 服务器 API

要执行 Java 同步逻辑，MobiLink 服务器必须能够访问 *mlscript.jar* 中的类。*mlscript.jar* 包含要在 Java 方法中使用的 MobiLink Java 服务器 API 类的存储库。编译 Java 类时，将引用 *mlscript.jar*。

♦ 创建用于自定义验证的 Java 或 .NET 类

1. 使用 Java 或 .NET 创建名为 MobiLinkAuth 的类。

   MobiLinkAuth 类包括用于 authenticate_user 同步事件的 authenticateUser 方法。authenticate_user 事件可为用户和口令提供参数。在 authentication_status inout 参数中返回验证结果。

   对于 Java，键入以下语句：

```
import ianywhere.ml.script.*;

public class MobiLinkAuth
{

 public void authenticateUser (
  ianywhere.ml.script.InOutInteger authentication_status,
  String user,
  String pwd,
  String newPwd )
 {
   // to do...

 }

}
```

   对于 .NET，键入以下语句：

```
using iAnywhere.MobiLink.Script;

public class MobiLinkAuth
{
  public void authenticateUser  (
    ref int authentication_status,
    string user,
    string pwd,
    string newPwd)
  {

   // to do...
```

```
    }
}
```

2. 编写 authenticateUser 方法。

   本教程说明一个非常简单的自定义用户验证的情况。如果用户名以 "ML" 开头，则验证成功。

   注意
   您将在“第 2 课：为 authenticate_user 事件注册 Java 或 .NET 脚本”一节第 172 页内注册 authenticate_user 同步事件的 authenticateUser 方法。

   对于 Java，键入以下语句：

```
public void authenticateUser (
 ianywhere.ml.script.InOutInteger authentication_status,
 String user,
 String pwd,
 String newPwd ) {

if (user.substring(0,2).equals("ML")) {
        // success: an auth status code of 1000
        authentication_status.setValue(1000);
    } else {
        // fail: an authentication_status code of 4000
        authentication_status.setValue(4000);
    }
}
```

   对于 .NET，键入以下语句：

```
public void authenticateUser(
 ref int authentication_status,
 string user,
 string pwd,
 string newPwd ) {

    if(user.Substring(0,2)=="ML") {
        // success: an auth status code of 1000
        authentication_status = 1000;
    } else {
        // fail: and authentication_status code of 4000
        authentication_status = 4000;
    }
}
```

3. 编译 MobiLinkAuth 类。

   - 对于 Java，在 *c:\mlauth* 中将文件保存为 *MobiLinkAuth.java*。浏览至 *c:\mlauth*。要编译文件，请运行以下命令：

     ```
     javac MobiLinkAuth.java -classpath "install-dir\java\mlscript.jar"
     ```

     将生成 *MobiLinkAuth.class* 文件。

   - 对于 .NET，在 *c:\mlauth* 中将文件保存为 *MobiLinkAuth.cs*。
   - 在命令提示符下，转到 *c:\mlauth*。要编译文件，请运行以下命令：

     ```
     csc /out:c:\mlauth\MobiLinkAuth.dll /target:library /
     reference:"%SQLANY11%\Assembly\v2\iAnywhere.MobiLink.Script.dll"
     MobiLinkAuth.cs
     ```


将生成 *MobiLinkAuth.dll* 程序集。

进一步阅读

有关 authenticate_user 事件（包括 authentication_status 返回代码表）的详细信息，请参见“authenticate_user 连接事件”一节 《MobiLink - 服务器管理》。

有关使用 Java 或 .NET 执行自定义验证的详细信息，请参见“Java 和 .NET 用户验证”一节 《MobiLink - 客户端管理》。

有关 Java 自定义验证的详细示例，请参见“Java 同步示例”一节 《MobiLink - 服务器管理》。

有关 .NET 自定义验证的详细示例，请参见“.NET 同步示例”一节 《MobiLink - 服务器管理》。

第 2 课：为 authenticate_user 事件注册 Java 或 .NET 脚本

在本课中，您将为 authenticate_user 同步事件注册 MobiLinkAuth authenticateUser 方法。您将把此脚本添加到 MobiLink 的示例数据库 CustDB 中。

MobiLink 数据库示例

SQL Anywhere 随附一个已针对同步进行了设置的 SQL Anywhere 示例数据库 (CustDB)。举例来说，CustDB ULCustomer 表是一个同步表，它支持多种表级别脚本。

CustDB 的设计用途是作为 UltraLite 和 SQL Anywhere 客户端的统一数据库服务器。CustDB 数据库有一个名为 SQL Anywhere 11 CustDB 的 DSN。

♦ 注册 authenticate_user 事件的 authenticateUser 方法

1. 使用 Interactive SQL 连接到示例数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=SQL Anywhere 11 CustDB"
   ```

2. 使用先前存储的 ml_add_java_connection_script 或 ml_add_dnet_connection_script 注册 authenticate_user 事件的 authenticateUser 方法。

   对于 Java，在 Interactive SQL 中执行以下命令：

   ```
   call ml_add_java_connection_script(
   'custdb 11.0',
   'authenticate_user',
   'MobiLinkAuth.authenticateUser');
   commit;
   ```

   对于 .NET，在 Interactive SQL 中执行以下命令：

   ```
   call ml_add_dnet_connection_script(
   'custdb 11.0',
   'authenticate_user',
   'MobiLinkAuth.authenticateUser');
   commit;
   ```

在下一课中，您将启动 MobiLink 服务器并加载您的类文件或程序集。

进一步阅读

有关添加和删除同步脚本的一般信息，请参见“添加和删除脚本”一节 《MobiLink - 服务器管理》。

有关 ml_add_java_connection_script 的详细信息，请参见“ml_add_java_connection_script 系统过程”一节 《MobiLink - 服务器管理》。

有关 ml_add_dnet_connection_script 的详细信息，请参见“ml_add_dnet_connection_script 系统过程”一节 《MobiLink - 服务器管理》

第 3 课：启动面向 Java 或 .NET 的 MobiLink 服务器

以 -sl java 或 -sl dnet 选项启动 MobiLink 服务器可允许您指定一组目录来搜索已编译的文件。

♦ 启动 MobiLink 服务器 (mlsrv11)

- 连接到 MobiLink 示例数据库，然后在 mlsrv11 命令行装载 Java 类或 .NET 程序集。

  对于 Java，运行以下命令：

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -sl java(-cp c:\mlauth)
  ```

  对于 .NET，运行以下命令：

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -sl dnet(-MLAutoLoadPath=c:\mlauth)
  ```

  当发生 authenticate_user 同步事件时，您的 Java 或 .NET 方法将会执行。

进一步阅读

有关启动面向 Java 的 MobiLink 服务器的详细信息，请参见“-sl java 选项”一节 《MobiLink - 服务器管理》。

有关启动面向 .NET 的 MobiLink 服务器的详细信息，请参见“-sl dnet 选项”一节 《MobiLink - 服务器管理》。

第 4 课：测试验证

UltraLite 随附一个示例 Windows 客户端，当用户发出同步请求时，该客户端自动调用 dbmlsync 实用程序。它是一个简单的销售状态应用程序，您可以对上一课中启动的 CustDB 统一数据库运行该程序。

♦ 启动示例应用程序并测试验证

1. 启动示例应用程序。

   从 [**开始**] 菜单中选择 [**程序**] » [**SQL Anywhere 11**] » [**UltraLite**] » [**Windows 示例应用程序**]。

2. 输入一个无效的雇员 ID 并进行同步。

   在此应用程序中，雇员 ID 也是 MobiLink 用户名。如果用户名不是以 "ML" 开头，则 Java 或 .NET 逻辑将使得同步失败。为雇员 ID 输入值 **50** 并按回车键。

   UltraLite CustDB Demo 窗口中将出现 SQL 代码 -103 同步错误，指明用户 ID 或口令无效。

进一步阅读

有关 CustDB Windows 应用程序的详细信息，请参见“研究 MobiLink 的 CustDB 示例”第 45 页。

清除

从计算机中删除教程资料。

♦ 从计算机中删除教程资料

1. 从统一数据库中删除 authenticate_user 脚本。
   a. 使用 Interactive SQL 连接到 MobiLink 示例数据库。
      运行以下命令：

      ```
      dbisql -c "dsn=SQL Anywhere 11 CustDB"
      ```

   b. 删除 authenticate_user 脚本。
      对于 Java，运行以下命令来删除 authenticate_user 脚本：

      ```
      call ml_add_java_connection_script(
      'custdb 11.0',
      'authenticate_user',
      null);
      commit;
      ```

      对于 .NET，运行以下命令来删除 authenticate_user 脚本：

      ```
      call ml_add_dnet_connection_script(
      'custdb 11.0',
      'authenticate_user',
      null);
      commit;
      ```

2. 删除 Java 或 .NET 源文件。
   例如，删除 *c:\mlauth* 目录。

   小心
   确保在此目录中只有与教程相关的资料。

3. 关闭 Interactive SQL 和 UltraLite Windows 客户端应用程序。
   从每个应用程序的 [**文件**] 菜单中选择 [**退出**]。
4. 关闭 SQL Anywhere、MobiLink 和同步客户端窗口。
   右击每个任务栏项并选择 [**关闭**]。
5. 为 Windows 示例应用程序重置数据库。
   从 *samples-dir\UltraLite\CustDB* 目录运行以下命令：

   ```
   newdb.bat
   ```

进一步阅读

有关 Java 同步逻辑的详细信息，请参见“使用 Java 语言编写同步脚本” 《MobiLink - 服务器管理》。

有关 .NET 同步逻辑的详细信息，请参见“使用 .NET 编写同步脚本” 《MobiLink - 服务器管理》。

有关说明自定义验证的 Java 或 .NET 同步脚本用法的详细示例，请分别参见“Java 同步示例”一节 《MobiLink - 服务器管理》或“.NET 同步示例”一节 《MobiLink - 服务器管理》。

有关调试 Java 或 .NET 同步逻辑的信息，请分别参见“调试 Java 类”一节 《MobiLink - 服务器管理》或“调试 .NET 同步逻辑”一节 《MobiLink - 服务器管理》。

有关同步脚本的详细信息，请参见“编写同步脚本” 《MobiLink - 服务器管理》和“同步事件” 《MobiLink - 服务器管理》。

教程：直接行处理简介

目录

直接行处理教程简介

本教程介绍如何实现 MobiLink 直接行处理，以便您可使用支持的统一数据库以外的其它数据源。

可以使用直接行处理实现远程数据与任何中央数据源、应用程序或 Web 服务之间的通信。

本教程介绍如何将面向 Java 和 .NET 的 MobiLink 服务器 API 用于简单直接行处理。将客户端 RemoteOrders 表与统一数据库同步，并为 OrderComments 表添加直接行处理的特殊处理。

您可以为 RemoteOrders 表建立一个简单的同步。

必需的软件

- Java 软件开发工具包或 Microsoft .NET Framework

能力和经验

您需要：

- 熟悉 Java 或 .NET。
- 了解 MobiLink 事件脚本和 MobiLink 同步的基本知识。

目标

您将掌握并熟悉以下内容：

- 面向 Java 和 .NET 的 MobiLink 服务器 API。
- 创建适用于 MobiLink 直接行处理的方法。

建议阅读的背景知识

- “了解 MobiLink 同步”第 3 页
- “同步技术” 《MobiLink - 服务器管理》
- “直接行处理” 《MobiLink - 服务器管理》

MobiLink 代码交换示例位于 http://www.sybase.com/detail?id=1058600#319。

可在 MobiLink 新闻组上发布问题：sybase.public.sqlanywhere.mobilink。

第 1 课：建立 MobiLink 统一数据库

MobiLink 统一数据库是数据的中央存储库，包括用来管理同步过程的 MobiLink 系统表和存储过程。使用直接行处理，可与统一数据库以外的其它数据源同步，但您仍需要统一数据库来维护 MobiLink 服务器所使用的信息。

在本课中，您将：

- 创建数据库并定义 ODBC 数据源。
- 添加数据表以同步远程客户端。
- 安装 MobiLink 系统表和存储过程。

注意
如果您已建立了具有 MobiLink 系统对象和 DSN 的 MobiLink 统一数据库，则可跳过本课。

创建统一数据库

在本教程中，您将使用 Sybase Central 的 [**创建数据库向导**] 来创建 SQL Anywhere 数据库。

♦ **创建 SQL Anywhere RDBMS**

1. 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 在 Sybase Central 中，选择 [**工具**] » [**SQL Anywhere 11**] » [**创建数据库**]。
3. 单击 [**下一步**]。
4. 保留缺省设置 [**在这台计算机上创建数据库**]。单击 [**下一步**]。
5. 在 [**将主数据库文件保存到以下文件**] 字段中键入数据库的文件名和路径。例如，*c:\MLdirect\MLconsolidated.db*。单击 [**下一步**]。
6. 请按照 [**创建数据库向导**] 中的其余说明进行操作并接受缺省值。在 [**连接到数据库**] 页面上，清除 [**最后一次断开连接后停止数据库**] 选项。
7. 单击 [**完成**]。

   在 Sybase Central 中就会出现名为 MLconsolidated 的数据库。

为统一数据库定义 ODBC 数据源。

使用 SQL Anywhere 11 驱动程序为 MLconsolidated 数据库定义 ODBC 数据源。

♦ **为统一数据库定义 ODBC 数据源**

1. 在 Sybase Central 中，选择 [**工具**] » [**SQL Anywhere 11**] » [**打开 ODBC 管理器**]。
2. 单击 [**用户 DSN**] 选项卡，然后单击 [**添加**]。
3. 在 [**名称**] 列表中，单击 [**SQL Anywhere 11**]。单击 [**完成**]。
4. 在 [**SQL Anywhere 11 的 ODBC 配置**] 窗口中，进行以下操作：
   a. 单击 [**ODBC**] 选项卡。

b. 在 [**数据源名**] 字段中，键入 **mobilink_db**。
c. 单击 [**登录**] 选项卡。
d. 在 [**用户 ID**] 字段键入 **DBA**。
e. 在 [**口令**] 字段中键入 **sql**。
f. 单击 [**数据库**] 选项卡。
g. 在 [**服务器名**] 字段中键入 **MLconsolidated**。
h. 在 [**数据库文件**] 字段中，键入 *c:\MLdirect\MLconsolidated.db*。
i. 单击 [**确定**]。

5. 单击 [**确定**]。

为同步创建表

在此过程中，在 MobiLink 统一数据库中创建 RemoteOrders 表。RemoteOrders 表包含以下各列：

列	说明
order_id	订单的唯一标识符。
product_id	产品的唯一标识符。
quantity	销售项目的数量。
order_status	订单状态。
last_modified	行上次修改的日期。此列用于基于时间戳的下载，这是一种为提高同步效率而过滤行的常用技术。

♦ 创建 RemoteOrders 表

1. 使用 Interactive SQL 连接到数据库。

   在命令提示符处，运行以下命令：

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. 在 Interactive SQL 中运行以下命令创建 RemoteOrders 表。

   ```
   CREATE TABLE RemoteOrders
   (
    order_id           integer not null,
    product_id         integer not null,
    quantity           integer,
    order_status       varchar(10) default 'new',
    last_modified      timestamp default current timestamp,
    primary key(order_id)
   )
   ```

   在 Interactive SQL 中为以下过程保持连接。

运行 MobiLink 安装脚本

可以在 SQL Anywhere 11 安装目录的 *MobiLink/setup* 子目录中查找每个支持的统一数据库的安装脚本。

在此过程中建立 SQL Anywhere 统一数据库。可以使用 *syncsa.sql* 安装脚本来执行这一操作。运行 *syncsa.sql* 时会创建一系列以 **ml_** 为前缀的系统表和存储过程。MobiLink 服务器在同步过程中会使用这些表和存储过程。

♦ 安装 MobiLink 系统表

1. 如果您尚未建立连接，则在 Interactive SQL 中连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. 在 Interactive SQL 中运行以下命令来创建 MobiLink 系统表和存储过程。用 SQL Anywhere 11 安装目录的位置来替换 *c:\Program Files\SQL Anywhere 11*。

   ```
   read "c:\Program Files\SQL Anywhere 11\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL 将 *syncsa.sql* 应用到您的统一数据库。

