SYBASE iAnywhere

Mobile Link
クイック・スタート

改訂 2007 年 3 月

版権と商標

目次

はじめに

このマニュアルの内容

このマニュアルでは、セッションベースのリレーショナル・データベース同期システムである Mobile Link について説明します。Mobile Link テクノロジは、双方向レプリケーションを可能にし、モバイル・コンピューティング環境に非常に適しています。

対象読者

このマニュアルは、使用している情報システムに同期を追加したいと考えている SQL Anywhere ユーザと他のリレーショナル・データベース・システムのユーザを対象としています。

始める前に

Mobile Link と他の同期／レプリケーション・テクノロジの比較については、「データ交換テクノロジの概要」『SQL Anywhere 10 - 紹介』を参照してください。

SQL Anywhere のマニュアル

このマニュアルは、SQL Anywhere のマニュアル・セットの一部です。この項では、マニュアル・セットに含まれる各マニュアルと使用法について説明します。

SQL Anywhere のマニュアル

SQL Anywhere の完全なマニュアルは、各マニュアルをまとめたオンライン形式とマニュアル別の PDF ファイルで提供されます。いずれの形式のマニュアルも、同じ情報が含まれ、次のマニュアルから構成されます。

- **『SQL Anywhere 10 - 紹介』** このマニュアルでは、データの管理および交換機能を提供する包括的なパッケージである SQL Anywhere 10 について説明します。SQL Anywhere を使用すると、サーバ環境、デスクトップ環境、モバイル環境、リモート・オフィス環境に適したデータベース・ベースのアプリケーションを迅速に開発できるようになります。
- **『SQL Anywhere 10 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 10 とそれ以前のバージョンに含まれる新機能について説明します。
- **『SQL Anywhere サーバ - データベース管理』** このマニュアルでは、SQL Anywhere データベースの実行、管理、設定について説明します。管理ユーティリティとオプションのほか、データベース接続、データベース・サーバ、データベース・ファイル、バックアップ・プロシージャ、セキュリティ、高可用性、Replication Server を使用したレプリケーションについて説明します。
- **『SQL Anywhere サーバ - SQL の使用法』** このマニュアルでは、データベースの設計と作成の方法、データのインポート・エクスポート・変更の方法、データの検索方法、ストアド・プロシージャとトリガの構築方法について説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルは、SQL Anywhere で使用する SQL 言語の完全なリファレンスです。また、SQL Anywhere のシステム・ビューとシステム・プロシージャについても説明しています。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java プログラミング言語、Visual Studio .NET を使用してデータベース・アプリケーションを構築、配備する方法について説明します。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。
- **『SQL Anywhere 10 - エラー・メッセージ』** このマニュアルでは、SQL Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。
- **『Mobile Link - クイック・スタート』** このマニュアルでは、セッションベースのリレーショナル・データベース同期システムである Mobile Link について説明します。Mobile Link テクノロジは、双方向レプリケーションを可能にし、モバイル・コンピューティング環境に非常に適しています。
- **『Mobile Link - サーバ管理』** このマニュアルでは、Mobile Link アプリケーションを設定して管理する方法について説明します。

- **『Mobile Link - クライアント管理』** このマニュアルでは、Mobile Link クライアントを設定、構成、同期する方法について説明します。Mobile Link クライアントには、SQL Anywhere または Ultra Light のいずれかのデータベースを使用できます。
- **『Mobile Link - サーバ起動同期』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期またはその他のリモート・アクションの開始を可能にする Mobile Link の機能です。
- **『QAnywhere』** このマニュアルでは QAnywhere について説明します。QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームであるほか、モバイル・クライアントや無線クライアント用のメッセージング・プラットフォームでもあります。
- **『SQL Remote』** このマニュアルでは、モバイル・コンピューティング用の SQL Remote データ・レプリケーション・システムについて説明します。このシステムによって、SQL Anywhere の統合データベースと複数の SQL Anywhere リモート・データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。
- **『SQL Anywhere 10 - コンテキスト別ヘルプ』** このマニュアルには、[接続] ダイアログ、クエリ・エディタ、Mobile Link モニタ、SQL Anywhere コンソール・ユーティリティ、インデックス・コンサルタント、Interactive SQL のコンテキスト別のヘルプが収録されています。
- **『Ultra Light - データベース管理とリファレンス』** このマニュアルでは、小型デバイス用 Ultra Light データベース・システムの概要を説明します。
- **『Ultra Light - AppForge プログラミング』** このマニュアルでは、Ultra Light for AppForge について説明します。Ultra Light for AppForge を使用すると、Palm OS、Symbian OS、または Windows CE を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - .NET プログラミング』** このマニュアルでは、Ultra Light.NET について説明します。Ultra Light.NET を使用すると、PC、ハンドヘルド、モバイル、埋め込みデバイスのデータベース・アプリケーションを開発し、これらのデバイスに配備できます。
- **『Ultra Light - M-Business Anywhere プログラミング』** このマニュアルは、Ultra Light for M-Business Anywhere について説明します。Ultra Light for M-Business Anywhere を使用すると、Palm OS、Windows CE、または Windows XP を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対して Web ベースのデータベース・アプリケーションを開発、配備できます。
- **『Ultra Light - C/C++ プログラミング』** このマニュアルでは、Ultra Light C および Ultra Light C++ のプログラミング・インタフェースについて説明します。Ultra Light を使用すると、ハンドヘルド、モバイル、埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。

マニュアルの形式

SQL Anywhere のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル** オンライン・マニュアルには、SQL Anywhere の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含

まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

UNIX オペレーティング・システムでオンライン・マニュアルにアクセスするには、SQL Anywhere のインストール・ディレクトリまたはインストール CD に保存されている HTML マニュアルを参照してください。

- **PDF ファイル** SQL Anywhere の完全なマニュアル・セットは、Adobe Reader で表示できる Adobe Portable Document Format (pdf) 形式のファイルとして提供されています。

  Windows では、PDF 形式のマニュアルはオンライン・マニュアルの各ページ上部にある PDF のリンクから、または Windows の [スタート] メニュー ([スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル - PDF フォーマット]) からアクセスできます。

  UNIX では、PDF 形式のマニュアルはインストール CD にあります。

表記の規則

この項では、このマニュアルで使用されている書体およびグラフィック表現の規則について説明します。

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

- **キーワード**　SQL キーワードはすべて次の例に示す ALTER TABLE のように大文字で表記します。

  ALTER TABLE [*owner.*]*table-name*

- **プレースホルダ**　適切な識別子または式で置き換えられる項目は、次の例に示す *owner* や *table-name* のように表記します。

  ALTER TABLE [*owner.*]*table-name*

- **繰り返し項目**　繰り返し項目のリストは、次の例に示す *column-constraint* のように、リストの要素の後ろに省略記号 (ピリオド 3 つ …) を付けて表します。

  ADD *column-definition* [*column-constraint*, …]

  複数の要素を指定できます。複数の要素を指定する場合は、各要素間をカンマで区切る必要があります。

- **オプション部分**　文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

  この例では、角カッコで囲まれた *savepoint-name* がオプション部分です。角カッコは入力しないでください。

- **オプション**　項目リストから 1 つだけ選択する場合や、何も選択しなくてもよい場合は、項目間を縦線で区切り、リスト全体を角カッコで囲みます。

  [**ASC** | **DESC**]

  この例では、ASC と DESC のどちらか 1 つを選択しても、選択しなくてもかまいません。角カッコは入力しないでください。

- **選択肢**　オプションの中の 1 つを必ず選択しなければならない場合は、選択肢を中カッコで囲み、縦棒で区切ります。

  [**QUOTES** { **ON** | **OFF** }]

  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

オペレーティング・システムの表記規則

- **Windows**　デスクトップおよびラップトップ・コンピュータ用の Microsoft Windows オペレーティング・システムのファミリのことです。Windows ファミリには Windows Vista や Windows XP も含まれます。
- **Windows CE**　Microsoft Windows CE モジュラ・オペレーティング・システムに基づいて構築されたプラットフォームです。Windows Mobile や Windows Embedded CE などのプラットフォームが含まれます。

  Windows Mobile は Windows CE 上に構築されています。これにより、Windows のユーザ・インタフェースや、Word や Excel といったアプリケーションの小規模バージョンなどの追加機能が実現されています。Windows Mobile は、モバイル・デバイスで最も広く使用されています。

  SQL Anywhere の制限事項や相違点は、基盤となっているオペレーティング・システム (Windows CE) に由来しており、使用しているプラットフォーム (Windows Mobile など) に依存していることはほとんどありません。
- **UNIX**　特に記述がないかぎり、UNIX は Linux プラットフォームと UNIX プラットフォームの両方のことです。