在 Interactive SQL 中为下一课保持连接。

进一步阅读

有关创建 SQL Anywhere 数据库的信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关创建表的信息，请参见“CREATE TABLE 语句”一节 《SQL Anywhere 服务器 - SQL 参考》。

有关建立 MobiLink 统一数据库的信息，请参见“MobiLink 统一数据库” 《MobiLink - 服务器管理》。

第 2 课：添加同步脚本

在本课中，您将为 SQL 行处理和直接行处理，将脚本添加到统一数据库中。

SQL 行处理

SQL 行处理可用于将远程数据与 MobiLink 统一数据库中的表同步。基于 SQL 的脚本可定义：

- 如何将从 MobiLink 客户端上载的数据应用到统一数据库。
- 应从统一数据库下载哪些数据。

在本课中，您将为以下基于 SQL 的上载和下载事件编写同步脚本。

- **upload_insert** 对插入远程客户端数据库的新订单如何应用到统一数据库进行定义。
- **download_cursor** 对 MobiLink 统一数据库中更新的哪些订单应下载到远程客户端进行定义。

在此过程中，可以使用存储过程将同步脚本信息添加到 MobiLink 统一数据库。

♦ 将基于 SQL 的脚本添加到 MobiLink 系统表

1. 如果您尚未建立连接，则在 Interactive SQL 中连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. 使用 ml_add_table_script 存储过程为 upload_insert 和 download_cursor 事件添加基于 SQL 的表脚本。

   在 Interactive SQL 中运行以下命令。upload_insert 脚本将上载的 order_id、product_id、quantity 和 order_status 插入到 MobiLink 统一数据库。download_cursor 脚本使用基于时间戳的过滤将更新的行下载到远程客户端。

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'upload_insert',
        'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
         VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'download_cursor',
        'SELECT order_id, product_id, quantity, order_status
         FROM RemoteOrders WHERE last_modified >= ?');

   commit
   ```

直接行处理处理

在本教程中，您将使用直接行处理将特殊处理添加到基于 SQL 的同步系统中。在此过程中，您将注册与 handle_UploadData、handle_DownloadData 和 end_download 事件对应的方法名。在“第 3 课：为处理直接行处理编写 Java 或 .NET 逻辑”一节第 185 页中创建您自己的 java 或 .NET 类。

♦ 在 MobiLink 系统表中添加直接行处理的信息

1. 在 Interactive SQL 中，连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. 注册用于 end_download 事件的 Java 或 .NET 方法。

   当触发 end_download 连接事件时，使用此方法释放 i/o 资源。

   对于 Java，在 Interactive SQL 中执行以下命令。

   ```
   CALL ml_add_java_connection_script( 'default',
      'end_download',
        'MobiLinkOrders.EndDownload' );
   ```

   对于 .NET，在 Interactive SQL 中执行以下命令。

   ```
   CALL ml_add_dnet_connection_script( 'default',
      'end_download',
        'MobiLinkOrders.EndDownload' );
   ```

   Interactive SQL 注册用于 end_download 事件的用户定义的 EndDownload 方法。

3. 注册用于 handle_UploadData 和 handle_DownloadData 同步事件的 Java 或 .NET 方法。

   对于 Java，在 Interactive SQL 中执行以下命令。

   ```
   CALL ml_add_java_connection_script( 'default',
      'handle_UploadData',
        'MobiLinkOrders.GetUpload' );

   CALL ml_add_java_connection_script( 'default',
      'handle_DownloadData',
        'MobiLinkOrders.SetDownload' );

   commit
   ```

   对于 .NET，在 Interactive SQL 中执行以下命令。

   ```
   CALL ml_add_dnet_connection_script( 'default',
      'handle_UploadData',
        'MobiLinkOrders.GetUpload' );

   CALL ml_add_dnet_connection_script( 'default',
      'handle_DownloadData',
        'MobiLinkOrders.SetDownload' );

   commit
   ```

   Interactive SQL 分别注册用于 handle_UploadData 和 handle_DownloadData 事件的用户定义的 GetUpload 和 SetDownload 方法。

进一步阅读

有关使用基于 SQL 的事件将数据从远程客户端上载到 MobiLink 统一数据库的信息，请参见：

- “编写用于上载行的脚本”一节 《MobiLink - 服务器管理》
- “upload_insert 表事件”一节 《MobiLink - 服务器管理》
- “upload_update 表事件”一节 《MobiLink - 服务器管理》
- “upload_delete 表事件”一节 《MobiLink - 服务器管理》

有关将数据上载到除统一数据库之外的其它数据源的信息，请参见“处理直接上载”一节 《MobiLink - 服务器管理》。

有关使用基于 SQL 的事件从 MobiLink 统一数据库下载数据的信息，请参见：

- “编写用于下载行的脚本”一节 《MobiLink - 服务器管理》
- “download_cursor 表事件”一节 《MobiLink - 服务器管理》
- “download_delete_cursor 表事件”一节 《MobiLink - 服务器管理》

有关将数据下载到除统一数据库之外的其它数据源的信息，请参见“处理直接下载”一节 《MobiLink - 服务器管理》。

有关同步事件序列的信息，请参见“MobiLink 事件概述”一节 《MobiLink - 服务器管理》。

有关下载过滤的同步技术的信息，请参见“基于时间戳的下载”一节 《MobiLink - 服务器管理》和“在远程数据库之间对行进行分区”一节 《MobiLink - 服务器管理》。

有关管理脚本的信息，请参见“添加和删除脚本”一节 《MobiLink - 服务器管理》。

有关直接行处理的信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

第 3 课：为处理直接行处理编写 Java 或 .NET 逻辑

在本课中，您将使用直接行处理来处理客户端数据库的 OrderComments 表中的行。为直接行处理添加以下方法：

- **GetUpload** 将此方法用于 handle_UploadData 事件。GetUpload 将上载的注释写入名为 *orderComments.txt* 的文件中。
- **SetDownload** 将此方法用于 handle_DownloadData 事件。SetDownload 使用 *orderResponses.txt* 文件来下载对远程客户端的响应。

还可以添加 EndDownload 方法来处理 end_download 事件。

以下过程介绍如何创建 Java 或 .NET 类，其中包括用于处理的方法。有关完整列表，请参见“完整的 MobiLinkOrders 列表 (Java)”一节第 191 页或“完整的 MobiLinkOrders 列表 (.NET)”一节第 193 页。

♦ 为直接行处理创建 Java 或 .NET 类

1. 使用 Java 或 .NET 创建名为 MobiLinkOrders 的类。

   对于 Java，在文本编辑器或开发环境中键入以下代码。

   ```
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   public class  MobiLinkOrders{

   // to do...
   ```

   对于 .NET，使用以下代码：

   ```
   using iAnywhere.MobiLink.Script;
   using System.IO;
   using System.Data;
   using System.Text;

   public class MobiLinkOrders

   // to do...
   ```

2. 声明一个类级别 DBConnectionContext 实例。

   在 Java 中：

   ```
   // class level DBConnectionContext
       DBConnectionContext _cc;
   ```

   在 .NET 中：

   ```
   // class level DBConnectionContext
       private DBConnectionContext _cc = null;
   ```

   MobiLink 服务器会将 DBConnectionContext 实例传递到类构造函数。DBConnectionContext 封装有关与 MobiLink 统一数据库的当前连接的信息。

3. 创建类构造函数。

   类构造函数设置类级别 DBConnectionContext 实例。

   对于 Java：

   ```
   public MobiLinkOrders( DBConnectionContext cc )
   {
    // set your class-level DBConnectionContext
    _cc = cc;
   }
   ```

   对于 .NET：

   ```
   public MobiLinkOrders( DBConnectionContext cc )
   {
    _cc = cc;
   }
   ```

4. 声明用于文件输入和输出的对象。

   对于 Java，声明 java.io.FileWriter 和 java.io.BufferedReader：

   ```
   // java objects for file i/o
       FileWriter my_writer;
       BufferedReader my_reader;
   ```

   对于 .NET，声明流写入器和流读取器：

   ```
   // instances for file I/O
   private static StreamWriter   my_writer = null;
   private static StreamReader   my_reader = null;
   ```

5. 编写 EndDownload 方法。

   此方法处理 end_download 连接事件并提供释放资源的机会。

   对于 Java：

   ```
   public void EndDownload() throws IOException
   {
     // free i/o resources
       if (my_reader!=null) my_reader.close();
       if (my_writer!=null) my_writer.close();
   }
   ```

   对于 .NET：

   ```
   public void EndDownload()
   {
     if( my_writer != null ) {
        my_writer.Close();
        my_writer = null;
     }
   ```

6. 编写 GetUpload 方法

   GetUpload 方法获得表示 OrderComments 表的 UploadedTableData 类实例。OrderComments 表包含由远程销售雇员进行的特殊注释。您将在“第 5 课：建立 MobiLink 客户端”一节第 197 页中创建该表。UploadedTableData getInserts 方法返回新订单注释的结果集。writeOrderComment 方法将结果集中的每行写出至文本文件。

对于 Java：

```
//method for the handle_UploadData synchronization event
public void GetUpload( UploadData ut ) throws SQLException, IOException
    {
   // get an UploadedTableData for OrderComments
   UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

   // get inserts uploaded by the MobiLink client
   ResultSet insertResultSet = orderCommentsTbl.getInserts();

   while( insertResultSet.next() ) {

      // get order comments
      int _commentID = insertResultSet.getInt("comment_id");
      int _orderID = insertResultSet.getInt("order_id");
      String _specialComments =
insertResultSet.getString("order_comment");

      if (_specialComments != null)
      {
         writeOrderComment(_commentID,_orderID,_specialComments);

      }
            }
   insertResultSet.close();

   }

// writes out comment details to file
public void writeOrderComment( int _commentID, int _orderID, String
_comments )
   throws IOException
    {

      // a FileWriter for writing order comments
      if(my_writer == null)
      {
        my_writer = new FileWriter( "C:\\MLdirect\
\orderComments.txt",true);
      }

      // write out the order comments to remoteOrderComments.txt
      my_writer.write(_commentID + "\t" + _orderID + "\t" + _comments);
      my_writer.write( "\n" );
      my_writer.flush();

    }
```

在 .NET 中：

```
// method for the handle_UploadData synchronization event.
public void GetUpload( UploadData ut )
    {
 // get UploadedTableData for remote table called OrderComments
 UploadedTableData order_comments_table_data =
ut.GetUploadedTableByName( "OrderComments" );

 // get inserts upload by the MobiLink client
 IDataReader new_comment_reader = order_comments_table_data.GetInserts();

 while( new_comment_reader.Read() ) {
```

```
        // columns are
        // 0 - "order_comment"
        // 1 - "comment_id"
        // 2 - "order_id"
        // you can look up these values using the DataTable returned by:
        // order_comments_table_data.GetSchemaTable() if the send column
names
        // option is turned on the remote.
        // in this example you just use the known column order to determine
the column
        // indexes

        // only process this insert if the order_comment is not null
        if( !new_comment_reader.IsDBNull( 2 ) ) {
  int comment_id = new_comment_reader.GetInt32( 0 );
  int order_id   = new_comment_reader.GetInt32( 1 );
  string comments= new_comment_reader.GetString( 2 );
  WriteOrderComment( comment_id, order_id, comments );
        }
 }
 // always close the reader when you are done with it!
 new_comment_reader.Close();
     }
```

7. 编写 SetDownload 方法：

   a. 获得表示 OrderComments 表的类实例。

   使用 DBConnectionContext getDownloadData 方法可获得 DownloadData 实例。使用 DownloadData getDownloadTableByName 方法可为 OrderComments 表返回 DownloadTableData 实例。

   对于 Java：

```
DownloadData download_d = _cc.getDownloadData();
DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );
```

   对于 .NET：

```
DownloadTableData comments_for_download =
    _cc.GetDownloadData().GetDownloadTableByName( "OrderComments" );
```

   注意
   您将在“第 5 课：建立 MobiLink 客户端”一节第 197 页中于远程数据库上创建此表。

   b. 获得允许您将插入或更新操作添加到下载的准备好的语句或者 IDBCommand。

   对于 Java，使用 DownloadTableData getUpsertPreparedStatement 方法来返回一个 java.sql.PreparedStatement 实例。

```
PreparedStatement update_ps = download_td.getUpsertPreparedStatement();
```

   对于 .NET，使用 DownloadTableData GetUpsertCommand 方法：

```
// add upserts to the set of operation that are going to be applied at
the
// remote database
IDbCommand  comments_upsert = comments_for_download.GetUpsertCommand();
```

   c. 将下载响应发送到远程客户端。

在 *c:\MLdirect* 中创建名为 *orderResponses.txt* 的文本文件。此文件包含对注释的响应。例如，*orderResponses.txt* 可以包含以下含有表示 comment_id、order_id 和 order_comment 的制表符分隔的值的条目。

```
...
786  34  OK, we will ship promotional material.
787  35  Yes, the product is going out of production.
788  36  No, we can't increase your commission...
...
```

d. 设置每行的下载数据。

对于 Java，以下示例遍历 *orderResponses.txt* 并将数据添加到 MobiLink 下载。

对于 Java：