ファイルの命名規則

マニュアルでは、パス名やファイル名などのオペレーティング・システムに依存するタスクと機能を表すときは、通常 Windows の表記規則が使用されます。ほとんどの場合、他のオペレーティング・システムで使用される構文に簡単に変換できます。

- **ディレクトリ名とパス名**　マニュアルでは、ドライブを示すコロンや、ディレクトリの区切り文字として使用する円記号など、Windows の表記規則を使用して、ディレクトリ・パスのリストを示します。次に例を示します。

  ```
  MobiLink¥redirector
  ```

  UNIX、Linux、Mac OS X では、代わりにスラッシュを使用してください。次に例を示します。

  ```
  MobiLink/redirector
  ```

  SQL Anywhere がマルチプラットフォーム環境で使用されている場合、プラットフォーム間でのパス名の違いに注意する必要があります。
- **実行ファイル**　マニュアルでは、実行ファイルの名前は、Windows の表記規則が使用され、拡張子 *.exe* が付きます。UNIX、Linux、Mac OS X では、実行ファイルの名前には拡張子は付きません。NetWare では、実行ファイルの名前には、拡張子 *.nlm* が付きます。

  たとえば、Windows では、ネットワーク・データベース・サーバは *dbsrv10.exe* です。UNIX、Linux、Mac OS X では、*dbsrv10* になります。NetWare では、*dbsrv10.nlm* になります。
- **install-dir**　インストール・プロセスでは、SQL Anywhere をインストールするロケーションを選択できます。マニュアルでは、このロケーションは *install-dir* という表記で示されます。

インストールが完了すると、環境変数 SQLANY10 によって SQL Anywhere コンポーネントがあるインストール・ディレクトリのロケーション (*install-dir*) が指定されます。SQLANYSH10 は、SQL Anywhere が他の Sybase アプリケーションと共有しているコンポーネントがあるディレクトリのロケーションを指定します。

オペレーティング・システム別の *install-dir* のデフォルト・ロケーションの詳細については、「SQLANY10 環境変数」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **samples-dir** インストール・プロセスでは、SQL Anywhere に含まれるサンプルをインストールするロケーションを選択できます。マニュアルでは、このロケーションは *samples-dir* という表記で示されます。

  インストールが完了すると、環境変数 SQLANYSAMP10 によってサンプルがあるディレクトリのロケーション (*samples-dir*) が指定されます。Windows の [スタート] メニューから、[プログラム] – [SQL Anywhere 10] – [サンプル・アプリケーションおよびプロジェクト] を選択すると、このディレクトリで [Windows エクスプローラ] ウィンドウが表示されます。

  オペレーティング・システム別の *samples-dir* のデフォルト・ロケーションの詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **環境変数** マニュアルでは、環境変数設定が引用されます。Windows では、環境変数を参照するのに、構文 *%envvar%* が使用されます。UNIX、Linux、Mac OS X では、環境変数を参照するのに、構文 *$envvar* または *${envvar}* が使用されます。

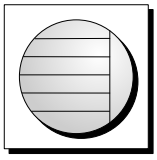
  UNIX、Linux、Mac OS X 環境変数は、*.cshrc* や *.tcshrc* などのシェルとログイン・スタートアップ・ファイルに格納されます。

グラフィック・アイコン

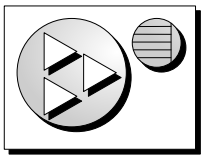
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

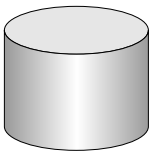
- SQL Anywhere などのデータベース・サーバ



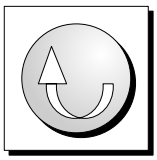
- Ultra Light アプリケーション



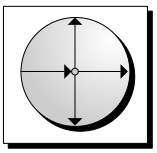
- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



- レプリケーションまたは同期のミドルウェア。ソフトウェアのこれらの部分は、データベース間のデータ共有を支援します。たとえば、Mobile Link サーバ、SQL Remote Message Agent などが挙げられます。



- Sybase Replication Server



- プログラミング・インタフェース

詳細情報の検索／フィードバックの提供

詳細情報の検索

詳しい情報やリソース (コード交換など) については、iAnywhere Developer Network (http://www.ianywhere.com/developer/) を参照してください。

ご質問がある場合や支援が必要な場合は、次に示す Sybase iAnywhere ニュースグループのいずれかにメッセージをお寄せください。

ニュースグループにメッセージをお送りいただく際には、ご使用の SQL Anywhere バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng10 -v** と入力して確認できます。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります (ニュースグループにおけるサービスは英語でのみの提供となります)。以下のニュースグループがあります。

- 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

ニュースグループに関するお断り
iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。
iAnywhere のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

マニュアルに関するご意見、ご提案は、SQL Anywhere ドキュメンテーション・チームの iasdoc@ianywhere.com 宛てに電子メールでお寄せください。このアドレスに送信された電子メールに返信はいたしませんが、お寄せいただいたご意見、ご提案は必ず読ませていただきます。

マニュアルまたはソフトウェアについてのフィードバックは、上記のニュースグループを通してお寄せいただいてもかまいません。

パート I. Mobile Link テクノロジの使用

パート I では、Mobile Link 同期テクノロジの概要と、Mobile Link 同期テクノロジを使用して 2 つ以上のデータベース間でデータをレプリケートする方法について説明します。

第 1 章

Mobile Link 同期について

目次

Mobile Link のクイック・スタート

Mobile Link は、1 つまたは複数の中央のデータ・ソースと断続的に接続する多数のリモート・アプリケーション間でデータを同期するように設計されています。基本的な Mobile Link アプリケーションでは、リモート・クライアントは SQL Anywhere または Ultra Light のデータベースで、中央のデータ・ソースはサポートされている ODBC 準拠のリレーショナル・データベースの 1 つです (SQL Anywhere、Adaptive Server Enterprise、Oracle、Microsoft SQL Server、または IBM DB2)。Mobile Link サーバ API を使用してこのアーキテクチャを拡張すると、サーバ側の同期先の制限を事実上なくすことができます。

すべての Mobile Link アプリケーションで、Mobile Link サーバが同期処理の主要要素です。通常は、Mobile Link リモート・サイトで Mobile Link サーバへの接続を開くと、同期が開始されます。同期中に、リモート・サイト側の Mobile Link クライアントは、前回の同期後にリモート・データベースに対して行われたデータベースの変更をアップロードできます。Mobile Link サーバは、このデータを受信すると、統合データベースを更新し、変更内容を統合データベースからリモート・データベースにダウンロードできます。

Mobile Link アプリケーションの開発を始める最も簡単な方法は、[同期モデル作成] ウィザードを使用することです。「Mobile Link のモデルの概要」 26 ページを参照してください。

入門情報

- 「同期システムの要素」 7 ページ
- 「同期の方法」 『Mobile Link - サーバ管理』
- 「データ交換テクノロジの概要」 『SQL Anywhere 10 - 紹介』

クイック・スタートのためのその他の資料

Mobile Link には、Mobile Link 機能を確かめるために調べたり実行したりできるサンプルが数多く用意されています。Mobile Link のサンプルは、製品とともに *samples-dir¥MobiLink* にインストールされます (*samples-dir* のロケーションについては、「サンプル・ディレクトリ」 『SQL Anywhere サーバ - データベース管理』を参照)。

Mobile Link のコード交換サンプルは、http://ianywhere.codexchange.sybase.com/servlets/ProjectDocumentList?folderID=319 にあります。

また、このマニュアルにはチュートリアルもいくつか用意されています。

- 「Mobile Link CustDB サンプルの解説」 51 ページ
- 「Ultra Light CustDB サンプルの解説」 『Ultra Light - データベース管理とリファレンス』
- 「Mobile Link Contact サンプルの解説」 73 ページ
- 「チュートリアル：Oracle 10g 統合データベースでの Mobile Link の使用」 87 ページ
- 「チュートリアル：Java 同期論理の使用」 97 ページ
- 「チュートリアル：.NET 同期論理の使用」 109 ページ
- 「チュートリアル：カスタム認証用の .NET と Java の使用」 123 ページ
- 「チュートリアル：ダイレクト・ロー・ハンドリングの使用」 135 ページ