```
//   a BufferedReader for reading in responses
if (my_reader==null)
 my_reader = new BufferedReader(new FileReader( "c:\\MLdirect\
\orderResponses.txt"));

// send updated comments down to clients
String commentLine;

// read the first line from orderResponses.txt
commentLine = my_reader.readLine();

   while(commentLine != null)
{
   // get the next line from orderResponses.txt
   String[] response_details = commentLine.split("\t");

   if (response_details.length != 3)
         {
            System.err.println("Error reading from
orderResponses.txt");
            System.err.println("Error setting direct row handling
download");
            return;
         }

   int comment_id = Integer.parseInt(response_details[0]);
   int order_id = Integer.parseInt(response_details[0]);
   String updated_comment = response_details[2];

   // set an order comment response in the MobiLink download
update_ps.setInt(1, comment_id);
   update_ps.setInt(2, order_id);
   update_ps.setString(3, updated_comment);
    update_ps.executeUpdate();

   // get next line from orderResponses.txt
         commentLine  = my_reader.readLine();
      }
```

对于 .NET：

```
string comment_line;
while( (comment_line = my_reader.ReadLine()) != null) {
     // three values are on each line separated by '\t'
     string[] response_details = comment_line.Split( '\t' );
     if( response_details.Length != 3 ) {
throw( new SynchronizationException( "Error reading from
```

```
orderResponses.txt" ) );
        }
        int comment_id = System.Int32.Parse( response_details[0] );
        int order_id   = System.Int32.Parse( response_details[1] );
        string comments= response_details[2];

        // Parameters of the correct number and type have already been
added
        // so you just need to set the values of the IDataParameters
        ((IDataParameter)(comments_upsert.Parameters[0])).Value =
comment_id;
        ((IDataParameter)(comments_upsert.Parameters[1])).Value =
order_id;
        ((IDataParameter)(comments_upsert.Parameters[2])).Value =
comments;
        // add the upsert operation
        comments_upsert.ExecuteNonQuery();
    }
```

e. 关闭用来将插入操作或更新操作添加到下载的预准备语句。

对于 Java：

```
update_ps.close();
```

对于 .NET，不必关闭 IDBCommand。MobiLink 会在下载结束时将其取消。

8. 编译您的类文件。

- 浏览到包含 Java 或 .NET 源文件的目录。
- 编译 MobiLinkOrders 时引用面向 Java 或 .NET 的 MobiLink 服务器 API 库。

对于 Java，需要引用位于 *install-dir\java* 中的 *mlscript.jar*。运行以下命令来编译 Java 类：

```
javac -classpath "%SQLANY11%\java\mlscript.jar" MobiLinkOrders.java
```

对于 .NET，运行以下命令：

```
csc /out:MobiLinkServerCode.dll /target:library /reference:"%SQLANY11%
\Assembly\v2\iAnywhere.MobiLink.Script.dll" MobiLinkOrders.cs
```

注意

此处显示的示例并未确保主键值是唯一的。请参见“维护唯一主键”一节 《MobiLink - 服务器管理》。

进一步阅读

有关类构造函数和 DBConnectionContext 的详细信息，请参见：

- .NET 同步逻辑： “构造函数”一节 《MobiLink - 服务器管理》
- Java 同步逻辑： “构造函数”一节 《MobiLink - 服务器管理》

有关 Java 同步逻辑的详细信息，请参见“使用 Java 语言编写同步脚本” 《MobiLink - 服务器管理》。

有关 .NET 同步逻辑的详细信息，请参见“使用 .NET 编写同步脚本” 《MobiLink - 服务器管理》。

有关直接行处理的详细信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

完整的 MobiLinkOrders 列表 (Java)

以下是适用于 Java 直接行处理的完整的 MobiLinkOrders 列表。有关分步说明，请参见“第 3 课：为处理直接行处理编写 Java 或 .NET 逻辑”一节第 185 页。

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class MobiLinkOrders {

    // class level DBConnectionContext
    DBConnectionContext _cc;

    // java objects for file i/o
    FileWriter my_writer;
    BufferedReader my_reader;
    public MobiLinkOrders( DBConnectionContext cc )
        throws IOException, FileNotFoundException
    {
        // declare a class-level DBConnectionContext
        _cc = cc;
    }

    public void writeOrderComment( int _commentID, int _orderID, String
_comments )
        throws IOException
    {
        if (my_writer == null)
            // a FileWriter for writing order comments
            my_writer = new FileWriter( "C:\\MLdirect\
\orderComments.txt",true);

        // write out the order comments to remoteOrderComments.txt
        my_writer.write(_commentID + "\t" + _orderID + "\t" + _comments);
        my_writer.write( "\n" );
        my_writer.flush();
    }

    //  method for the handle_UploadData synchronization event
    public void GetUpload( UploadData ut )
        throws SQLException, IOException
    {
        //  get an UploadedTableData for OrderComments
        UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

        // get inserts uploaded by the MobiLink client
        ResultSet insertResultSet = orderCommentsTbl.getInserts();

        while ( insertResultSet.next() )
        {
            // get order comments
            int _commentID = insertResultSet.getInt("comment_id");
            int _orderID = insertResultSet.getInt("order_id");
            String _specialComments =
insertResultSet.getString("order_comment");
            if (_specialComments != null)
            {
                writeOrderComment(_commentID,_orderID,_specialComments);
            }
        }
```

```
            insertResultSet.close();
        }

        public void SetDownload()
            throws SQLException, IOException
        {
            DownloadData download_d = _cc.getDownloadData();

            DownloadTableData download_td =
    download_d.getDownloadTableByName( "OrderComments" );

            PreparedStatement update_ps =
    download_td.getUpsertPreparedStatement();
            // a BufferedReader for reading in responses
            if (my_reader == null)
                my_reader = new BufferedReader(new FileReader( "c:\\MLdirect\
    \orderResponses.txt"));

            // get the next line from orderResponses
            String commentLine;
            commentLine = my_reader.readLine();

            // send comment responses down to clients
            while (commentLine != null)
            {
                // get the next line from orderResponses.txt
                String[] response_details = commentLine.split("\t");

                if (response_details.length != 3)
                {
                    System.err.println("Error reading from orderResponses.txt");
                    System.err.println("Error setting direct row handling
    download");
                    return;
                }
                int comment_id = Integer.parseInt(response_details[0]);
                int order_id = Integer.parseInt(response_details[1]);
                String updated_comment = response_details[2];

                // set an order comment response in the MobiLink download
                update_ps.setInt(1, comment_id);
                update_ps.setInt(2, order_id);
                update_ps.setString(3, updated_comment);
                update_ps.executeUpdate();

                // get next line
                commentLine = my_reader.readLine();
            }
            update_ps.close();
        }

        public void EndDownload()
            throws IOException
        {
            // close i/o resources
            if (my_reader != null) my_reader.close();
            if (my_writer != null) my_writer.close();
        }
    }
```


完整的 MobiLinkOrders 列表 (.NET)

以下是适用于 .NET 基于对象的上载和下载的完整的 MobiLinkOrders 列表。有关分步说明，请参见“第 3 课：为处理直接行处理编写 Java 或 .NET 逻辑”一节第 185 页。

```
using iAnywhere.MobiLink.Script;
using System.IO;
using System.Data;
using System.Text;

public class MobiLinkOrders
{
    // class level DBConnectionContext
    private DBConnectionContext _cc = null;

    // instances for file I/O
    private static StreamWriter my_writer = null;
    private static StreamReader my_reader = null;


    public MobiLinkOrders(DBConnectionContext cc)
    {
        _cc = cc;
    }

    public void WriteOrderComment(int comment_id,
        int order_id,
        string comments)
    {
        if (my_writer == null)
        {
            my_writer = new StreamWriter(
                "c:\\MLdirect\\orderComments.txt");
        }
        my_writer.WriteLine("{0}\t{1}\t{2}",
            comment_id, order_id, comments);
        my_writer.Flush();
    }

    // method for the handle_UploadData synchronization event.
    public void GetUpload(UploadData ut)
    {
        // get UploadedTableData for remote table called OrderComments
        UploadedTableData order_comments_table_data =
            ut.GetUploadedTableByName("OrderComments");

        // get inserts uploaded by the MobiLink client
        IDataReader new_comment_reader =
            order_comments_table_data.GetInserts();

        while (new_comment_reader.Read())
        {
            // Columns are
            // 0 - "order_comment"
            // 1 - "comment_id"
            // 2 - "order_id"
            // You can look up these values using the DataTable returned by:
            // order_comments_table_data.GetSchemaTable() if the send
            // column names option is turned on at the remote.
            // In this example, you just use the known column order to
            // determine the column indexes
```

```
            // Only process this insert if the order_comment is not null
            if (!new_comment_reader.IsDBNull(2))
            {
                int comment_id = new_comment_reader.GetInt32(0);
                int order_id = new_comment_reader.GetInt32(1);
                string comments = new_comment_reader.GetString(2);
                WriteOrderComment(comment_id, order_id, comments);
            }
        }
        // Always close the reader when you are done with it!
        new_comment_reader.Close();
    }

    private const string read_file_path =
        "c:\\MLdirect\\orderResponses.txt";

    // method for the handle_DownloadData synchronization event
    public void SetDownload()
    {
        if ((my_reader == null) && !File.Exists(read_file_path))
        {
            System.Console.Out.Write("There is no file to read.");
            return;
        }
        DownloadTableData comments_for_download =
            _cc.GetDownloadData().GetDownloadTableByName("OrderComments");

        // Add upserts to the set of operation that are going to be
        // applied at the remote database
        IDbCommand comments_upsert =
            comments_for_download.GetUpsertCommand();

        if (my_reader == null)
        {
            my_reader = new StreamReader(read_file_path);
        }
        string comment_line;
        while ((comment_line = my_reader.ReadLine()) != null)
        {
            // three values are on each line separated by '\t'
            string[] response_details = comment_line.Split('\t');
            if (response_details.Length != 3)
            {
                throw (new SynchronizationException(
                    "Error reading from orderResponses.txt"));
            }
            int comment_id = System.Int32.Parse(response_details[0]);
            int order_id = System.Int32.Parse(response_details[1]);
            string comments = response_details[2];

            // Parameters of the correct number and type have
            // already been added so you just need to set the
            // values of the IDataParameters
            ((IDataParameter)(comments_upsert.Parameters[0])).Value =
                comment_id;
            ((IDataParameter)(comments_upsert.Parameters[1])).Value =
                order_id;
            ((IDataParameter)(comments_upsert.Parameters[2])).Value =
                comments;
            // Add the upsert operation
            comments_upsert.ExecuteNonQuery();
        }
    }
```



```
        public void EndDownload()
        {
            if (my_writer != null)
            {
                my_writer.Close();
                my_writer = null;
            }
            if (my_reader != null)
            {
                my_reader.Close();
                my_reader = null;
            }
        }
    }
```

第 4 课：启动 MobiLink 服务器

在本课中，您将启动 MobiLink 服务器。使用 -c 选项启动 MobiLink 服务器 (mlsrv11) 以连接到统一数据库。使用 -sl java 或 -sl dnet 选项装载 Java 或 .NET 类。

♦ 为直接行处理启动 MobiLink 服务器

- 连接到统一数据库，然后在 mlsrv11 命令行处装载 .NET 或 Java 类。

  对于 Java，运行以下命令。用 Java 源文件的位置替换 *c:\MLdirect*。

  ```
  mlsrv11 -c "dsn=mobilink_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip -sl
  java (-cp c:\MLdirect)
  ```

  对于 .NET：

  ```
  mlsrv11 -c "dsn=mobilink_db;uid=DBA;pwd=sql" -o serverOut.txt -v+ -dl -zu+
  -x tcpip -sl dnet (-MLAutoLoadPath=c:\MLdirect)
  ```

  即会出现 MobiLink 服务器窗口。

  下面对本教程中使用的每个 MobiLink 服务器选项均加以说明。选项 -o、-v 和 -dl 提供调试和疑难解答信息。适宜在开发环境中使用这些记录选项。出于性能原因，-v 和 -dl 通常不在生产中使用。

选项	说明
-c	引出连接字符串。
-o	指定消息日志文件 *serverOut.txt*。
-v+	-v 选项指定记录哪些信息。使用 -v+ 设置最大详细记录。
-dl	在屏幕上显示所有日志消息。
-zu+	自动添加新用户。
-x	为 MobiLink 客户端设置通信协议和参数。
-sl java	指定一组目录以便用来搜索类文件，并强制在服务器启动时装载 Java 虚拟机。
-sl dnet	指定 .NET 程序集的位置，并强制在服务器启动时装载 CLR。

进一步阅读

有关 MobiLink 服务器选项的完整列表，请参见“MobiLink 服务器选项” 《MobiLink - 服务器管理》。

有关装载 Java 和 .NET 类的详细信息，请分别参见“-sl java 选项”一节 《MobiLink - 服务器管理》和“-sl dnet 选项”一节 《MobiLink - 服务器管理》。

第 5 课：建立 MobiLink 客户端

在本教程中，您将针对统一数据库和 MobiLink 客户端使用 SQL Anywhere 数据库。用于教程目的时，MobiLink 客户端、统一数据库和 MobiLink 服务器均驻留在同一台计算机上。

若要建立 MobiLink 客户端数据库，请创建 RemoteOrders 和 OrderComments 表。RemoteOrders 表对应于统一数据库中的 RemoteOrders 表。MobiLink 服务器使用基于 SQL 的脚本与远程订单进行同步。OrderComments 表仅在客户端数据库上使用。MobiLink 服务器使用特殊事件处理 OrderComments 表。

也在客户端数据库上创建同步用户、同步发布和同步预订。

♦ 建立 MobiLink 客户端数据库

1. 创建 MobiLink 客户端数据库。

   在本课中，您将使用 dbinit 命令行实用程序创建 SQL Anywhere 数据库。

   a. 要创建 SQL Anywhere 数据库，请运行以下命令：

      ```
      dbinit -i -k remote1
      ```

      -i 和 -k 选项分别告诉 dbinit 忽略 jConnect 支持和 Watcom SQL 兼容性视图。

   b. 要启动数据库服务器，请运行以下命令：

      ```
      dbeng11 remote1
      ```

2. 使用 Interactive SQL 连接到 MobiLink 客户端数据库。

   运行以下命令：

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

3. 创建 RemoteOrders 表。

   在 Interactive SQL 中执行以下命令：

   ```
   create table RemoteOrders (
       order_id           integer not null,
       product_id         integer not null,
       quantity   integer,
       order_status       varchar(10) default 'new',
       primary key(order_id)
   )
   ```

4. 在 Interactive SQL 中运行以下命令创建 OrderComments 表：

   ```
   create table OrderComments (
       comment_id         integer not null,
       order_id           integer not null,
       order_comment      varchar (255),
       primary key(comment_id),
       foreign key (order_id) references
        RemoteOrders (order_id)
   )
   ```

5. 创建 MobiLink 同步用户、同步发布和同步预订：

   ```
   CREATE SYNCHRONIZATION USER ml_sales1;
   CREATE PUBLICATION order_publ (TABLE RemoteOrders, Table OrderComments);
   ```

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
 TYPE TCPIP ADDRESS 'host=localhost'
```

注意
使用 CREATE SYNCHRONIZATION SUBSCRIPTION 语句中的 TYPE 子句和 ADDRESS 子句指定如何连接到 MobiLink 服务器。

可以使用发布来确定同步哪些数据。在本例中指定整个 RemoteOrders 和 OrderComments 表。

进一步阅读

有关创建 SQL Anywhere 数据库的信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关 MobiLink 客户端的信息，请参见“MobiLink 客户端” 《MobiLink - 客户端管理》。

有关在客户端创建 MobiLink 对象的信息，请参见：

- “CREATE SYNCHRONIZATION USER 语句 [MobiLink]”一节 《SQL Anywhere 服务器 - SQL 参考》
- “CREATE PUBLICATION 语句 [MobiLink] [SQL Remote]”一节 《SQL Anywhere 服务器 - SQL 参考》
- “CREATE SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节 《SQL Anywhere 服务器 - SQL 参考》

第 6 课：同步

dbmlsync 实用程序为 SQL Anywhere 远程数据库启动 MobiLink 同步。在启动 dbmlsync 之前，将订单数据和注释添加到远程数据库。

♦ 建立远程数据（客户端）

1. 使用 Interactive SQL 连接到 MobiLink 客户端数据库：

   运行以下命令：

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

2. 将订单添加到客户端数据库的 RemoteOrders 表中。

   在 Interactive SQL 中执行以下命令。

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. 将注释添加到客户端数据库的 OrderComments 表中。

   在 Interactive SQL 中执行以下命令。

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. 提交所做的更改。

   在 Interactive SQL 中执行以下语句：

   ```
   COMMIT;
   ```

5. 在与统一数据库相同的目录中创建名为 *orderResponses.txt* 的空白文本文件。

♦ 启动同步客户端（客户端）

- 在命令提示符中，运行以下命令（在一行上）：

  ```
  dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
  ```

  以下是每个选项的说明：

选项	说明
-c	指定连接字符串。
-e scn	将 SendColumnNames 设置为 on。按名称引用列时，直接行处理需要这样设置。
-o	指定消息日志文件 *rem1.txt*。
-v+	-v 选项指定记录哪些信息。使用 -v+ 设置最大详细记录。

一旦启动 MobiLink 同步客户端，即会显示一个输出屏幕，指示同步已成功。基于 SQL 的同步会将客户端 RemoteOrders 表中的行传送到统一数据库中的 RemoteOrders 表。

Java 或 .NET 处理在 *orderComments.txt* 中插入注释。下一步，在 *orderResponses.txt* 中插入响应以下载到远程数据库。

♦ 使用直接行处理下载返回注释（服务器端和客户端）

1. 关闭任何 [SQL Anywhere MobiLink 客户端] 窗口
2. 插入返回注释。此操作在服务器端上进行。

   将以下文本添加到 *orderResponses.txt* 中。必须使用制表符分隔条目。在行的结尾处，按 Enter 键。

   ```
   1  1  Promotional material shipped
   ```
3. 使用 dbmlsync 客户端实用程序运行同步。

   此操作在客户端上进行。

   运行以下命令：

   ```
   dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -o rem1.txt -v+ -e scn=on
   ```

   出现 MobiLink 客户端实用程序。

   在 Interactive SQL 中，从 OrderComments 表进行选择，以验证下载了行。

注意
使用直接行处理下载的行不会由 mlsrv11 -v+ 选项输出，但是会由远程 -v+ 选项输出到远程日志中。

进一步阅读

有关 dbmlsync 的详细信息，请参见“SQL Anywhere 客户端” 《MobiLink - 客户端管理》。

清除

从计算机中删除教程资料。

♦ 删除教程资料

1. 关闭 Interactive SQL 的所有实例。
2. 关闭 SQL Anywhere、MobiLink 和同步客户端窗口。
3. 删除所有与教程相关的 DSN：
   a. 启动 ODBC 管理器。
      在命令提示符处键入以下命令：

      ```
      odbcad32
      ```

   b. 删除 mobilink_db 数据源。
4. 删除统一数据库和远程数据库：
   - 导航到统一数据库和远程数据库所在的目录。
   - 删除 *MLconsolidated.db*、*MLconsolidated.log*、*remote1.db* 和 *remote1.log*。

进一步阅读

有关 MobiLink 服务器 API 的详细信息，请参见：

- “使用 Java 语言编写同步脚本” 《MobiLink - 服务器管理》
- “使用 .NET 编写同步脚本” 《MobiLink - 服务器管理》

有关 MobiLink 直接行处理的详细信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

教程：与 Microsoft Excel 同步

目录

与 Excel 同步教程简介

本教程将指导您完成使用直接行处理以将 Microsoft Excel 电子表格中的数据与 MobiLink 客户端同步的基本步骤。本教程将 Java 实现用作一个示例。

本教程介绍如何实现 MobiLink 直接行处理，以便您可使用支持的统一数据库以外的其它数据源。

可以使用直接行处理实现远程数据与任何中央数据源、应用程序或 Web 服务之间的通信。

有关直接行处理的详细信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

本教程说明如何将 Microsoft Excel 电子表格中的数据同步到远程客户端。

必需的软件

- SQL Anywhere 11
- Java 软件开发工具包
- Microsoft Excel 2000 或更高版本

能力和经验

您需要：

- 熟悉 Java
- 熟悉 Microsoft Excel
- 了解 MobiLink 事件脚本和 MobiLink 同步的基本知识

目标

您将掌握并熟悉以下内容：

- 面向 Java 的 MobiLink 服务器 API
- 创建适用于 MobiLink 直接行处理的方法
- 使用 Java 访问 Microsoft Excel 工作表的数据

建议阅读的背景知识

- “了解 MobiLink 同步” 第 3 页
- “同步技术” 《MobiLink - 服务器管理》
- “直接行处理” 《MobiLink - 服务器管理》

MobiLink 代码交换示例位于 http://www.sybase.com/detail?id=1058600#319。

可在 MobiLink 新闻组上发布问题：sybase.public.sqlanywhere.mobilink。

第 1 课：建立 Excel 工作表

在本课中，可以创建 Excel 工作表并使用 Microsoft Excel 驱动程序定义 ODBC 数据源。Excel 工作表存储产品信息。

♦ 建立 Excel 数据源

1. 创建一个包含以下内容的 Excel 工作表：

order_id	comment_id	order_comment
1	2	附带的宣传材料
1	3	有关所需材料的详细信息

2. 将缺省工作表名称 **Sheet1** 更改为 **order_sheet**。
   a. 双击 [**Sheet1**] 选项卡。
   b. 键入 **order_sheet**。
3. 保存 Excel 工作簿。

   此教程将 *c:\MLobjexcel* 假定为服务器端组件的工作目录。在此工作目录中将工作簿另存为 *order_central.xls*。
4. 使用 Microsoft Excel 驱动程序创建 ODBC 数据源：
   a. 单击 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**ODBC 管理器**]。
   b. 单击 [**用户 DSN**] 选项卡。
   c. 单击 [**添加**]。
   d. 单击 [**Microsoft Excel Driver (*.xls)**]。
   e. 单击 [**完成**]。
   f. 在 [**数据源名**] 字段中，键入 **excel_datasource**。
   g. 单击 [**Select Workbook**] 并浏览至 *c:\MLobjexcel\order_central.xls*（包含工作表的文件）。
   h. 单击 [**确定**]。
   i. 单击 [**确定**]。

第 2 课：建立 MobiLink 统一数据库

MobiLink 统一数据库是数据的中央存储库，包括用来管理同步过程的 MobiLink 系统表和存储过程。使用直接行处理，可与统一数据库以外的其它数据源同步，但您仍需要统一数据库来维护 MobiLink 服务器所使用的信息。

在本课中，您将：

- 创建数据库并定义 ODBC 数据源。
- 添加数据表以同步远程客户端。
- 安装 MobiLink 系统表和存储过程。

注意
如果您已建立了具有 MobiLink 系统对象和 DSN 的 MobiLink 统一数据库，则可跳过本课。

创建统一数据库

在本教程中，您将使用 Sybase Central 的 [创建数据库向导] 创建 SQL Anywhere 数据库。

♦ 创建 SQL Anywhere RDBMS

1. 单击 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 单击 [**工具**] » [**SQL Anywhere 11**] » [**创建数据库**]。
3. 单击 [**下一步**]。
4. 单击 [**下一步**]。
5. 在 [**将主数据库文件保存到以下文件**] 字段中键入数据库的文件名和路径。例如，*c:\MLobjexcel \MLconsolidated.db*。
6. 请按照 [**创建数据库向导**] 中的其余说明进行操作并接受缺省值。在 [**连接到数据库**] 页面上，清除 [**最后一次断开连接后停止数据库**] 选项。
7. 单击 [**完成**]。

   MLconsolidated 数据库会出现在 Sybase Central 中。

为统一数据库定义 ODBC 数据源。

使用 SQL Anywhere 11 驱动程序为 MLconsolidated 数据库定义 ODBC 数据源。

♦ 为统一数据库定义 ODBC 数据源

1. 启动 Sybase Central。
2. 单击 [**工具**] » [**SQL Anywhere 11**] » [**打开 ODBC 管理器**]。
3. 单击 [**用户 DSN**] 选项卡，然后单击 [**添加**]。
4. 在 [**名称**] 列表中，单击 [**SQL Anywhere 11**]。单击 [**完成**]。

5. 在 [**SQL Anywhere 11 的 ODBC 配置**] 窗口中，进行以下操作：
   a. 单击 [**ODBC**] 选项卡。
   b. 在 [**数据源名**] 字段中，键入 **mlexcel_db**。
   c. 单击 [**登录**] 选项卡。
   d. 在 [**用户 ID**] 字段键入 **DBA**。
   e. 在 [**口令**] 字段中键入 **sql**。
   f. 单击 [**数据库**] 选项卡。
   g. 在 [**服务器名**] 字段中键入 **MLconsolidated**。
   h. 在 [**数据库文件**] 字段中，键入 *c:\MLobjexcel\MLconsolidated.db*。
   i. 单击 [**确定**]。
6. 单击 [**确定**]。

为同步创建表

在此过程中，在 MobiLink 统一数据库中创建 RemoteOrders 表。RemoteOrders 表包含以下各列：

列	说明
order_id	订单的唯一标识符。
product_id	产品的唯一标识符。
quantity	销售项目的数量。
order_status	订单状态。
last_modified	行上次修改的日期。此列用于基于时间戳的下载，这是一种为提高同步效率而过滤行的常用技术。

♦ 创建 RemoteOrders 表

1. 使用 Interactive SQL 连接到数据库。

   在命令提示符处，运行以下命令：

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. 在 Interactive SQL 中运行以下命令创建 RemoteOrders 表。

   ```
   CREATE TABLE RemoteOrders
   (
    order_id           integer not null,
    product_id         integer not null,
    quantity           integer,
    order_status       varchar(10) default 'new',
    last_modified      timestamp default current timestamp,
    primary key(order_id)
   )
   ```

Interactive SQL 会在统一数据库中创建 RemoteOrders 表。

在 Interactive SQL 中为以下过程保持连接。

运行 MobiLink 安装脚本

可以在 SQL Anywhere 11 安装目录的 *MobiLink/setup* 子目录中找到每个支持的统一数据库的安装脚本。

在此过程中建立 SQL Anywhere 统一数据库。可以使用 *syncsa.sql* 安装脚本来执行这一操作。运行 *syncsa.sql* 时会创建一系列以 **ml_** 为前缀的系统表和存储过程。MobiLink 服务器在同步过程中会使用这些表和存储过程。

♦ 安装 MobiLink 系统表

1. 如果您尚未建立连接，则在 Interactive SQL 中连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. 在 Interactive SQL 中运行以下命令来创建 MobiLink 系统表和存储过程。用 SQL Anywhere 11 安装目录的位置来替换 *c:\Program Files\SQL Anywhere 11*\。

   ```
   read "c:\Program Files\SQL Anywhere 11\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL 将 *syncsa.sql* 应用到您的统一数据库。

在 Interactive SQL 中为下一课保持连接。

进一步阅读

有关创建 SQL Anywhere 数据库的信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关创建表的信息，请参见“CREATE TABLE 语句”一节 《SQL Anywhere 服务器 - SQL 参考》。

有关建立 MobiLink 统一数据库的信息，请参见“MobiLink 统一数据库” 《MobiLink - 服务器管理》。

第 3 课：添加同步脚本

在本课中，您将为 SQL 行处理和直接行处理，将脚本添加到统一数据库中。

SQL 行处理

SQL 行处理允许您将远程数据与 MobiLink 统一数据库中的表同步。在本教程中，基于 SQL 的脚本定义：

- 如何将从 MobiLink 客户端上载的数据应用到统一数据库。
- 应从统一数据库下载哪些数据。

在本课中，您将为以下基于 SQL 的上载和下载事件编写同步脚本。

- **upload_insert** 对插入远程客户端数据库的新订单如何应用到统一数据库进行定义。
- **download_cursor** 对 MobiLink 统一数据库中更新的哪些订单应下载到远程客户端进行定义。

在此过程中，可以使用存储过程将同步脚本信息添加到 MobiLink 统一数据库。

♦ 将基于 SQL 的脚本添加到 MobiLink 系统表

1. 如果您尚未建立连接，则在 Interactive SQL 中连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. 使用 ml_add_table_script 存储过程为 upload_insert 和 download_cursor 事件添加基于 SQL 的表脚本。

   在 Interactive SQL 中运行以下命令。upload_insert 脚本将上载的 order_id、product_id、quantity 和 order_status 插入到 MobiLink 统一数据库。download_cursor 脚本使用基于时间戳的过滤将更新的行下载到远程客户端。