Mobile Link の機能

Mobile Link 同期には高い適応性と柔軟性があります。以下に、主な機能の一部を示します。

機能

- **使い始めるのが簡単**　[同期モデル作成] ウィザードを使用すると、簡単に同期アプリケーションを作成できます。このウィザードによって、複雑な同期システムに伴う多くの難解な実装作業が処理されます。Sybase Central モデル・モードを使用すると、同期モデルをオフラインで表示し、簡単なインタフェースで変更を行い、配備オプションを使用してモデルを統合データベースに配備できます。
- **モニタとレポート**　Mobile Link には、同期をモニタする 2 つのメカニズムが用意されています。Mobile Link モニタと統計スクリプトです。
- **パフォーマンス・チューニング**　Mobile Link のパフォーマンスをチューニングするためのメカニズムが多数用意されています。たとえば、競合レベル、アップロードのキャッシュ・サイズ、データベース接続数、ロギングの冗長性、または BLOB のキャッシュ・サイズを調整できます。
- **スケーラビリティ**　Mobile Link はスケーラビリティに優れ、堅牢な同期プラットフォームです。Mobile Link の単一のサーバが数千の同期を同時に処理したり、負荷分散を使用して複数の Mobile Link サーバを同時に稼働したりできます。Mobile Link サーバはマルチスレッド化されており、統合データベースで接続プールを使用します。
- **セキュリティ**　Mobile Link には、既存の認証に統合できるユーザ認証、暗号化、安全な証明書の交換によって機能するトランスポート・レイヤ・セキュリティなど、豊富なセキュリティ・オプションがあります。Mobile Link には、FIPS 承認のセキュリティ・オプションもあります。

アーキテクチャ

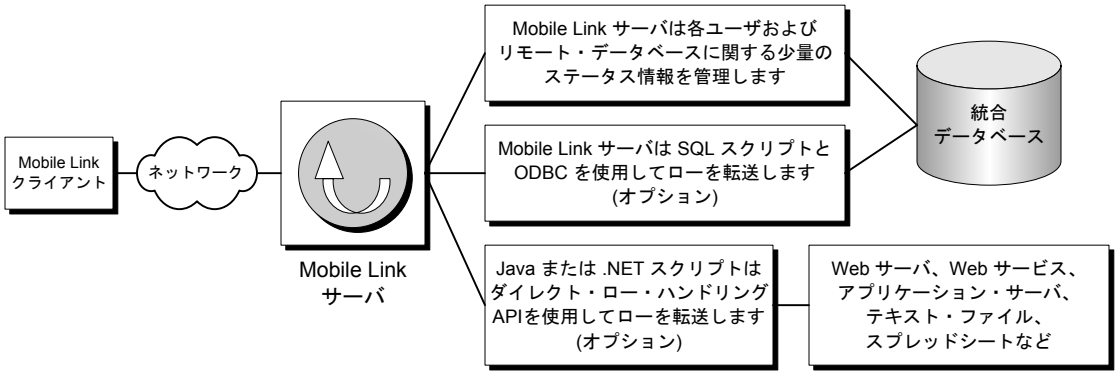
- **データ調整**　Mobile Link によって、データの特定部分を同期対象として選択できます。また、Mobile Link 同期では、異なるデータベースで行われた変更内容の競合を解決できます。同期処理は、SQL、Java または .NET アプリケーションとして作成できる同期論理によって制御されます。この論理の各部は、「**スクリプト**」と呼ばれます。スクリプトを使用すると、たとえば、アップロードされたデータを統合データベースに適用する方法の指定やダウンロード内容を取得するデータベースの指定を行ったり、統合データベースとリモート・データベースとで異なるスキーマや名前を処理したりできます。イベントベースのスクリプト機能により、競合解決、エラー・レポート、ユーザ認証などの機能を含め、同期処理の設計がきわめて柔軟になります。
- **双方向の同期**　すべてのロケーションでデータベースを変更できます。
- **アップロード専用の同期またはダウンロード専用の同期**　アップロードのみ、ダウンロードのみ、または双方向同期を行うことを選択できます。
- **ファイルベースのダウンロード**　ダウンロード内容はファイルとして配布することが可能であり、同期の変更をオフラインで配布できます。これには、適切なデータの適用を保証する機能が含まれます。

- **サーバ起動同期**　Mobile Link 同期は、統合データベースから開始できます。これは、リモート・データベースが統合データベースにデータをアップロードできることに加え、データの更新をリモート・データベースにプッシュできることを意味しています。
- **複数のネットワーク・プロトコル**　TCP/IP、HTTP、または HTTPS を使用して同期を実行できます。Palm デバイスは HotSync を使用して同期できます。Windows CE デバイスは、ActiveSync を使用して同期できます。
- **セッションベース**　すべての変更内容は、単一のトランザクションでアップロードし、単一のトランザクションでダウンロードできます。同期が成功するたびに、統合データベースとリモート・データベースが一貫した状態になります。トランザクションの順序を保持する場合は、リモートの各トランザクションを別個のトランザクションとしてアップロードすることもできます。

  トランザクション全体が同期されるか、トランザクション全体がまったく同期されないかのどちらかになります。これにより、各データベースでトランザクション単位の整合性が確保されます。
- **データの一貫性**　Mobile Link は、緩やかな一貫性方式を使用しています。つまり、変更内容はすべて一貫性が保たれるように各サイトで同期されますが、時間的にはわずかなズレがあるため、ある瞬間だけを見ると、各サイトに存在するデータのコピーが異なる場合もあります。
- **多様なハードウェアとソフトウェアのプラットフォーム**　Mobile Link の統合データベースとして、さまざまな一般的なデータベース管理システムを使用できます (SQL Anywhere、Adaptive Server Enterprise、Oracle、Microsoft SQL Server、または IBM DB2 UDB)。また、Mobile Link サーバ API を使用して任意のデータ・ソースへの同期を定義することもできます。リモート・データベースには、SQL Anywhere または Ultra Light を使用できます。Mobile Link サーバは、Windows または UNIX/Linux プラットフォーム上で動作します。SQL Anywhere は、Windows、Windows CE、または UNIX/Linux 上で動作します。Ultra Light は Palm または Windows CE 上で動作します。

同期システムの要素

Mobile Link 同期では、多くのクライアントが Mobile Link サーバを介して中央のデータ・ソースと同期します。



- **Mobile Link クライアント**　クライアントは、Palm Pilot デバイスや Windows Mobile デバイスなどのハンドヘルド・デバイス、サーバまたはデスクトップ・コンピュータ、またはスマートフォンにインストールできます。Ultra Light と SQL Anywhere データベースという 2 種類のクライアントがあります。単一の Mobile Link インストール環境では、これらのうちのどちらか 1 つまたは両方を使用できます。

  「Mobile Link クライアント」 10 ページを参照してください。

- **ネットワーク**　Mobile Link サーバと Mobile Link クライアント間の接続では、複数のプロトコルを使用できます。

  詳細については、次の項を参照してください。

  - Mobile Link サーバ：「-x オプション」 『Mobile Link - サーバ管理』
  - Ultra Light と SQL Anywhere クライアント：「Mobile Link クライアントのネットワーク・プロトコル・オプション」 『Mobile Link - クライアント管理』

- **Mobile Link サーバ**　同期処理を管理し、すべての Mobile Link クライアントと統合データベース・サーバ間のインタフェースを提供します。

  「Mobile Link サーバ」 『Mobile Link - サーバ管理』を参照してください。

- **統合データベース**　統合データベースには、Mobile Link 同期に必要なシステム・テーブルとプロシージャに加えて、同期するために必要なステータス情報が格納されます。また、通常は同期システムの情報の中核となるコピーも収められています。統合データベースには、SQL Anywhere、Adaptive Server Enterprise、Oracle、DB2、または Microsoft SQL Server を使用できます。

  「中央データ・ソース」 9 ページと「Mobile Link 統合データベース」 『Mobile Link - サーバ管理』を参照してください。

- **ステータス情報**　Mobile Link サーバは、統合データベース内のシステム・テーブルで情報を管理する必要があります。情報の管理は、ODBC 接続を介して行われます。
- **SQL ロー・ハンドリング**　Mobile Link サーバ用に SQL スクリプトを作成すると、サーバではこれらのスクリプトを使用し、ODBC 接続を介して、統合データベースとの間でローが転送されます。

  「同期論理の作成オプション」 20 ページと「Mobile Link 対応の iAnywhere Solutions ODBC ドライバ」 『Mobile Link - サーバ管理』を参照してください。
- **ダイレクト・ロー・ハンドリング**　Mobile Link のダイレクト・ロー・ハンドリング API を使用して、統合データベース以外にオプションで他のデータ・ソースと同期することもできます。このような接続では、さまざまなインタフェースを使用できます。

  「ダイレクト・ロー・ハンドリング」 『Mobile Link - サーバ管理』を参照してください。

リモート・データベースの各テーブルに対して「**同期スクリプト**」を記述し、これらのスクリプトを統合データベースの Mobile Link システム テーブルに保存してください。これらのスクリプトは、アップロード・データに対して行う処理や、ダウンロードするデータを決定します。スクリプトは、テーブル・スクリプトと接続レベル・スクリプトの 2 種類があります。次の項を参照してください。

- 「Mobile Link イベントの概要」 『Mobile Link - サーバ管理』
- 「同期スクリプトの作成」 『Mobile Link - サーバ管理』
- 「同期イベント」 『Mobile Link - サーバ管理』
- 「同期論理の作成オプション」 20 ページ

Mobile Link の開発環境

Mobile Link 同期システムを開発および保守するには次の 2 つの方法があります。

- **モデルを作成する**　Sybase Central Mobile Link プラグインでは、[同期モデル作成] ウィザードとモデル・モードを使用して、統合データベース、リモート・データベース、および同期スクリプトの作成と設定を自動化したり、Mobile Link サーバおよびクライアントを実行したりできます。

  「Mobile Link のモデルの概要」 26 ページを参照してください。
- **システム・オブジェクトを直接変更する**　Mobile Link には、Mobile Link システム・テーブルでデータベース・オブジェクトを作成したり同期論理を登録したりするために使用できる、システム・プロシージャやコマンド・ライン・ユーティリティが用意されています。次の項を参照してください。

  - 「Mobile Link サーバ・システム・プロシージャ」 『Mobile Link - サーバ管理』
  - 「Mobile Link ユーティリティ」 『Mobile Link - サーバ管理』
  - 「Mobile Link サーバ・システム・テーブル」 『Mobile Link - サーバ管理』

Sybase Central 管理モードを使用して、システム・オブジェクトを直接変更することもできます。「管理モード」 31 ページを参照してください。

中央データ・ソース

Mobile Link 同期システムでは、次の 2 種類の中央データ・ソースを使用できます。

- **統合データベース**　統合データベースは必須です。統合データベースには、SQL Anywhere、Adaptive Server Enterprise、Oracle、Microsoft SQL Server、または　IBM DB2 を使用できます。DB2 については、Linux、UNIX、Windows 用はサポートされますが、AS/400 またはメインフレーム用はサポートされません。データベースを統合データベースとして使用できるようにするには、設定スクリプトを実行して、同期に必要な Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、イベントをインストールします。Mobile Link システム・テーブルには、同期に必要なステータス、設定、ユーザ情報が格納されます。ほとんどのアプリケーションでは、統合データベースは同期システム内の情報のマスタ・レポジトリでもあります。

  「Mobile Link 統合データベース」『Mobile Link - サーバ管理』を参照してください。

- **別の中央データ・ソース**　オプションで、中核データのすべて、または一部を、統合データベース以外のデータ・ソースに格納できます。これには、アプリケーション・サーバ、スプレッドシート、Web サーバ、Web サービス、テキスト・ファイルなどを使用できます。Mobile Link のダイレクト・ロー・ハンドリング API を使用して、アップロードの読み込みとダウンロードの作成を行えます。同期論理は Java または .NET で記述できます。

  「ダイレクト・ロー・ハンドリング」『Mobile Link - サーバ管理』を参照してください。

Mobile Link クライアント

各リモート・データベースとそのアプリケーションを、「**Mobile Link クライアント**」といいます。次の 2 種類の Mobile Link クライアントがサポートされています。

- SQL Anywhere
- Ultra Light

「Mobile Link クライアントの紹介」『Mobile Link - クライアント管理』を参照してください。

リモート・データベースには、中央データ・ソースと同じテーブル、サブセット、または全く別のスキーマを格納できます。

異なるスキーマの処理方法の詳細については、「リモート・データベース間でローを分割する」『Mobile Link - サーバ管理』を参照してください。

Mobile Link サーバ

Mobile Link サーバ (mlsrv10) は、Mobile Link クライアントと中央データ・ソースの間に置かれ、クライアントとサーバ間のすべての通信が Mobile Link サーバを介して行われます。

mlsrv10 の実行方法の詳細については、次の項を参照してください。

- 「Mobile Link サーバ」 『Mobile Link - サーバ管理』
- 「Mobile Link サーバのオプション」 『Mobile Link - サーバ管理』

同期処理

「**同期**」とは、Mobile Link クライアントと同期サーバの間で行われるデータ交換処理です。この処理の最中、クライアントは同期サーバとのセッションを確立して維持します。同期に成功した場合、セッションによってリモート・データベースと統合データベースは互いに一貫した状態に保たれます。

同期処理は通常クライアントが開始します。この処理は、Mobile Link サーバとの接続を確立することから始まります。

アップロードとダウンロード

ローをアップロードするために、Mobile Link クライアントが「**アップロード**」を準備し送信します。このアップロードは、Mobile Link クライアント上で前回の同期以後に更新、挿入、または削除されたローすべてのリストを含みます。同様に、ローをダウンロードするために、挿入、更新、削除のリストを含む「**ダウンロード**」を Mobile Link サーバが準備し送信します。

1. **アップロード** デフォルトでは、Mobile Link クライアントは、前回成功した同期以後にリモート・データベースで挿入、更新、または削除されたローを自動的に追跡します。接続が確立すると、Mobile Link クライアントはこれらのすべての変更を記載したリストを同期サーバにアップロードします。

   アップロードは、リモート・データベースで変更されたローに対する新旧のロー値のセットで構成されます(更新には新旧のロー値があります。削除には古い値のみ、挿入には新しい値のみがあります)。ローが更新されたり削除されたりしていれば、前回成功した同期直後に存在していた値が古い値になります。ローが挿入または更新されていれば、現在のローの値が新しい値です。現在の状態に至るまでローが数回変更されていても、その中間値は送信されません。

   Mobile Link サーバは、アップロードを受信して、定義されたアップロード・スクリプトを実行します。デフォルトでは、1 回のトランザクションですべての変更が適用されます。処理が完了すると、Mobile Link サーバはトランザクションをコミットします。

   注意
   Mobile Link は、統合データベースのデフォルトの独立性レベルとして ODBC 独立性レベル SQL_TXN_READ_COMMITTED を使用して動作します。統合データベースで使用される RDBMS がスナップショット・アイソレーションをサポートし、スナップショットがデータベースに対して有効である場合、Mobile Link はデフォルトで、スナップショット・アイソレーションをダウンロードに使用します。「Mobile Link 独立性レベル」『Mobile Link - サーバ管理』を参照してください。

2. **ダウンロード** Mobile Link サーバは、ユーザが作成した同期論理を使用して、Mobile Link クライアント側で挿入、更新、または削除されるローのリストを収集します。これらのローを Mobile Link クライアントにダウンロードします。このリストを収集するために、Mobile Link サーバは統合データベースで新しいトランザクションを開きます。

   Mobile Link クライアントは、ダウンロードを受信します。Mobile Link クライアントは、ダウンロードの着信を、アップロードしたすべての変更内容が統合データベースで正常に適用

されたことの確認とみなします。確認後、Mobile Link クライアントはこれらの変更内容が統合データベースに再送されないようにします。

次に、Mobile Link クライアントは、ダウンロードを自動的に処理して、古いローの削除、新しいローの挿入、変更されたローの更新を行います。これらの変更はすべて、リモート・データベース内の 1 つのトランザクションで適用されます。終了すると、トランザクションをコミットします。

Mobile Link 同期中に情報が明確に交換されることはほとんどありません。クライアントは完全なアップロードを構築してアップロードします。これに応答して、同期サーバは完全なダウンロードを構築してダウンロードします。電話回線または公共無線ネットワークを使用しているために通信が低速で遅延時間が長い場合は、プロトコルのチャティネスの制限が特に重要になります。

参照

- 「Mobile Link イベントの概要」 『Mobile Link - サーバ管理』
- 「アップロード中のイベント」 『Mobile Link - サーバ管理』
- 「ダウンロード中のイベント」 『Mobile Link - サーバ管理』

Mobile Link イベント

Mobile Link クライアントが同期を開始すると、複数の同期イベントが発生します。イベントが発生すると、Mobile Link はその同期イベントに適したスクリプトを探します。このスクリプトには、実行する作業の詳細を示す指示が含まれています。イベントのスクリプトを定義して Mobile Link システム・テーブルに格納している場合は、そのイベントが呼び出されます。

同期イベントとスクリプトの詳細については、次の項を参照してください。

- 「同期スクリプトの作成」 『Mobile Link - サーバ管理』
- 「同期イベント」 『Mobile Link - サーバ管理』

Mobile Link スクリプト

イベントが発生すると、関連するスクリプトを作成していれば、Mobile Link サーバがそのスクリプトを実行します。スクリプトが存在しなければ、次の順位のイベントが発生します。

注意
[同期モデル作成] ウィザードを使用して Mobile Link アプリケーションを作成すると、必要な Mobile Link のスクリプトが自動的に作成されます。ただし、場合によっては、スクリプトを編集する必要があります。