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'upload_insert',
        'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
         VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'download_cursor',
        'SELECT order_id, product_id, quantity, order_status
         FROM RemoteOrders WHERE last_modified >= ?');

   commit
   ```

直接行处理处理

在本教程中，您将使用直接行处理将特殊处理添加到基于 SQL 的同步系统中。在此过程中，您将注册与 handle_UploadData、handle_DownloadData 和 end_download 事件对应的方法名。在“第 4 课：使用 MobiLink 直接行处理创建 Java 类”一节第 211 页中创建您自己的 Java 类。

♦ 在 MobiLink 系统表中添加直接行处理的信息

1. 在 Interactive SQL 中，连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. 注册用于 handle_UploadData 和 handle_DownloadData 同步事件的 Java 方法。

   在 Interactive SQL 中执行以下命令。

   ```
   CALL ml_add_java_connection_script( 'default',
      'handle_UploadData',
        'MobiLinkOrders.GetUpload' );

   CALL ml_add_java_connection_script( 'default',
      'handle_DownloadData',
        'MobiLinkOrders.SetDownload' );

   commit
   ```

   Interactive SQL 分别注册用于 handle_UploadData 和 handle_DownloadData 事件的用户定义的 GetUpload 和 SetDownload 方法。

进一步阅读

有关使用基于 SQL 的事件将数据从远程客户端上载到 MobiLink 统一数据库的信息，请参见：

- “编写用于上载行的脚本”一节 《MobiLink - 服务器管理》
- “upload_insert 表事件”一节 《MobiLink - 服务器管理》
- “upload_update 表事件”一节 《MobiLink - 服务器管理》
- “upload_delete 表事件”一节 《MobiLink - 服务器管理》

有关将数据上载到除统一数据库之外的其它数据源的信息，请参见“处理直接上载”一节 《MobiLink - 服务器管理》。

有关使用基于 SQL 的事件从 MobiLink 统一数据库下载数据的信息，请参见：

- “编写用于下载行的脚本”一节 《MobiLink - 服务器管理》
- “download_cursor 表事件”一节 《MobiLink - 服务器管理》
- “download_delete_cursor 表事件”一节 《MobiLink - 服务器管理》

有关将数据下载到除统一数据库之外的其它数据源的信息，请参见“处理直接下载”一节 《MobiLink - 服务器管理》。

有关同步事件序列的信息，请参见“MobiLink 事件概述”一节 《MobiLink - 服务器管理》。

有关下载过滤的同步技术的信息，请参见“基于时间戳的下载”一节 《MobiLink - 服务器管理》和“在远程数据库之间对行进行分区”一节 《MobiLink - 服务器管理》。

有关管理脚本的信息，请参见“添加和删除脚本”一节 《MobiLink - 服务器管理》。

有关直接行处理的信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

第 4 课：使用 MobiLink 直接行处理创建 Java 类

在本课中，您将使用直接行处理来处理客户端数据库的 OrderComments 表中的行。为直接行处理添加以下方法：

- **GetUpload** 将此方法用于 handle_UploadData 事件。GetUpload 将上载的注释写入 Excel 工作表 *order_central.xls*。
- **SetDownload** 将此方法用于 handle_DownloadData 事件。SetDownload 检索存储在 Excel 工作表 *order_central.xls* 中的数据并将其发送到远程客户端。

以下过程介绍如何创建 Java 类，其中包括用于处理的方法。有关完整列表，请参见“完整的 MobiLinkOrders 代码列表 (Java)”一节第 214 页。

♦ 为仅下载直接行处理创建 Java 类

1. 创建名为 MobiLinkOrders 的类。

   在文本编辑器或开发环境中键入以下代码。

   ```
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   public class MobiLinkOrders {
       // ...
   }
   ```

2. 声明一个类级别 DBConnectionContext 实例。

   ```
   DBConnectionContext _cc;
   ```

   MobiLink 服务器会将 DBConnectionContext 实例传递到类构造函数。DBConnectionContext 封装有关与 MobiLink 统一数据库的当前连接的信息。

3. 创建类构造函数。

   类构造函数设置类级别 DBConnectionContext 实例。

   在文本编辑器或开发环境中键入以下代码。

   ```
   public MobiLinkOrders( DBConnectionContext cc ) {
       _cc = cc;
   }
   ```

4. 编写 GetUpload() 方法。

   GetUpload 方法获得表示 OrderComments 表的 UploadedTableData 类实例。OrderComments 表包含由远程销售雇员进行的特殊注释。您将在“第 6 课：建立 MobiLink 客户端”一节第 217 页中创建该表。UploadedTableData getInserts 方法返回新订单注释的结果集。

   在文本编辑器或开发环境中键入以下代码。

   ```
   //  method for the handle_UploadData synchronization event
   public void GetUpload( UploadData ut )
     throws SQLException, IOException {
     //  get an UploadedTableData for OrderComments
     UploadedTableData orderCommentsTbl =
   ```

```
ut.getUploadedTableByName("OrderComments");

  // get inserts uploaded by the MobiLink client
  ResultSet insertResultSet = orderCommentsTbl.getInserts();
  try {
    // connect to the excel worksheet through ODBC
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

    while( insertResultSet.next() ) {

      // get order comments
      int _commentID = insertResultSet.getInt("comment_id");
      int _orderID = insertResultSet.getInt("order_id");
      String _specialComments =
insertResultSet.getString("order_comment");

      // execute an insert statement to add the order comment to the
worksheet
      Statement st = con.createStatement();
      st.executeQuery( "insert into [order_sheet$]"
        + "(order_id, comment_id, order_comment) VALUES ("
        + Integer.toString(_orderID) + ", "
        + Integer.toString(_commentID) + ", '"
        + _specialComments + "')");

      st.close();
    }
    con.close();
  } catch(Exception ex) {
    System.err.print("Exception: ");
    System.err.println(ex.getMessage());
  }
  insertResultSet.close();
}
```

5. 编写 SetDownload 方法：

   a. 获得表示 OrderComments 表的类实例。

   使用 DBConnectionContext getDownloadData 方法可获得 DownloadData 实例。使用 DownloadData getDownloadTableByName 方法可为 OrderComments 表返回 DownloadTableData 实例。

   在文本编辑器或开发环境中键入以下代码。

```
DownloadData download_d = _cc.getDownloadData();
DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );
```

> **注意**
> 您将在“第 6 课：建立 MobiLink 客户端”一节第 217 页中于远程数据库上创建此表。

   b. 获得允许您将插入或更新操作添加到下载的准备好的语句或者 IDBCommand。

   使用 DownloadTableData getUpsertPreparedStatement 方法返回一个 java.sql.PreparedStatement 实例。

   在文本编辑器或开发环境中键入以下代码。


```
PreparedStatement download_upserts =
download_td.getUpsertPreparedStatement();
```

c. 设置每行的下载数据。

以下示例遍历 *order_central.xls* 工作表并将数据添加到 MobiLink 下载。

在文本编辑器或开发环境中键入以下代码。

```
try {
  // connect to the excel worksheet through ODBC
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

  // retrieve all the rows in the worksheet
  Statement st = con.createStatement();
  ResultSet Excel_rs = st.executeQuery( "select * from [order_sheet
$]" );

  while (Excel_rs.next()) {
    // retrieve the row data
    int Excel_comment_id = Excel_rs.getInt(1);
    int Excel_order_id = Excel_rs.getInt(2);
    String Excel_comment = Excel_rs.getString(3);

    // add the Excel data to the MobiLink download.
    download_upserts.setInt( 1, Excel_comment_id );
    download_upserts.setInt( 2, Excel_order_id );
    download_upserts.setString( 3, Excel_comment );
    download_upserts.executeUpdate();
  }
  // close the excel result set, statement, and connection.
  Excel_rs.close();
  st.close();
  con.close();
} catch(Exception ex) {
  System.err.print("Exception: ");
  System.err.println(ex.getMessage());
}
```

d. 关闭用来将插入操作或更新操作添加到下载的准备好的语句。

在文本编辑器或开发环境中键入以下代码。

```
download_upserts.close();
```

6. 将 Java 代码在工作目录 *c:\MLobjexcel* 中另存为 *MobiLinkOrders.java*。
7. 编译您的类文件。
   a. 浏览到包含 Java 源文件的目录。
   b. 编译 MobiLinkOrders 时引用面向 Java 的 MobiLink 服务器 API 库。

   需要引用位于 *install-dir\Java* 中的 *mlscript.jar*。运行以下命令来编译 Java 类，用 SQL Anywhere 11 目录替换 *c:\Program Files\SQL Anywhere 11*\：

```
javac -classpath "c:\Program Files\SQL Anywhere 11\java\mlscript.jar"
MobiLinkOrders.java
```

进一步阅读

有关同步逻辑的详细信息，请参见“使用 Java 语言编写同步脚本”《MobiLink - 服务器管理》。

有关直接行处理的详细信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

完整的 MobiLinkOrders 代码列表 (Java)

以下列表显示了本教程使用的完整的 Java MobiLinkOrders 类列表。

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class MobiLinkOrders {

  // class level DBConnectionContext
  DBConnectionContext _cc;

  public MobiLinkOrders( DBConnectionContext cc )
    throws IOException, FileNotFoundException {
    // declare a class-level DBConnectionContext
    _cc = cc;
  }


  //  method for the handle_UploadData synchronization event
  public void GetUpload( UploadData ut )
    throws SQLException, IOException {
    //  get an UploadedTableData for OrderComments
    UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

    // get inserts uploaded by the MobiLink client
    ResultSet insertResultSet = orderCommentsTbl.getInserts();
    try {
      // connect to the excel worksheet through ODBC
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

      while( insertResultSet.next() ) {
        // get order comments
        int _commentID = insertResultSet.getInt("comment_id");
        int _orderID = insertResultSet.getInt("order_id");
        String _specialComments = insertResultSet.getString("order_comment");

        // execute an insert statement to add the order comment to the
worksheet
        Statement st = con.createStatement();
        st.executeQuery( "insert into [order_sheet$]"
          + "(order_id, comment_id, order_comment) VALUES ("
          + Integer.toString(_orderID) + ", "
          + Integer.toString(_commentID) + ", '"
          + _specialComments + "')");
        st.close();
      }
      con.close();
    } catch(Exception ex) {
      System.err.print("Exception: ");
      System.err.println(ex.getMessage());
    }
    insertResultSet.close();
  }
```



```
  public void SetDownload() throws SQLException, IOException {

    DownloadData download_d = _cc.getDownloadData();

    DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );

    // Prepared statement to compile upserts (inserts or updates).
    PreparedStatement download_upserts =
download_td.getUpsertPreparedStatement();
    try {
      // connect to the excel worksheet through ODBC
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

      // retrieve all the rows in the worksheet
      Statement st = con.createStatement();
      ResultSet Excel_rs = st.executeQuery( "select * from [order_sheet$]" );

      while (Excel_rs.next()) {
        // retrieve the row data
        int Excel_comment_id = Excel_rs.getInt(1);
        int Excel_order_id = Excel_rs.getInt(2);
        String Excel_comment = Excel_rs.getString(3);

        // add the Excel data to the MobiLink download.
        download_upserts.setInt( 1, Excel_comment_id );
        download_upserts.setInt( 2, Excel_order_id );
        download_upserts.setString( 3, Excel_comment );
        download_upserts.executeUpdate();
      }
      // close the excel result set, statement, and connection.
      Excel_rs.close();
      st.close();
      con.close();
    } catch (Exception ex) {
      System.err.print("Exception: ");
      System.err.println(ex.getMessage());
    }

    download_upserts.close();
  }

}
```

第 5 课：启动 MobiLink 服务器

在本课中，您将启动 MobiLink 服务器。分别使用 -c 选项启动 MobiLink 服务器 (mlsrv11) 以连接到统一数据库，使用 -sl java 选项装载 Java 类。

♦ 为直接行处理启动 MobiLink 服务器

- 连接到统一数据库，然后在 mlsrv11 命令行装载 Java 类。

  运行以下命令。用 Java 源文件的位置替换 *c:\MLobjexcel*。

  ```
  mlsrv11 -c "dsn=mlexcel_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl
  java (-cp c:\MLobjexcel)
  ```

  下面对本教程中使用的每个 MobiLink 服务器选项均加以说明。选项 -o、-v 和 -dl 提供调试和故障排除信息。适宜在开发环境中使用这些记录选项。出于性能原因，-v 和 -dl 通常不在生产中使用。

选项	说明
-c	引出连接字符串。
-o	指定消息日志文件 *serverOut.txt*。
-v+	-v 选项指定记录哪些信息。使用 -v+ 设置最大详细记录。
-dl	在屏幕上显示所有日志消息。
-zu+	自动添加新用户。
-x	为 MobiLink 客户端设置通信协议和参数。
-sl java	指定一组目录以便用来搜索类文件，并强制在服务器启动时装载 Java 虚拟机。
-sl dnet	指定 .NET 程序集的位置，并强制在服务器启动时装载 CLR。

进一步阅读

有关 MobiLink 服务器选项的完整列表，请参见“MobiLink 服务器选项”《MobiLink - 服务器管理》。

有关装载 Java 和 .NET 类的详细信息，请分别参见“-sl java 选项”一节《MobiLink - 服务器管理》和“-sl dnet 选项”一节《MobiLink - 服务器管理》。

第 6 课：建立 MobiLink 客户端

在本教程中，您将针对统一数据库和 MobiLink 客户端使用 SQL Anywhere 数据库。用于教程目的时，MobiLink 客户端、统一数据库和 MobiLink 服务器均驻留在同一台计算机上。

若要建立 MobiLink 客户端数据库，请创建 RemoteOrders 和 OrderComments 表。RemoteOrders 表对应于统一数据库中的 RemoteOrders 表。MobiLink 服务器使用基于 SQL 的脚本与远程订单进行同步。OrderComments 表仅在客户端数据库上使用。MobiLink 服务器使用特殊事件处理 OrderComments 表。

也在客户端数据库上创建同步用户、同步发布和同步预订。

♦ 建立 MobiLink 客户端数据库

1. 创建 MobiLink 客户端数据库。

   在本课中，您将使用 dbinit 命令行实用程序创建 SQL Anywhere 数据库。

   - 要创建 SQL Anywhere 数据库，请运行以下命令：

     ```
     dbinit -I -k remote1
     ```

     -I 和 -k 选项分别指示 dbinit 忽略 jConnect 支持和 Watcom SQL 兼容性视图。

   - 要启动数据库服务器，请运行以下命令：

     ```
     dbeng11 remote1
     ```

2. 使用 Interactive SQL 连接到 MobiLink 客户端数据库。

   运行以下命令：

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

3. 创建 RemoteOrders 表。

   在 Interactive SQL 中执行以下命令：

   ```
   create table RemoteOrders (
       order_id           integer not null,
       product_id         integer not null,
       quantity   integer,
       order_status       varchar(10) default 'new',
       primary key(order_id)
   )
   ```