テーブルに対する一般的なアップロード・スクリプトを以下に示します。

イベント	スクリプトの内容例
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}

最初のイベント upload_insert は、emp_id と emp_name を emp テーブルに挿入する upload_insert スクリプトの実行をトリガします。これと同じように、upload_delete と upload_update スクリプトは、同じ emp テーブルでの削除と更新アクションに対して同様の機能を実行します。

ダウンロード・スクリプトはカーソルを使用します。次に、download_cursor スクリプトの例を示します。

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

SQL 同期スクリプト、または SQL 同期スクリプトから呼び出されるプロシージャやトリガで、暗黙的または明示的なコミットまたはロールバックを実行しないでください。SQL スクリプト内に COMMIT 文または ROLLBACK 文があると、同期手順のトランザクションの性質が変化してしまいます。これらの文を使用すると、Mobile Link では、障害が発生した場合にデータの整合性を保証できません。

SQL、Java、または .NET でスクリプトを作成可能

統合データベースのネイティブ SQL ダイアレクトを使用するか、Java または .NET の同期論理を使用して、スクリプトを記述できます。Java と .NET の同期論理を使用すると、Mobile Link サーバによって呼び出されるコードを記述して、データベースへの接続、変数の操作、アップロードされたロー・ハンドリングの直接操作、またはダウンロードへのロー・ハンドリングの追加が可能です。同期の要件に適したクラスとメソッドを持つ Mobile Link サーバ API (Java 用と .NET 用) があります。

同期論理のプログラミングの詳細については、「同期論理の作成オプション」 20 ページを参照してください。

Oracle、Microsoft SQL Server、IBM DB2 UDB、または Adaptive Server Enterprise のデータベースのスクリプト記述など、RDBMS 依存のスクリプト記述については、「Mobile Link 統合データベース」 『Mobile Link - サーバ管理』を参照してください。

スクリプトの格納

SQL スクリプトは統合データベースの Mobile Link システム・テーブルに格納されます。Mobile Link サーバ API を使用して記述されたスクリプトの場合は、完全に修飾されたメソッド名をスクリプトとして格納します。スクリプトを統合データベースに追加する方法は複数あります。

- [同期モデル作成] ウィザードを使用する場合は、プロジェクトを配備するときにスクリプトが Mobile Link システム・テーブルに格納されます。
- 統合データベースを設定するときにインストールされたストアド・プロシージャを使用して、スクリプトを手動でシステム・テーブルに追加できます。
- Sybase Central を使用して、スクリプトをシステム・テーブルに手動で追加できます。

「スクリプトの追加と削除」 『Mobile Link - サーバ管理』を参照してください。

同期処理のトランザクション

Mobile Link サーバは、各 Mobile Link クライアントからアップロードされた変更を、1 回のトランザクションで統合データベースに組み込みます。Mobile Link サーバは、新しいローの挿入、古いローの削除、更新の実行、競合の解決が完了した後で、これらの変更をコミットします。

Mobile Link サーバは、別のトランザクションを使用して、ダウンロードを準備します。このダウンロードには、すべての削除、挿入、更新処理が含まれます。ダウンロード確認を指定した場合は、クライアントが正常なダウンロードを確認すると、Mobile Link サーバがダウンロード・トランザクションをコミットします。ダウンロード確認が指定されていて、アプリケーションで問題が発生したり返信できなかったりする場合は、Mobile Link サーバがダウンロード・トランザクションをロールバックします。デフォルトでは、ダウンロード確認は使用されません。

スクリプト内ではトランザクションをコミットしたりロールバックしたりしないでください
スクリプト内に COMMIT 文または ROLLBACK 文があると、同期手順のトランザクションの性質が変化してしまいます。これらの文を使用すると、Mobile Link では、障害が発生した場合にデータの整合性を保証できません。同期スクリプト、または同期スクリプトから呼び出されるプロシージャやトリガで、暗黙的または明示的なコミットまたはロールバックを実行しないでください。

ダウンロードした情報の追跡

Mobile Link では、リモート・データベースに格納されている最終ダウンロード・タイムスタンプを使用して、ダウンロードの作成方法が簡素化されます。

ダウンロード・トランザクションの主な役割は、統合データベースのローを選択することです。ダウンロードに失敗しても、リモートが同じ最終ダウンロード・タイムスタンプを何度もアップロードするため、データが失われることはありません。

「スクリプトでの最終ダウンロード時間の使用」 『Mobile Link - サーバ管理』を参照してください。

開始時と終了時のトランザクション

Mobile Link クライアントはダウンロードの情報を *1 つのトランザクション*内で処理します。ローを挿入、更新、削除して、リモート・データベースを統合データベースの最新の状態にします。

Mobile Link サーバは、他にトランザクションを 2 つ使用します。1 つは同期の開始時に、もう 1 つは同期の終了時に使用します。これらのトランザクションは、各同期とその処理時間に関する

情報を記録します。したがって、試行された同期、成功した同期、同期にかかった時間についての統計を記録できます。データは処理のさまざまな時点でコミットされるので、これらのトランザクションによって、データを失敗した同期の分析に役立てられるようにコミットできます。

参照

- 「Mobile Link イベントの概要」 『Mobile Link - サーバ管理』
- 「アップロード中のイベント」 『Mobile Link - サーバ管理』
- 「ダウンロード中のイベント」 『Mobile Link - サーバ管理』

同期の障害処理の方法

Mobile Link は、フォールト・トレラントになっています。たとえば、同期中に通信リンクに障害が起きた場合は、リモート・データベースと統合データベースの両方が同じ状態のままになります。

クライアントでは、障害はリターン・コードで示されます。

同期障害の処理方法は、発生したタイミングによって異なります。次に示すケースは、それぞれ異なる方法で処理されます。

- **アップロード中の障害** アップロードの構築中や適用中に障害が起きた場合は、リモート・データベースは同期の起動時とまったく同じ状態のままになります。サーバ側では、適用されたアップロードのすべての部分がロールバックされます。
- **アップロードとダウンロード間の障害** アップロードの完了後、Mobile Link クライアントがダウンロードを受信する前に障害が発生した場合、クライアントはアップロードした変更が統合データベースに適切に適用されたかどうかを確認できません。アップロードが完全に適用されコミットされているか、サーバがアップロード全体を適用する前に障害が起きています。Mobile Link サーバは、統合データベースにある不完全なトランザクションを自動的にロールバックします。

  アップロードされたすべての変更を再送する必要が生じた場合に備え、Mobile Link クライアントはそれらの記録を維持します。Mobile Link クライアントは、次に同期したときに前回のアップロードの状態を要求してから、新しいアップロードを構築します。前回のアップロードがコミットされていない場合は、新しいアップロードに前回のアップロードからの変更をすべて含めます。
- **ダウンロード中の障害** ダウンロード確認が指定されていて、ダウンロードの適用中にリモート・デバイスで障害が起きた場合は、適用されたダウンロードはすべての部分がロールバックされ、リモート・データベースはダウンロード前と同じ状態のままになります。Mobile Link サーバは、統合データベースのダウンロード・トランザクションもロールバックします。

どのような障害が発生しても、データは失われません。Mobile Link サーバと Mobile Link クライアントが障害時のデータ管理を行います。開発者やユーザは、アプリケーション内のデータが一貫性を保持しているかどうか心配する必要はありません。

アップロードの処理方法

Mobile Link サーバが Mobile Link クライアントからアップロードを受信すると、同期が完了するまでアップロード全体が格納されます。このような処理が行われるのは、次の理由によります。

- **ダウンロード・ローのフィルタ**　ダウンロードするローを決定する方法で最も一般的なのは、前回のダウンロード以後に修正されたローをダウンロードすることです。同期中は、ダウンロードよりアップロードが優先されます。アップロード中に挿入または更新されたローが、前回のダウンロード以後に修正されたローになります。

  アップロードの一部として送られたダウンロード・ローから除外する download_cursor スクリプトを記述するのは困難です。このため、Mobile Link サーバがダウンロードからこのようなローを自動的に取り除きます。

- **挿入と更新の処理**　デフォルトでは、アップロード内のテーブルは、参照整合性違反を回避する順序で統合データベースに適用されます。アップロード内のテーブルは、外部キー関係に基づいて並べられます。たとえば、テーブル A とテーブル C の両方がテーブル B のプライマリ・キー・カラムを参照する外部キーを持っている場合は、テーブル B のローに関する挿入や更新が先にアップロードされます。

- **挿入と更新後の削除の処理**　削除は、すべての挿入と更新が適用された後で統合データベースに適用されます。削除が適用されると、テーブルはアップロードの処理とは逆の順序で処理されます。削除されるローが削除される別のテーブルのローを参照すると、この操作の順序では、参照元ローが参照先ローより先に削除されることになります。

- **デッドロック**
  アップロードを統合データベースに適用すると、他のトランザクションとの同時実行性が原因でデッドロックが発生することがあります。そのトランザクションは、他の Mobile Link サーバのデータベース接続からのアップロード・トランザクションの場合や、統合データベースを使用している他のアプリケーションからのトランザクションの場合があります。アップロード・トランザクションがデッドロックされている場合は、そのトランザクションはロールバックされ、Mobile Link サーバが自動的にアップロードを最初から適用し始めます。

> **パフォーマンスに関するヒント**
> 競合をできるだけ避けるように同期スクリプトを書くことが重要です。複数のユーザが同時に同期しているときに競合が起きると、パフォーマンスに大きな影響があります。

参照整合性と同期

すべての Mobile Link クライアントは、ダウンロードをリモート・データベースに組み込むときに参照整合性を確保します。

参照整合性に違反するローがあった場合、デフォルトでは、Mobile Link クライアントはダウンロード・トランザクションを失敗させずに、参照整合性に違反するすべてのローを自動的に削除します。

この機能には次のような利点があります。

- 同期スクリプトの間違いから保護します。スクリプトに柔軟性があると、リモート・データベースの整合性をそこなうローを誤ってダウンロードしてしまうことがあります。Mobile Link クライアントは、介入を要求せずに参照整合性を自動的に管理します。
- この参照整合性のメカニズムを使用して、リモート・データベースから情報を効率的に削除できます。親レコードに削除データを送信するだけで、Mobile Link クライアントはすべての子レコードを自動的に削除します。これにより、Mobile Link がリモート・データベースに送信するトラフィックの量を大幅に減らすことができます。

Mobile Link クライアントは、参照整合性を維持するためにローを明示的に削除する必要がある場合は、通知を行います。

- SQL Anywhere クライアントの場合は、dbmlsync によってログにエントリが書き込まれます。dbmlsync イベント・フックも使用できます。次の項を参照してください。
  - 「sp_hook_dbmlsync_download_ri_violation」『Mobile Link - クライアント管理』
  - 「sp_hook_dbmlsync_download_log_ri_violation」『Mobile Link - クライアント管理』
- Ultra Light クライアントの場合は、SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY 警告が発生します。この警告には、テーブル名のパラメータが含まれます。警告は、参照整合性を維持するために削除されるすべてのローで発生します。同期をそのまま進める場合は、警告を無視してかまいません。警告を明示的に処理する場合は、エラー・コールバック関数を使用して、たとえば、削除されたローをカウントしてその数を取得します。

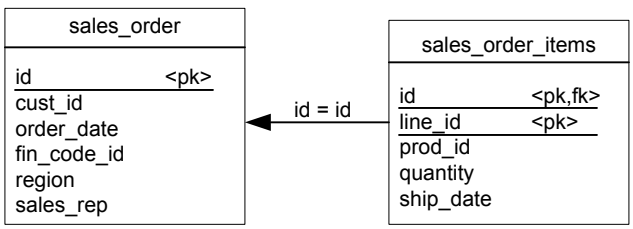
  警告が発生したときに同期を失敗させるには、同期 observer を実装し、observer に (グローバル変数などを使用して) エラー・コールバック関数から信号を送信する必要があります。この場合、同期は observer への次回の呼び出しで失敗します。

トランザクション終了時にチェックされる参照整合性

Mobile Link クライアントは、1 つのトランザクション内のダウンロードから変更を組み込みます。柔軟性をより向上させるために、参照整合性のチェックはこのトランザクションの終了時に行われます。チェックが遅いため、参照整合性に違反する状態を一時的にパスすることがあります。しかし、参照整合性に違反するローは、ダウンロードがコミットされる前に自動的に削除されます。

例

Ultra Light 販売アプリケーションに、他のテーブルとともに次の 2 つのテーブルが含まれているとします。1 つのテーブルには、販売注文が含まれています。もう 1 つのテーブルには、各注文で販売された商品情報が含まれています。この 2 つのテーブルは、次のような関係です。



1 つの注文を削除するために sales_order テーブルに download_delete_cursor を使用すると、削除された受注を示す sales_order_items テーブルのすべてのローが、デフォルトの参照整合性メカニズムによって自動的に削除されます。

この方法には、次のような利点があります。

- sales_order_items テーブルからのローは自動的に削除されるので、sales_order_items スクリプトは必要ありません。
- 同期の効率が向上します。ダウンロード・ローを sales_order_item テーブルから削除する必要がありません。各販売注文に項目がたくさんある場合は、ダウンロードが小さくなるのでパフォーマンスが向上します。この方法は、速度の遅い通信方法を使用しているときに特に役立ちます。

デフォルトの動作の変更

SQL Anywhere クライアントの場合は、sp_hook_dbmlsync_download_ri_violation クライアント・イベント・フックを使用して、参照整合性違反を処理できます。Dbmlsync も、ログにエントリを書き込みます。

次の項を参照してください。

- 「sp_hook_dbmlsync_download_log_ri_violation」『Mobile Link - クライアント管理』
- 「sp_hook_dbmlsync_download_ri_violation」『Mobile Link - クライアント管理』

同期論理の作成オプション

Mobile Link 同期スクリプトは、SQL で記述することも、Java (Java 用 Mobile Link サーバ API を使用) または .NET (.NET 用 Mobile Link サーバ API を使用) で記述することもできます。

サポートされている統合データベース (SQL Anywhere、Adaptive Server Enterprise、Oracle、SQL Server、DB2) と同期する場合は、一般に SQL 同期論理が最適です。

サポート対象外の統合データベースと同期する場合は、Java や .NET が便利です。また、SQL 言語の制限事項やデータベース管理システムの機能によって設計が制限されている場合や、異なる RDBMS タイプ間での移植性が必要な場合も、Java や .NET が便利です。

Java と .NET の同期論理は、SQL 論理と同様に機能します。Mobile Link サーバは、Mobile Link イベントの発生時に SQL スクリプトにアクセスするのと同様に、Java メソッドや .NET メソッドを呼び出すことができます。Java または .NET を使用している場合は、一部の追加処理を実行するイベントを使用できます。ただし、アップロード・ローやダウンロード・ローを直接処理するイベントのスクリプトを処理するときは、Java または .NET の実装が SQL 文字列を返す必要があります。ダイレクト・ロー・ハンドリングで使用される 2 つのイベントを除き、Java と .NET の同期論理では、アップロードとダウンロードに直接アクセスできません。Java または .NET から SQL 文字列で返された内容を Mobile Link が実行します。

handle_UploadData イベントと handle_DownloadData イベントを使用して、統合データベース以外のデータ・ソースと同期するダイレクト・ロー・ハンドリングでは、アップロード・ローとダウンロード・ローは、統合データベースを使用しないで直接操作*されます*。

Java または .NET でのスクリプトの作成を検討する場合のシナリオを以下に示します。

- **ダイレクト・ロー・ハンドリング**　Java と .NET の同期論理では、Mobile Link を使用して、統合データベース以外のデータ・ソース (アプリケーション・サーバ、Web サーバ、ファイルなど) にアクセスできます。
- **認証**　ユーザ認証プロシージャを Java または .NET で記述し、Mobile Link 認証を企業のセキュリティ・ポリシーに組み込みます。
- **ストアド・プロシージャ**　データベースにユーザ定義のストアド・プロシージャを作成する機能がない場合は、必要な機能を実行できるメソッドを Java や .NET で作成できます。
- **外部呼び出し**　プログラムが同期イベント中に外部サーバへの接続を必要とする場合は、Java または .NET 同期論理を使用して、同期イベントによりトリガされるアクションを実行できます。Java または .NET 同期論理は、複数の接続間で共有できます。
- **変数**　データベースに変数を処理する機能がない場合は、接続または同期をとおして持続する変数を Java や .NET で作成できます。また、SQL スクリプトでユーザ定義の名前付きパラメータを使用することもできます。この方法は、すべてのタイプの統合データベースに使用できます。「ユーザ定義の名前付きパラメータ」『Mobile Link - サーバ管理』を参照してください。

Mobile Link サーバ API

Java と .NET 同期論理は、Mobile Link サーバ API を介して使用できます。Mobile Link サーバ API は、Mobile Link 同期用のクラスとインタフェースのセットです。

Java 用 Mobile Link サーバ API には、次の利点があります。

- 統合データベースへの既存の ODBC 接続に JDBC 接続としてアクセスできます。
- JDBC、Web サービス、JNI などのインタフェースを使用して、別のデータ・ソースにアクセスできます。
- 統合データベースへの新規 JDBC 接続を作成し、現在の同期接続の外部でデータベースを変更できます。たとえば、同期接続でロールバックを行う場合でも、これをエラー・ログや監査に使用できます。
- 統合データベースと同期する場合は、Java コードを作成してデバッグしてから Mobile Link サーバで実行できます。多くのデータベース管理システムの SQL 開発環境は、Java アプリケーションが使用可能な環境に比べると初歩的です。
- SQL ロー・ハンドリングとダイレクト・ロー・ハンドリングの両方を使用できます。
- 高度で豊かな Java 言語が提供する多数の既存のコードやライブラリを使用できます。