4. 在 Interactive SQL 中运行以下命令创建 OrderComments 表：

   ```
   create table OrderComments (
       comment_id         integer not null,
       order_id           integer not null,
       order_comment      varchar (255),
       primary key(comment_id),
       foreign key (order_id) references
        RemoteOrders (order_id)
   )
   ```

5. 创建 MobiLink 同步用户、同步发布和同步预订：

   ```
   CREATE SYNCHRONIZATION USER ml_sales1;
   CREATE PUBLICATION order_publ (TABLE RemoteOrders, Table OrderComments);
   ```

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
 TYPE TCPIP ADDRESS 'host=localhost'
```

注意
使用 CREATE SYNCHRONIZATION SUBSCRIPTION 语句中的 TYPE 子句和 ADDRESS 子句指定如何连接到 MobiLink 服务器。

可以使用发布来确定同步哪些数据。在本例中指定整个 RemoteOrders 和 OrderComments 表。

进一步阅读

有关创建 SQL Anywhere 数据库的信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关 MobiLink 客户端的信息，请参见“MobiLink 客户端” 《MobiLink - 客户端管理》。

有关在客户端创建 MobiLink 对象的信息，请参见：

- “CREATE SYNCHRONIZATION USER 语句 [MobiLink]”一节 《SQL Anywhere 服务器 - SQL 参考》
- “CREATE PUBLICATION 语句 [MobiLink] [SQL Remote]”一节 《SQL Anywhere 服务器 - SQL 参考》
- “CREATE SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节 《SQL Anywhere 服务器 - SQL 参考》

第 7 课：同步

dbmlsync 实用程序为 SQL Anywhere 远程数据库启动 MobiLink 同步。在启动 dbmlsync 之前，将订单数据和注释添加到远程数据库。

♦ 建立远程数据（客户端）

1. 使用 Interactive SQL 连接到 MobiLink 客户端数据库：

   运行以下命令：

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

2. 将订单添加到客户端数据库的 RemoteOrders 表中。

   在 Interactive SQL 中执行以下命令。

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. 将注释添加到客户端数据库的 OrderComments 表中。

   在 Interactive SQL 中执行以下命令。

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. 提交所做的更改。

   在 Interactive SQL 中执行以下语句：

   ```
   COMMIT;
   ```

♦ 启动同步客户端（客户端）

- 在命令提示符处，运行以下命令（在一行上）：

  ```
  dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
  ```

  以下是每个选项的说明：

选项	说明
-c	指定连接字符串。
-e scn	将 SendColumnNames 设置为 on。按名称引用列时，直接行处理需要这样设置。
-o	指定消息日志文件 *rem1.txt*。
-v+	-v 选项指定记录哪些信息。使用 -v+ 设置最大详细记录。

一旦启动 MobiLink 同步客户端，即会显示一个输出屏幕，指示同步已成功。基于 SQL 的同步将客户端 RemoteOrders 表中的行传送到统一数据库中的 RemoteOrders 表。

Java 处理在 *order_central.xls* 工作表中插入注释。存储在 *order_central.xls* 工作表中的信息会下载到客户端

在 Interactive SQL 中，从 OrderComments 表进行选择，以验证下载了行。

> **注意**
> 使用直接行处理下载的行不会由 mlsrv11 -v+ 选项输出，而会由远程 -v+ 选项输出到远程日志中。

进一步阅读

有关 dbmlsync 的详细信息，请参见“SQL Anywhere 客户端” 《MobiLink - 客户端管理》。

清除

从计算机中删除教程资料。

♦ 从计算机中删除教程资料

1. 关闭以下应用程序的所有实例。
   - Interactive SQL
   - Microsoft Excel
2. 删除 Excel 工作簿 *order_central.xls*。
3. 关闭 SQL Anywhere、MobiLink 和同步客户端窗口。
4. 删除所有与教程相关的 DSN。
   a. 启动 ODBC 管理器。
      在命令提示符处，键入以下命令：

      ```
      odbcad32
      ```

   b. 删除 excel_datasource 和 mlexcel_db 数据源。
5. 删除统一数据库和远程数据库。
   a. 导航到统一数据库和远程数据库所在的目录。
   b. 删除 *MLconsolidated.db*、*MLconsolidated.log*、*remote1.db* 和 *remote1.log*。

进一步阅读

有关 MobiLink 同步的详细信息，请参见“了解 MobiLink 同步”第 3 页。

有关 MobiLink 客户端的详细信息，请参见“MobiLink 客户端” 《MobiLink - 客户端管理》。

有关 MobiLink 直接行处理的详细信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

教程：与 XML 同步

目录

与 XML 同步教程简介

本教程介绍如何实现 MobiLink 直接行处理，以便您可使用支持的统一数据源以外的其它数据源。本教程将 Java 实现用作一个示例。

可以使用直接行处理实现远程数据与任何中央数据源、应用程序或 Web 服务之间的通信。

有关直接行处理的详细信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

本教程介绍如何将 XML 文件中的数据同步到远程客户端。

必需的软件

- SQL Anywhere 11
- Java 软件开发工具包
- XML DOM 库

能力和经验

您需要：

- 熟悉 Java
- 熟悉 XML
- 熟悉 XML DOM
- 了解 MobiLink 事件脚本和 MobiLink 同步的基本知识

目标

您将掌握并熟悉以下内容：

- 面向 Java 的 MobiLink 服务器 API
- 创建适用于 MobiLink 直接行处理的方法

建议阅读的背景知识

- “了解 MobiLink 同步” 第 3 页
- “同步技术” 《MobiLink - 服务器管理》
- “直接行处理” 《MobiLink - 服务器管理》

MobiLink 代码交换示例位于 http://www.sybase.com/detail?id=1058600#319。

可在 MobiLink 新闻组上发布问题：sybase.public.sqlanywhere.mobilink。

有关将 XML DOM 与 Java 结合使用的详细信息，请参见 Java API for XML Processing (JAXP)。

第 1 课：建立 XML 数据源

在本课中，您将创建一个 XML 文件。XML 文件会存储订单信息。

♦ 建立 XML 数据源

1. 创建一个包含以下内容的 XML 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<orders></orders>
```

2. 保存 XML 文件。

   本教程假定 *c:\MLobjxml* 作为服务器端组件的工作目录。在此工作目录中将 XML 文件另存为 *order_comments.xml*。

第 2 课：建立 MobiLink 统一数据库

MobiLink 统一数据库是数据的中央存储库，包括用来管理同步过程的 MobiLink 系统表和存储过程。使用直接行处理，可与统一数据库以外的其它数据源同步，但您仍需要统一数据库来维护 MobiLink 服务器所使用的信息。

在本课中，您将：

- 创建数据库并定义 ODBC 数据源。
- 添加数据表以同步远程客户端。
- 安装 MobiLink 系统表和存储过程。

注意
如果您已建立了具有 MobiLink 系统对象和 DSN 的 MobiLink 统一数据库，则可跳过本课。

♦ 创建 SQL Anywhere RDBMS

1. 选择 [**开始**] » [**程序**] » [**SQL Anywhere 11**] » [**Sybase Central**]。
2. 在 Sybase Central 中，选择 [**工具**] » [**SQL Anywhere 11**] » [**创建数据库**]。
3. 单击 [**下一步**]。
4. 保留缺省值 [**在这台计算机上创建数据库**]，然后单击 [**下一步**]。
5. 在 [**将主数据库文件保存到以下文件**] 字段中键入数据库的文件名和路径。例如，*c:\MLobjxml\MLconsolidated.db*。单击 [**下一步**]。
6. 请按照 [**创建数据库向导**] 中的其余说明进行操作并接受缺省值。在 [**连接到数据库**] 页面上，清除 [**最后一次断开连接后停止数据库**] 选项。
7. 单击 [**完成**]。

   在 Sybase Central 中就会出现名为 MLconsolidated 的数据库。

♦ 为统一数据库定义 ODBC 数据源

1. 在 Sybase Central [**工具**] 菜单中，选择 [**SQL Anywhere 11**] » [**打开 ODBC 管理器**]。
2. 单击 [**用户 DSN**] 选项卡，然后单击 [**添加**]。
3. 在 [**名称**] 列表中，单击 [**SQL Anywhere 11**]。单击 [**完成**]。
4. 在 [**SQL Anywhere 11 的 ODBC 配置**] 窗口中，进行以下操作：
   a. 单击 [**ODBC**] 选项卡。
   b. 在 [**数据源名**] 字段中，键入 **mlxml_db**。
   c. 单击 [**登录**] 选项卡。
   d. 在 [**用户 ID**] 字段键入 **DBA**。
   e. 在 [**口令**] 字段中键入 **sql**。

f. 单击 [**数据库**] 选项卡。

g. 在 [**服务器名**] 字段中键入 **MLconsolidated**。

h. 在 [**数据库文件**] 字段中，键入 *c:\MLobjxml\MLconsolidated.db*。

i. 单击 [**确定**]。

5. 单击 [**确定**]。

为同步创建表

在此过程中，在 MobiLink 统一数据库中创建 RemoteOrders 表。RemoteOrders 表包含以下各列：

列	说明
order_id	订单的唯一标识符。
product_id	产品的唯一标识符。
quantity	销售项目的数量。
order_status	订单状态。
last_modified	行上次修改的日期。此列用于基于时间戳的下载，这是一种为提高同步效率而过滤行的常用技术。

♦ 创建 RemoteOrders 表

1. 使用 Interactive SQL 连接到数据库。

   可以从 Sybase Central 或命令提示符启动 Interactive SQL。

   - 若要从 Sybase Central 启动 Interactive SQL，请单击数据库 **MLconsolidated - DBA**。从 Sybase Central [**文件**] 菜单中，选择 [**打开 Interactive SQL**]。
   - 若要在命令提示符下启动 Interactive SQL，请运行以下命令：

     ```
     dbisql -c "dsn=mlxml_db"
     ```

2. 在 Interactive SQL 中运行以下命令创建 RemoteOrders 表。

   ```
   CREATE TABLE RemoteOrders
   (
    order_id           integer not null,
    product_id         integer not null,
    quantity           integer,
    order_status       varchar(10) default 'new',
    last_modified      timestamp default current timestamp,
    primary key(order_id)
   )
   ```

   Interactive SQL 会在统一数据库中创建 RemoteOrders 表。

   在 Interactive SQL 中为以下过程保持连接。

运行 MobiLink 安装脚本

可以在 SQL Anywhere 11 安装目录的 *MobiLink/setup* 子目录中查找每个支持的统一数据库的安装脚本。

在此过程中建立 SQL Anywhere 统一数据库。可以使用 *syncsa.sql* 安装脚本来执行这一操作。运行 *syncsa.sql* 时会创建一系列以 **ml_** 为前缀的系统表和存储过程。MobiLink 服务器在同步过程中会使用这些表和存储过程。

♦ 安装 MobiLink 系统表

1. 如果您尚未建立连接，则在 Interactive SQL 中连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. 在 Interactive SQL 中运行以下命令来创建 MobiLink 系统表和存储过程。用 SQL Anywhere 11 安装目录的位置来替换 *c:\Program Files\SQL Anywhere 11*\。

   ```
   read "c:\Program Files\SQL Anywhere 11\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL 将 *syncsa.sql* 应用到您的统一数据库。

在 Interactive SQL 中为下一课保持连接。

进一步阅读

有关创建 SQL Anywhere 数据库的信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关创建表的信息，请参见“CREATE TABLE 语句”一节 《SQL Anywhere 服务器 - SQL 参考》。

有关建立 MobiLink 统一数据库的信息，请参见“MobiLink 统一数据库” 《MobiLink - 服务器管理》。

第 3 课：添加同步脚本

在本课中，您将为 SQL 行处理和直接行处理，将脚本添加到统一数据库中。

SQL 行处理

SQL 行处理可用于将远程数据与 MobiLink 统一数据库中的表同步。基于 SQL 的脚本可定义：

- 如何将从 MobiLink 客户端上载的数据应用到统一数据库。
- 应从统一数据库下载哪些数据。

在本课中，您将为以下基于 SQL 的上载和下载事件编写同步脚本。

- **upload_insert** 对插入远程客户端数据库的新订单如何应用到统一数据库进行定义。
- **download_cursor** 对 MobiLink 统一数据库中更新的哪些订单应下载到远程客户端进行定义。

在此过程中，可以使用存储过程将同步脚本信息添加到 MobiLink 统一数据库。

♦ 将基于 SQL 的脚本添加到 MobiLink 系统表

1. 如果您尚未建立连接，则在 Interactive SQL 中连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. 使用 ml_add_table_script 存储过程为 upload_insert 和 download_cursor 事件添加基于 SQL 的表脚本。

   在 Interactive SQL 中运行以下命令。upload_insert 脚本将上载的 order_id、product_id、quantity 和 order_status 插入到 MobiLink 统一数据库。download_cursor 脚本使用基于时间戳的过滤将更新的行下载到远程客户端。

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'upload_insert',
        'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
         VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
      'download_cursor',
        'SELECT order_id, product_id, quantity, order_status
         FROM RemoteOrders WHERE last_modified >= ?');

   commit
   ```

直接行处理处理

在本教程中，您将使用直接行处理将特殊处理添加到基于 SQL 的同步系统中。在此过程中，您将注册与 handle_UploadData、handle_DownloadData 和 end_download 事件对应的方法名。在“第 4 课：使用 MobiLink 直接行处理创建 Java”一节第 231 页中创建您自己的 java 类。

♦ 在 MobiLink 系统表中添加直接行处理的信息

1. 在 Interactive SQL 中，连接到统一数据库。

   运行以下命令：

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. 注册用于 handle_UploadData 和 handle_DownloadData 同步事件的 Java 方法。

   在 Interactive SQL 中执行以下命令。

   ```
   CALL ml_add_java_connection_script( 'default',
      'handle_UploadData',
        'MobiLinkOrders.GetUpload' );

   commit
   ```

   Interactive SQL 分别注册用于 handle_UploadData 和 handle_DownloadData 事件的用户定义的 GetUpload 和 SetDownload 方法。

进一步阅读

有关使用基于 SQL 的事件将数据从远程客户端上载到 MobiLink 统一数据库的信息，请参见：

- “编写用于上载行的脚本”一节 《MobiLink - 服务器管理》
- “upload_insert 表事件”一节 《MobiLink - 服务器管理》
- “upload_update 表事件”一节 《MobiLink - 服务器管理》
- “upload_delete 表事件”一节 《MobiLink - 服务器管理》