「Java 用 Mobile Link サーバ API リファレンス」 『Mobile Link - サーバ管理』を参照してください。

.NET 用 Mobile Link サーバ API には、次の利点があります。

- .NET から ODBC を呼び出す iAnywhere クラスを使用して、統合データベースへの既存の ODBC 接続にアクセスできます。
- ADO.NET、Web サービス、OLE DB などのインタフェースを使用して、別のデータ・ソースにアクセスできます。
- 統合データベースと同期する場合は、.NET コードを作成してデバッグしてから、Mobile Link サーバで実行できます。多くのデータベース管理システムの SQL 開発環境は、.NET アプリケーションが使用可能な環境に比べると初歩的です。
- SQL ロー・ハンドリングとダイレクト・ロー・ハンドリングの両方を使用できます。
- .NET Common Language Runtime (CLR) 内でコードが実行されるため、すべての .NET ライブラリ (SQL ロー・ハンドリングとダイレクト・ロー・ハンドリングの両方を含む) にアクセスできます。

「.NET 用 Mobile Link サーバ API リファレンス」 『Mobile Link - サーバ管理』を参照してください。

参照

- 「同期スクリプトの作成」 『Mobile Link - サーバ管理』
- 「同期の方法」 『Mobile Link - サーバ管理』
- 「Java による同期スクリプトの作成」 『Mobile Link - サーバ管理』
- 「.NET での同期スクリプトの作成」 『Mobile Link - サーバ管理』
- 「ダイレクト・ロー・ハンドリング」 『Mobile Link - サーバ管理』

セキュリティ

Mobile Link のインストール環境のように、広範囲の分散システム全体のデータの安全を確保するには、いくつかの要素があります。

- **統合データベースのデータ保護**　統合データベース内のデータは、DBMS のユーザ認証システムとその他のセキュリティ機能で保護できます。

  詳細については、使用している DBMS のマニュアルを参照してください。SQL Anywhere 統合データベースを使用している場合は、「安全なデータの管理」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **リモート・データベースのデータ保護**　SQL Anywhere リモート・データベースを使用している場合、SQL Anywhere のセキュリティ機能を使用してデータを保護できます。このセキュリティ機能は、デフォルトではクライアント／サーバ通信での不正なアクセスを防止するように設計されていますが、データベース・ファイルから直接情報を抽出するという悪質な攻撃に対する保証にはなりません。

  クライアント上のファイルは、クライアントのオペレーティング・システムのセキュリティ機能によって保護されます。

  SQL Anywhere リモート・データベースを使用している場合は、「安全なデータの管理」『SQL Anywhere サーバ - データベース管理』を参照してください。

  Ultra Light データベースを使用している場合は、「Ultra Light でのセキュリティの考慮事項」『Ultra Light - データベース管理とリファレンス』を参照してください。

- **同期中のデータ保護**　Mobile Link クライアントから Mobile Link サーバへの通信は、Mobile Link トランスポート・レイヤ・セキュリティ機能で保護できます。

  「トランスポート・レイヤ・セキュリティ」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **権限のないユーザからの同期システムの保護**　Mobile Link 同期は、パスワードによるユーザ認証システムで保護できます。このメカニズムにより、権限のないユーザはデータを同期できなくなります。

  「Mobile Link ユーザ」『Mobile Link - クライアント管理』を参照してください。

Mobile Link の Mac OS X での実行

Mac OS X 上の Mobile Link 統合データベースを同期するには、ドライバ・マネージャとして SQL Anywhere ODBC ドライバを使用します。「Mac OS X での ODBC データ・ソースの作成」『SQL Anywhere サーバ - データベース管理』を参照してください。

♦ Mobile Link サーバを Mac OS X で開始するには、次の手順に従います。

1. SynCConsole を起動します。

   [Finder] で、SynCConsole をダブルクリックします。SynCConsole アプリケーションは */Applications/SQLAnywhere10* にあります。

2. [ファイル] – [新規] – [Mobile Link サーバ] を選択します。

   サーバ・オプションのダイアログが表示されます。

3. Mobile Link サーバを設定します。

   a. [接続パラメータ] フィールドに次の文字列を入力します。

      ```
      dsn=dsn-name
      ```

      dsn-name は SQL Anywhere ODBC データ・ソース名です。ODBC データ・ソースの作成については、「UNIX と Mac OS X での環境変数の設定」『SQL Anywhere サーバ - データベース管理』を参照してください。

      dsn-name にスペースが含まれている場合は、文字列を二重引用符で囲みます。次に例を示します。

      ```
      dsn="SQL Anywhere 10 Demo"
      ```

   b. [オプション] フィールドは空のままにしておきます。

      [オプション] フィールドを使用すると、Mobile Link サーバ動作をさまざまな面から制御できます。オプションの完全なリストについては、「mlsrv10 の構文」『Mobile Link - サーバ管理』を参照してください。

4. Mobile Link サーバを起動します。

   [起動] をクリックして、サーバを起動します。サーバ・メッセージ・ウィンドウが開き、サーバが同期要求を受け付ける準備ができていることを示すメッセージが表示されます。

♦ dbmlsync を Mac OS X で開始するには、次の手順に従います。

1. SynCConsole を起動します。

   [Finder] で、SynCConsole をダブルクリックします。SynCConsole アプリケーションは */Applications/SQLAnywhere10* にあります。

2. [ファイル] – [新規] – [Mobile Link クライアント] を選択します。

   クライアント・オプションのダイアログが表示されます。

オプションのダイアログには設定オプションが多数用意されていますが、それぞれが dbmlsync コマンド・ライン・オプションに対応しています。完全なリストについては、「dbmlsync 構文」『Mobile Link - クライアント管理』を参照してください。

[ログイン]、[データベース]、[ネットワーク]、[詳細] の各タブのオプションはすべて、Mobile Link クライアントから SQL Anywhere リモート・データベースへの接続を定義しています。ほとんどの場合、[ログイン] タブの ODBC データ・ソースを指定するだけで接続できます。

[DBMLSync] タブのオプションは、Mobile Link サーバへの接続について定義します。この機能がリモート・データベースのパブリケーションおよびサブスクリプションで定義されている場合は、このタブのオプションは空のままにできます。

♦ **サンプル・データベースを Mac OS X で実行するには、次の手順に従います。**

1. *sa_config* 設定スクリプトを準備します。

   詳細については、「UNIX と Mac OS X での環境変数の設定」『SQL Anywhere サーバ - データベース管理』を参照してください。

2. ODBC データ・ソースを設定します。次に例を示します。

   ```
   dbdsn -w  "SQL Anywhere 10 Demo"
   -c "uid=DBA;pwd=sql;dbf=/Applications/SQLAnywhere10/System/demo.db"
   ```

3. Mobile Link サーバを実行します。次に例を示します。

   ```
   dbmlsrv10 -c "dsn=SQL Anywhere 10 Demo"
   ```

第 2 章

Mobile Link のモデル

目次

Mobile Link のモデルの概要

「**同期モデル**」は、Mobile Link アプリケーションを簡単に作成できるツールです。同期モデルは、Sybase Central の **[同期モデル作成] ウィザード**によって作成されるファイルです。

[同期モデル作成] ウィザードを実行すると、スキーマ情報を取得するために統合データベースに接続するように求められます。データベースを統合データベースとして設定していない場合は、Mobile Link システム・テーブルや同期に必要なその他のオブジェクトを作成する設定スクリプトがウィザードによって適用されます。データベースを統合データベースとして設定してある場合は、モデルを展開するまで統合データベースは変更されません。ウィザードを完了すると、データベースへの接続が切断されます。

[同期モデル作成] ウィザードを終了すると、「**モデル・モード**」でモデルが表示されます。モデル・モードを使用してモデルをカスタマイズできます。モデル・モードにあるときは、オフラインでの作業になります。統合データベースは変更されません。モデルは拡張子 *.mlsm* のモデル・ファイルに格納されます。

モデルが完成したら、**[同期モデル展開] ウィザード**を使用してモデルを展開します。[同期モデル展開] ウィザードでは、選択する展開オプションを使用して、同期サーバとクライアントを実行するスクリプト・ファイルを作成できます。展開時に既存のデータベースを変更するか、ウィザードを使用して実行できるファイルを作成するかを選択できます。