有关将数据上载到除统一数据库之外的其它数据源的信息，请参见“处理直接上载”一节 《MobiLink - 服务器管理》。

有关使用基于 SQL 的事件从 MobiLink 统一数据库下载数据的信息，请参见：

- “编写用于下载行的脚本”一节 《MobiLink - 服务器管理》
- “download_cursor 表事件”一节 《MobiLink - 服务器管理》
- “download_delete_cursor 表事件”一节 《MobiLink - 服务器管理》

有关将数据下载到除统一数据库之外的其它数据源的信息，请参见“处理直接下载”一节 《MobiLink - 服务器管理》。

有关同步事件序列的信息，请参见“MobiLink 事件概述”一节 《MobiLink - 服务器管理》。

有关下载过滤的同步技术的信息，请参见“基于时间戳的下载”一节 《MobiLink - 服务器管理》和“在远程数据库之间对行进行分区”一节 《MobiLink - 服务器管理》。

有关管理脚本的信息，请参见“添加和删除脚本”一节 《MobiLink - 服务器管理》。

有关直接行处理的信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

第 4 课：使用 MobiLink 直接行处理创建 Java

在本课中，您将使用直接行处理来处理客户端数据库的 OrderComments 表中的行。为直接行处理添加以下方法：

- **GetUpload** 将此方法用于 handle_UploadData 事件。GetUpload 将上载的注释写入 XML 文件中。

以下过程介绍如何创建 Java 类，其中包括用于处理的方法。有关完整列表，请参见“完整的 MobiLinkOrders 代码列表 (Java)”一节第 235 页。

♦ 为仅下载直接行处理创建 Java 类

1. 使用 Java 创建名为 MobilinkOrders 的类。

   在文本编辑器或开发环境中键入以下代码：

```
//Mobilink imports
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

//XML parser
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

//DOM Objects
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

//For writing XML objects
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

public class MobiLinkOrders {
    // ...
}
```

2. 声明一个类级别 DBConnectionContext 实例和文档实例。文档是将 XML 文档作为对象表示的类。

```
DBConnectionContext _cc;
 Document _doc;
```

   MobiLink 服务器会将 DBConnectionContext 实例传递到类构造函数。DBConnectionContext 封装有关与 MobiLink 统一数据库的当前连接的信息。

3. 创建类构造函数。

   类构造函数设置类级别 DBConnectionContext 实例。

在文本编辑器或开发环境中键入以下代码：

```
public MobiLinkOrders( DBConnectionContext cc ) {
  _cc = cc;
}
```

4. 编写 GetUpload() 方法。
   a. 编写方法声明。

      在文本编辑器或开发环境中键入以下代码：

      ```
      // method for the handle_UploadData synchronization event
      public void GetUpload( UploadData ut ) throws SQLException, IOException
      {
      ```

   b. 从 Mobilink 客户端获得任何上载的插入。

      在文本编辑器或开发环境中键入以下代码：

      ```
      //  get an UploadedTableData for OrderComments
        UploadedTableData orderCommentsTbl =
          ut.getUploadedTableByName("OrderComments");

        // get inserts uploaded by the MobiLink client
        ResultSet insertResultSet = orderCommentsTbl.getInserts();
      ```

   c. 在现有 XML 文件 **order_comments.xml** 中读取。

      在文本编辑器或开发环境中键入以下代码：

      ```
            readDom("order_comments.xml");
      ```

   d. 对于每个插入行，将其添加到 XML 文件。

      在文本编辑器或开发环境中键入以下代码：

      ```
            // Write out each insert in the XML file
            while( insertResultSet.next() ) {
              buildXML(insertResultSet);
            }
      ```

   e. 编写 XML 文件。

      在文本编辑器或开发环境中键入以下代码：

      ```
        writeXML();
      ```

   f. 关闭 ResultSet。

      在文本编辑器或开发环境中键入以下代码：

      ```
            // Close the result set of uploaded inserts
            insertResultSet.close();
      ```

5. 编写 buildXML 方法。

   在文本编辑器或开发环境中键入以下代码：

   ```
   private void buildXML( ResultSet rs ) throws SQLException {
     int order_id = rs.getInt(1);
     int comment_id = rs.getInt(2);
     String order_comment = rs.getString(3);

     //Create the comment object to be added to the XML file
   ```



```
      Element comment = _doc.createElement("comment");
      comment.setAttribute("id", Integer.toString(comment_id));
      comment.appendChild(_doc.createTextNode(order_comment));

      //Get the root element (orders)
      Element root = _doc.getDocumentElement();

      //get each individual order
      NodeList rootChildren = root.getChildNodes();
      for(int i = 0; i < rootChildren.getLength(); i++) {
        //if the order exists, add the comment to the order
        Node n = rootChildren.item(i);
        if(n.getNodeType() == Node.ELEMENT_NODE) {
          Element e = (Element) n;
          int idIntVal = Integer.parseInt(e.getAttribute("id"));
          if(idIntVal == order_id) {
            e.appendChild(comment);
            //The comment has been added to the file, so exit the function
            return;
          }
        }
      }

      //if the order did not exist already, create it
      Element order = _doc.createElement("order");
      order.setAttribute("id", Integer.toString(order_id));
      //add the comment to the new order
      order.appendChild(comment);
      root.appendChild(order);
    }
```

6. 编写 writeXML 方法。

   在文本编辑器或开发环境中键入以下代码：

```
    private void writeXML() {

      try {

        // Use a Transformer for output
        TransformerFactory tFactory = TransformerFactory.newInstance();
        Transformer transformer = tFactory.newTransformer();

        //The XML source is _doc
        DOMSource source = new DOMSource(_doc);
        //write the xml data to order_comments.xml
        StreamResult result = new StreamResult(new
  File("order_comments.xml"));
        transformer.transform(source, result);

      } catch (TransformerConfigurationException tce) {
        // Error generated by the parser
        System.out.println ("\n** Transformer Factory error");
        System.out.println("   " + tce.getMessage() );

        // Use the contained exception, if any
        Throwable x = tce;
        if (tce.getException() != null) x = tce.getException();
        x.printStackTrace();

      } catch (TransformerException te) {
        // Error generated by the parser
        System.out.println ("\n** Transformation error");
        System.out.println("   " + te.getMessage() );
```

```
            // Use the contained exception, if any
            Throwable x = te;
            if (te.getException() != null) x = te.getException();
            x.printStackTrace();

        }

         }
```

7. 编写 readDOM 方法。

   在文本编辑器或开发环境中键入以下代码：

```
    private void readDom(String filename) {
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        try {
          //parse the Document data into _doc
          DocumentBuilder builder = factory.newDocumentBuilder();
          _doc = builder.parse( new File(filename) );

        } catch (SAXException sxe) {
          // Error generated during parsing)
          Exception x = sxe;
          if (sxe.getException() != null) x = sxe.getException();
          x.printStackTrace();

        } catch (ParserConfigurationException pce) {
          // Parser with specified options can't be built
          pce.printStackTrace();

        } catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
        }

    }
```

8. 将 Java 代码在工作目录 *c:\MLobjxml* 中保存为 *MobiLinkOrders.java*。
9. 编译您的类文件。
   a. 浏览到包含 Java 源文件的目录。
   b. 编译 MobiLinkOrders 时引用面向 Java 的 MobiLink 服务器 API 库。

      对于 Java，需要引用位于 *install-dir\Java* 中的 *mlscript.jar*。您还需要确保正确安装了 XML DOM 库。运行以下命令来编译 Java 类，用您的 SQL Anywhere 11 目录替换 *c:\Program Files\SQL Anywhere 11*：

```
javac -classpath "c:\Program Files\SQL Anywhere 11\java\mlscript.jar"
MobiLinkOrders.java
```

进一步阅读

有关同步逻辑的详细信息，请参见“使用 Java 语言编写同步脚本”《MobiLink - 服务器管理》。

有关直接行处理的详细信息，请参见“直接行处理”《MobiLink - 服务器管理》。

完整的 MobiLinkOrders 代码列表 (Java)

以下列表显示了本教程使用的完整的 Java MobiLinkOrders 类列表。

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

// For write operation
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

public class MobiLinkOrders {

  // class level DBConnectionContext
  DBConnectionContext _cc;
  Document _doc;

  public MobiLinkOrders( DBConnectionContext cc ) throws IOException,
FileNotFoundException {
    // declare a class-level DBConnectionContext
    _cc = cc;
  }


  //  method for the handle_UploadData synchronization event
  public void GetUpload( UploadData ut ) throws SQLException, IOException {
    // Get an UploadedTableData for the remote table
    UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("OrderComments");

    // Get inserts uploaded by the MobiLink client
    // as a java.sql.ResultSet
    ResultSet insertResultSet = remoteOrdersTable.getInserts();

    readDom("order_comments.xml");

    // Write out each insert in the XML file
    while( insertResultSet.next() ) {
      buildXML(insertResultSet);
    }
    writeXML();

    // Close the result set of uploaded inserts
    insertResultSet.close();

  }
```

```
private void buildXML( ResultSet rs ) throws SQLException {
  int order_id = rs.getInt(1);
  int comment_id = rs.getInt(2);
  String order_comment = rs.getString(3);

  //Create the comment object to be added to the XML file
  Element comment = _doc.createElement("comment");
  comment.setAttribute("id", Integer.toString(comment_id));
  comment.appendChild(_doc.createTextNode(order_comment));

  //Get the root element (orders)
  Element root = _doc.getDocumentElement();

  //get each individual order
  NodeList rootChildren = root.getChildNodes();
  for(int i = 0; i < rootChildren.getLength(); i++) {
    //if the order exists, add the comment to the order
    Node n = rootChildren.item(i);
    if(n.getNodeType() == Node.ELEMENT_NODE) {
      Element e = (Element) n;
      int idIntVal = Integer.parseInt(e.getAttribute("id"));
      if(idIntVal == order_id) {
        e.appendChild(comment);
        //The comment has been added to the file, so exit the function
        return;
      }
    }
  }

  //if the order did not exist already, create it
  Element order = _doc.createElement("order");
  order.setAttribute("id", Integer.toString(order_id));
  //add the comment to the new order
  order.appendChild(comment);
  root.appendChild(order);
}

private void writeXML() {

  try {

    // Use a Transformer for output
    TransformerFactory tFactory = TransformerFactory.newInstance();
    Transformer transformer = tFactory.newTransformer();

    //The XML source is _doc
    DOMSource source = new DOMSource(_doc);
    //write the xml data to order_comments.xml
    StreamResult result = new StreamResult(new File("order_comments.xml"));
    transformer.transform(source, result);

  } catch (TransformerConfigurationException tce) {
    // Error generated by the parser
    System.out.println ("\n** Transformer Factory error");
    System.out.println("   " + tce.getMessage() );

    // Use the contained exception, if any
    Throwable x = tce;
    if (tce.getException() != null) x = tce.getException();
    x.printStackTrace();

  } catch (TransformerException te) {
    // Error generated by the parser
    System.out.println ("\n** Transformation error");
```



```
        System.out.println("   " + te.getMessage() );

        // Use the contained exception, if any
        Throwable x = te;
        if (te.getException() != null) x = te.getException();
        x.printStackTrace();

    }

     }

     private void readDom(String filename) {
       DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        try {
          //parse the Document data into _doc
          DocumentBuilder builder = factory.newDocumentBuilder();
          _doc = builder.parse( new File(filename) );

        } catch (SAXException sxe) {
          // Error generated during parsing)
          Exception x = sxe;
          if (sxe.getException() != null) x = sxe.getException();
          x.printStackTrace();

        } catch (ParserConfigurationException pce) {
          // Parser with specified options can't be built
          pce.printStackTrace();

        } catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
        }

     }

  }
```

第 5 课：启动 MobiLink 服务器

在本课中，您将启动 MobiLink 服务器。使用 -c 选项启动 MobiLink 服务器 (mlsrv11) 以连接到统一数据库，使用 -sl java 选项装载 Java 类。

♦ 为直接行处理启动 MobiLink 服务器

- 连接到统一数据库，然后在 mlsrv11 命令行装载 Java 类。

  运行以下命令。用 Java 源文件的位置替换 *c:\MLobjxml*。

  ```
  mlsrv11 -c "dsn=mlxml_db" -o c:\MLobjxml\serverOut.txt -v+ -dl -zu+ -x
  tcpip  -sl java (-cp c:\MLobjxml)
  ```

  即会出现 MobiLink 服务器窗口。

  下面对本教程中使用的每个 MobiLink 服务器选项均加以说明。选项 -o、-v 和 -dl 提供调试和故障排除信息。适宜在开发环境中使用这些记录选项。出于性能原因，-v 和 -dl 通常不在生产中使用。

选项	说明
-c	引出连接字符串。
-o	指定消息日志文件 *serverOut.txt*。
-v+	-v 选项指定记录哪些信息。使用 -v+ 设置最大详细记录。
-dl	在屏幕上显示所有日志消息。
-zu+	自动添加新用户。
-x	为 MobiLink 客户端设置通信协议和参数。
-sl java	指定一组目录以便用来搜索类文件，并强制在服务器启动时装载 Java 虚拟机。
-sl dnet	指定 .NET 程序集的位置，并强制在服务器启动时装载 CLR。

进一步阅读

有关 MobiLink 服务器选项的完整列表，请参见“MobiLink 服务器选项” 《MobiLink - 服务器管理》。

有关装载 Java 类的详细信息，请参见“-sl java 选项”一节 《MobiLink - 服务器管理》。

第 6 课：建立 MobiLink 客户端

在本教程中，您将针对统一数据库和 MobiLink 客户端使用 SQL Anywhere 数据库。用于教程目的时，MobiLink 客户端、统一数据库和 MobiLink 服务器均驻留在同一台计算机上。

若要建立 MobiLink 客户端数据库，请创建 RemoteOrders 和 OrderComments 表。RemoteOrders 表对应于统一数据库中的 RemoteOrders 表。MobiLink 服务器使用基于 SQL 的脚本与远程订单进行同步。OrderComments 表仅在客户端数据库上使用。MobiLink 服务器使用特殊事件处理 OrderComments 表。

也在客户端数据库上创建同步用户、同步发布和同步预订。

♦ 建立 MobiLink 客户端数据库

1. 创建 MobiLink 客户端数据库。

   在本课中，您将使用 dbinit 命令行实用程序创建 SQL Anywhere 数据库。

   a. 要创建 SQL Anywhere 数据库，请运行以下命令：

      ```
      dbinit -i -k remote1
      ```

      -i 和 -k 选项分别告诉 dbinit 忽略 jConnect 支持和 Watcom SQL 兼容性视图。

   b. 要启动数据库服务器，请运行以下命令：

      ```
      dbeng11 remote1
      ```

2. 使用 Interactive SQL 连接到 MobiLink 客户端数据库。

   运行以下命令：

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

3. 创建 RemoteOrders 表。

   在 Interactive SQL 中执行以下命令：

   ```
   create table RemoteOrders (
       order_id           integer not null,
       product_id         integer not null,
       quantity   integer,
       order_status       varchar(10) default 'new',
       primary key(order_id)
   )
   ```

4. 在 Interactive SQL 中运行以下命令创建 OrderComments 表：

   ```
   create table OrderComments (
       order_id           integer not null,
       comment_id         integer not null,
       order_comment      varchar (255),
       primary key(comment_id),
       foreign key (order_id) references
        RemoteOrders (order_id)
   )
   ```

5. 创建 MobiLink 同步用户、同步发布和同步预订：

   ```
   CREATE SYNCHRONIZATION USER ml_sales1;
   CREATE PUBLICATION order_publ (TABLE RemoteOrders, Table OrderComments);
   ```

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
 TYPE TCPIP ADDRESS 'host=localhost'
```

注意
使用 CREATE SYNCHRONIZATION SUBSCRIPTION 语句中的 TYPE 子句和 ADDRESS 子句指定如何连接到 MobiLink 服务器。

可以使用发布来确定同步哪些数据。在本例中指定整个 RemoteOrders 和 OrderComments 表。

进一步阅读

有关创建 SQL Anywhere 数据库的信息，请参见“初始化实用程序 (dbinit)”一节 《SQL Anywhere 服务器 - 数据库管理》。

有关 MobiLink 客户端的信息，请参见“MobiLink 客户端” 《MobiLink - 客户端管理》。

有关在客户端创建 MobiLink 对象的信息，请参见：

- “CREATE SYNCHRONIZATION USER 语句 [MobiLink]”一节 《SQL Anywhere 服务器 - SQL 参考》
- “CREATE PUBLICATION 语句 [MobiLink] [SQL Remote]”一节 《SQL Anywhere 服务器 - SQL 参考》
- “CREATE SYNCHRONIZATION SUBSCRIPTION 语句 [MobiLink]”一节 《SQL Anywhere 服务器 - SQL 参考》

第 7 课：同步

dbmlsync 实用程序为 SQL Anywhere 远程数据库启动 MobiLink 同步。在启动 dbmlsync 之前，将订单数据和注释添加到远程数据库。

♦ 建立远程数据（客户端）

1. 使用 Interactive SQL 连接到 MobiLink 客户端数据库：

   运行以下命令：

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

2. 将订单添加到客户端数据库的 RemoteOrders 表中。

   在 Interactive SQL 中执行以下命令。

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. 将注释添加到客户端数据库的 OrderComments 表中。

   在 Interactive SQL 中执行以下命令。

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. 提交所做的更改。

   在 Interactive SQL 中执行以下语句：

   ```
   COMMIT;
   ```

♦ 启动同步客户端（客户端）

- 在命令提示符处，运行以下命令。用 Java 源文件的位置替换 *c:\MLobjxml*。

  ```
  dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -e scn=on -o c:\MLobjxml
  \rem1.txt -v+
  ```

  以下是每个选项的说明：

选项	说明
-c	指定连接字符串。
-e scn	将 SendColumnNames 设置为 on。按名称引用列时，直接行处理需要这样设置。
-o	指定消息日志文件 *rem1.txt*。
-v+	-v 选项指定记录哪些信息。使用 -v+ 设置最大详细记录。

一旦启动 MobiLink 同步客户端，即会显示一个输出屏幕，指示同步已成功。基于 SQL 的同步会将客户端 RemoteOrders 表中的行传送到统一数据库中的 RemoteOrders 表。

Java 处理在 XML 文件中插入注释。

进一步阅读

有关 dbmlsync 的详细信息，请参见“SQL Anywhere 客户端” 《MobiLink - 客户端管理》。

清除

从计算机中删除教程资料。

♦ 从计算机中删除教程资料

1. 关闭以下应用程序的所有实例。
   - Interactive SQL
2. 关闭 SQL Anywhere、MobiLink 和同步客户端窗口。
3. 删除所有与教程相关的 DSN。
   a. 启动 ODBC 管理器。
      在命令提示符处，键入以下命令：

      ```
      odbcad32
      ```

   b. 删除 mlxml_db 数据源。
4. 删除统一数据库和远程数据库。
   a. 浏览到统一数据库和远程数据库所在的目录。
   b. 删除 *MLconsolidated.db*、*MLconsolidated.log*、*remote1.db* 和 *remote1.log*。

进一步阅读

有关 MobiLink 同步的详细信息，请参见“了解 MobiLink 同步”第 3 页。

有关 MobiLink 客户端的详细信息，请参见“MobiLink 客户端” 《MobiLink - 客户端管理》。

有关 MobiLink 直接行处理的详细信息，请参见“直接行处理” 《MobiLink - 服务器管理》。

术语表

术语表

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio 的关系数据库服务器组件，专供在移动和嵌入式环境中使用，或作为中小型企业的服务器使用。在版本 10.0.0 中，Adaptive Server Anywhere 更名为 SQL Anywhere 服务器，SQL Anywhere Studio 更名为 SQL Anywhere。

另请参见： “SQL Anywhere”一节第 264 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 274 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 274 页
- “代码页”一节第 249 页
- “归类”一节第 253 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 261 页
- “隔离级别”一节第 252 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 274 页
- “外键”一节第 266 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 265 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 264 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 251 页

触发器

一种特殊形式的存储过程，用户运行修改数据的查询时会自动执行该存储过程。

另请参见：

- “行级触发器”一节第 253 页
- “语句级触发器”一节第 271 页
- “完整性”一节第 267 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 272 页

代理 ID

另请参见： “客户端消息存储库 ID”一节第 257 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 274 页
- “编码”一节第 247 页
- “归类”一节第 253 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 263 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 264 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 265 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 251 页
- “项目”一节第 269 页
- “发布更新”一节第 250 页

发布更新

SQL Remote 复制中对一个数据库中的一个或多个发布所做更改的列表。发布更新将作为复制消息的一部分定期发送到远程数据库。

另请参见：

- “复制”一节第 251 页
- “发布”一节第 250 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 251 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 251 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 258 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 258 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 251 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 251 页
- “发布更新”一节第 250 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 257 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 263 页

规范化

对数据库模式的改进，目的在于按照基于关系数据库理论的规则消除冗余并改善组织。

归类

定义数据库中文本属性的字符集与排序顺序的组合。对于 SQL Anywhere 数据库，缺省归类取决于运行服务器时所使用的操作系统和语言；例如，英语 Windows 系统上的缺省归类为 1252LATIN1。归类（也称作归类序列）用于对字符串进行比较和排序。

另请参见：

- “字符集”一节第 274 页
- “代码页”一节第 249 页
- “编码”一节第 247 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 249 页
- “语句级触发器”一节第 271 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 261 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 254 页
- “jConnect”一节第 254 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 254 页
- “iAnywhere JDBC 驱动程序”一节第 253 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 254 页
- “iAnywhere JDBC 驱动程序”一节第 253 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 257 页
- “视图”一节第 261 页

基于会话的同步

一种同步类型，这种同步会使数据表示在统一数据库和远程数据库都一致。MobiLink 基于会话。

基于脚本的上载

MobiLink 中一种将上载过程自定义为使用日志文件的替代方法的方式。

基于 SQL 的同步

MobiLink 中一种使用 MobiLink 事件将表数据与支持 MobiLink 的统一数据库进行同步的方式。对于基于 SQL 的同步，可以直接使用 SQL，也可以使用面向 Java 和 .NET 平台的 MobiLink 服务器 API 返回 SQL。

基于文件的下载

在 MobiLink 中同步数据的一种方式，其中下载以文件的方式进行分发，从而支持脱机分发同步更改。

集成登录

一种登录功能，它允许将同一个用户 ID 和口令用于操作系统登录、网络登录和数据库连接。

监听器

一个程序 (dblsn)，用于 MobiLink 服务器启动的同步。监听器安装在远程设备上，它们被配置为在接收到来自通告程序的信息时启动针对设备的操作。

另请参见： “服务器启动的同步”一节第 251 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 273 页
- “外键约束”一节第 267 页
- “主键约束”一节第 274 页
- “唯一约束”一节第 268 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 261 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 262 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 266 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 257 页
- “全局临时表”一节第 260 页

客户端/服务器

一种软件体系结构，在这种体系结构中，一个应用程序（客户端）从另一个应用程序（服务器）获取信息并向该应用程序发送信息。这两个应用程序常位于通过网络连接的不同计算机上。

客户端消息存储库

QAnywhere 中一种用于在远程设备上存储消息的 SQL Anywhere 数据库。

客户端消息存储库 ID

QAnywhere 中一种对客户端消息存储库进行唯一标识的 MobiLink 远程 ID。

快照隔离

一种为发出读请求的事务返回数据的已提交版本的隔离级别。SQL Anywhere 提供了以下三种快照隔离级别：快照、语句快照和只读语句快照。使用快照隔离时，读操作不会阻塞写操作。

另请参见： “隔离级别”一节第 252 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 257 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 251 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 257 页
- “生成的连接条件”一节第 262 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 256 页
- “全局临时表”一节第 260 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 252 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 251 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 265 页
- “同步”一节第 265 页
- “UltraLite”一节第 266 页

MobiLink 服务器

运行 MobiLink 同步的计算机程序，即 mlsrv11。

MobiLink 监控器

一种用于监控 MobiLink 同步的图形化工具。

MobiLink 客户端

有两种 MobiLink 客户端。对于 SQL Anywhere 远程数据库，MobiLink 客户端是 dbmlsync 命令行实用程序。对于 UltraLite 远程数据库，MobiLink 客户端内置于 UltraLite 运行时库中。

MobiLink 系统表

MobiLink 同步所需的系统表。它们由 MobiLink 安装程序脚本安装到 MobiLink 统一数据库中。

MobiLink 用户

MobiLink 用户用于与 MobiLink 服务器进行连接。在远程数据库上创建 MobiLink 用户，然后在统一数据库中注册该用户。MobiLink 用户名完全独立于数据库用户名。

模式

数据库的结构，其中包括表、列和索引以及它们之间的关系。

内连接

一种连接，在这种连接中，仅当两个表都满足连接条件时行才会出现在结果集中。内连接是缺省设置。

另请参见：

- “连接”一节第 257 页
- “外连接”一节第 267 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 251 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 257 页
- “局部临时表”一节第 256 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 261 页
- “事务日志镜像”一节第 262 页
- “完全备份”一节第 267 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 251 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 254 页
- “视图”一节第 261 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 255 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 261 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 257 页
- “连接条件”一节第 257 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 271 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 266 页
- “主键”一节第 274 页
- “数据库管理系统 (DBMS)”一节第 263 页
- “关系数据库管理系统 (RDBMS)”一节第 252 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 252 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 264 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 263 页
- “SYS”一节第 265 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 250 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 259 页
- “SQL”一节第 264 页
- “数据库管理系统 (DBMS)”一节第 263 页

锁定

一种在同时执行多个事务的过程中保护数据完整性的并发控制机制。SQL Anywhere 会自动应用锁以防止两个连接同时更改同一数据，并防止其它连接读取正接受更改的数据。

您可以通过设置隔离级别来控制锁定。

另请参见：

- “隔离级别”一节第 252 页
- “并发”一节第 247 页
- “完整性”一节第 267 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 265 页
- “复制”一节第 251 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 251 页
- “监听器”一节第 255 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 258 页
- “SQL Remote”一节第 264 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 251 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 260 页
- “服务器启动的同步”一节第 251 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 266 页

外部登录

与远程服务器通信时使用的替代登录名和口令。缺省情况下，SQL Anywhere 每次代表其客户端连接到远程服务器时都会使用这些客户端的名称和口令。但是，您可以通过创建外部登录来替换这一缺省设置。外部登录是指与远程服务器通信时使用的替代登录名和口令。

外键

一个表中复制另一个表中主键值的一个或多个列。外键建立表间的关系。

另请参见：

- “主键”一节第 274 页
- “外表”一节第 266 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 273 页
- “检查约束”一节第 255 页
- “主键约束”一节第 274 页
- “唯一约束”一节第 268 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 257 页
- “内连接”一节第 259 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 273 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 248 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 251 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 252 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 267 页
- “主键约束”一节第 274 页
- “约束”一节第 273 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 268 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 255 页

物理索引

索引存储在磁盘上的实际索引结构。

系统表

一种表，由 SYS 或 dbo 拥有，用于保存元数据。系统表也称作数据字典表，由数据库服务器创建并维护。

系统对象

由 SYS 或 dbo 拥有的数据库对象。

系统视图

存在于每一个数据库中的一种视图，它以易于理解的格式表示系统表中包含的信息。

下载

同步过程的一个阶段，在此阶段数据从统一数据库传送到远程数据库。

相关名

查询的 FROM 子句中使用的表或视图的名称—要么是表或视图的原始名称，要么是在 FROM 子句中定义的替代名称。

项目

在 MobiLink 或 SQL Remote 中，项目是表示整个表或表中行和列子集的数据库对象。项目在发布中组合在一起。

另请参见：

- “复制”一节第 251 页
- “发布”一节第 250 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 256 页
- “服务器消息存储库”一节第 251 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 251 页
- “统一数据库”一节第 265 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 251 页
- “FILE”一节第 251 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 273 页
- “用户定义数据类型”一节第 271 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 266 页

用户定义数据类型

请参见“域”一节第 271 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 271 页
- “游标位置”一节第 271 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 271 页
- “游标位置”一节第 271 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 271 页
- “游标结果集”一节第 271 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 249 页
- “行级触发器”一节第 253 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 262 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 250 页
- “MobiLink 用户”一节第 259 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 259 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 250 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 265 页
- “统一数据库”一节第 265 页

约束

对特定数据库对象（如表或列）中所包含值的限制。例如，列可以具有唯一性约束，该约束要求该列中的所有值互不相同。表可以具有外键约束，该约束指定该表中的信息与某个其它表中数据的关系。

另请参见：

- “检查约束”一节第 255 页
- “外键约束”一节第 267 页
- “主键约束”一节第 274 页
- “唯一约束”一节第 268 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 251 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 261 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 265 页
- “基于 SQL 的同步”一节第 255 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 266 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 273 页
- “检查约束”一节第 255 页
- “外键约束”一节第 267 页
- “唯一约束”一节第 268 页
- “完整性”一节第 267 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 249 页
- “编码”一节第 247 页
- “归类”一节第 253 页

索引

其它

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