展開後に、モデルまたはデータベースをさらに変更を加え、再展開することができます。

モデルの作成

Mobile Link のモデルは、[同期モデル作成] ウィザードを使用して作成できます。

[同期モデル作成] ウィザード

[同期モデル作成] ウィザードでは、Mobile Link 同期アプリケーションを順を追って作成できます。

♦ [同期モデル作成] ウィザードを使用した Mobile Link アプリケーションの設定の概要

1. Sybase Central を開きます。

   [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択します。

2. [ツール] – [Mobile Link 10] – [Mobile Link の同期の設定] を選択します。

   [同期モデル作成] ウィザードが表示されます。

3. モデルの名前とロケーションを選択します。モデルは拡張子 *.mlsm* のモデル・ファイルに格納されます。

   注意： この名前とロケーションは、ウィザードによって作成されるファイルとディレクトリのデフォルトとして使用されます。

4. [プライマリ・キー要件] ページが表示されます。

   このページは、安全のために用意されています。プライマリ・キーを永続的でユニークなものにするよう注意を促します。続行するには、3 つのチェックボックスをオンにする必要があります。このページが今後表示されないようにするには、ページ下部のチェックボックスをオンにします。

   「ユニークなプライマリ・キーの管理」 『Mobile Link - サーバ管理』を参照してください。

5. [統合データベース・スキーマ] ページが表示されます。Mobile Link アプリケーションの統合データベースにするデータベースに接続し、ウィザードでこのデータベースのスキーマ情報を取得できるようにします。

   このデータベースが統合データベースとして設定されていない場合は、設定するように求められます。Mobile Link 設定処理によって、Mobile Link に必要なシステム・オブジェクトがデータベースに追加されます。これらのオブジェクトがすぐに統合データベースに追加されるように設定している場合は、この時点で追加されます(これらのオブジェクトが、後で配備ウィザードを使用したとき、または設定ファイルを適用したときに追加されるように設定することもできます)。

   「統合データベースの設定」 『Mobile Link - サーバ管理』を参照してください。

   統合データベースへの接続は、別の統合データベースに接続したとき、またはウィザードを終了したときに切断されます。この時点から、このウィザードで追加した変更は、統合データベースではなくモデル・ファイルに適用されます。

6. [リモート・データベース・スキーマ] ページが表示されます。リモート・データベース・スキーマは、統合データベースまたは既存のリモート・データベースに基づいて作成できま

す。既存のリモート・データベースには、SQL Anywhere または Ultra Light を使用できます。展開時に、スキーマを新しいまたは既存のリモート・データベースに適用できます。「リモート・データベース」 28 ページを参照してください。

新しい SQL Anywhere リモート・データベースでは、リモート・テーブルの所有者は、統合データベース内の対応するテーブルの所有者と同じになります。別の所有者にするには、その所有者がテーブルを所有する既存のリモート・データベースを使用してください。

7. ウィザードの残りの指示に従います。可能な場合はベスト・プラクティスに基づいたデフォルトの推奨事項が使用されます。すべてのページにオンライン・ヘルプがあります。

8. [完了] をクリックします。

   [完了] をクリックすると、作成したモデルがモデル・モードで開きます。また、統合データベースへの接続が切断されます。これでオフラインで作業している状態になり、モデルに変更を加えることができます。モデルを展開するまで、モデルの外で変更は行われません。そのときまで統合データベースは変更されず、またリモート・データベースは作成または変更されません。

   「モデル・モード」 31 ページと「モデルの配備」 44 ページを参照してください。

注意

- 1 つのモデルに設定できるパブリケーションリは 1 つだけです。「データのパブリッシュ」『Mobile Link - クライアント管理』を参照してください。
- 1 つのモデルに設定できるバージョンは 1 つだけです。「スクリプト・バージョン」 『Mobile Link - サーバ管理』を参照してください。

リモート・データベース

モデルには、リモート・データベースのスキーマが含まれています。このスキーマは、既存のリモート・データベースまたは統合データベースから取得できます。

既存のリモート・データベースは、次のような場合に使用します。

- すでにリモート・データベースがある場合、特にスキーマが統合データベース・スキーマのサブセットではない場合。
- 統合カラムとリモート・カラムのタイプが異なっている必要がある場合。たとえば、統合データベースの NCHAR カラムを Ultra Light リモート・データベースの CHAR カラムにマッピングする必要がある場合。
- リモート・テーブルと統合データベースのテーブルの所有者が異なっている必要がある場合。新しい SQL Anywhere リモート・データベースでは、リモート・テーブルの所有者は、統合データベース内の対応するテーブルの所有者と同じになります。別の所有者にするには、その所有者がテーブルを所有する既存の SQL Anywhere リモート・データベースを使用してください。Ultra Light データベースは、所有者を持ちません。

ヒント：
既存のリモート・データベースのスキーマを変更する必要がある場合は、モデルの外部にあるデータベースを変更してから、[スキーマ更新] ウィザードを実行してモデルを更新します。

モデルの展開時は、モデルでのリモート・スキーマの作成方法にかかわらず、リモート・データベースは 3 つのオプションから選択して作成できます。展開時のリモート・データベースのオプションは次のとおりです。

- **新しいリモート・データベースを作成する。** 展開時に、同期モデルのスキーマが使用され、新しいリモート・データベースが作成されます。
- **ユーザ・テーブルがない既存のリモート・データベースを更新する。** 空のリモート・データベースに展開すると、モデルのリモート・スキーマがデータベース内で作成されます。このオプションは、照合など、デフォルトでないデータベース作成オプションを使用する場合に便利です。

  SQL Anywhere データベースの場合は、「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバ - データベース管理』の説明部分に、データベースの作成後に設定できないオプションのリストを表示できます。

  Ultra Light データベースの場合は、データベース作成後に、データベースのプロパティは変更できません。「Ultra Light で使用する作成時のデータベース・プロパティの選択」『Ultra Light - データベース管理とリファレンス』を参照してください。
- **モデルと同じスキーマを持つ既存のリモート・データベースを更新する。** これは、同期する既存のリモート・データベースがある場合に便利です。既存のリモート・データベースに直接展開した場合、既存のリモート・データベースは変更されません。既存のリモート・データベースのスキーマがモデルのリモート・スキーマと異なる場合に、既存のリモート・データベースに直接展開しようとすると、モデル内のリモート・スキーマを更新するよう要求されます。

SQL Anywhere リモート・データベースでは、テーブルと元のデータベースの所有者は同じです。Ultra Light テーブルは、所有者を持ちません。

参照

- 「モデルの作成」 27 ページ
- 「モデルの配備」 44 ページ

統合データベースの変更

Mobile Link のモデルを使用すると、同期機能に必要なテーブル、カラム、トリガなどの多くのオブジェクトを作成できるので、同期アプリケーションが簡単になります。

これらの変更を統合データベースに加える前に、適用する変更を正確に指定できます。

- [同期モデル作成] ウィザードまたは [同期モデル展開] ウィザードでメッセージが表示された場合に、Mobile Link の設定をインストールする代わりに、設定ファイル (.sql ファイル) を調べ、ファイルを実行して変更を適用できます。

- 統合データベースに直接展開する代わりに、[同期モデル展開] ウィザードを使用して、.sql ファイルに展開できます。これにより、ユーザ自身で検査して実行できるようになります。

「モデルの配備」 44 ページを参照してください。

モデル・モード

[同期モデル作成] ウィザードを終了するか、既存のモデルを開くと、モデルが「**モデル・モード**」で表示されます。モデル・モードを使用してモデルをさらにカスタマイズできます。モデル・モードで作業している場合はオフラインになり、変更はモデル・ファイルに対して行われます。統合データベースまたはリモート・データベースは、モデルを展開するまで変更されません。

管理モード

Sybase Central の Mobile Link プラグインには、モデル・モードと管理モードの 2 種類のモードがあります。これらのモードは、ツールバーの [モード] メニューで切り替えることができます。

管理モードでは、同期アプリケーションをカスタマイズできます。ただし、モデルを展開してモデルの外で変更を行った場合は、変更内容をリバース・エンジニアリングでモデルに戻すことはできません。そのため、モデルを再展開する場合は、管理モードでの変更は行わないでください。

管理モードの詳細については、マニュアルの関連する項を参照してください。管理モードのオンライン・ヘルプの完全なリストについては、「Mobile Link プラグインの管理モードのヘルプ」『SQL Anywhere 10 - コンテキスト別ヘルプ』を参照してください。

テーブル・マッピングとカラム・マッピングの変更

テーブル・マッピングは、どのテーブルを同期するか、テーブルをどのように同期するか、およびリモート・データベースと統合データベースとの間で同期データをどのようにマッピングするかを指定します。

アップロード専用、ダウンロード専用、非同期テーブルまたはカラム

デフォルトでは、Mobile Link は完全に双方向の同期を行います。各テーブルは、アップロード専用またはダウンロード専用に変更できます。テーブルが同期されないように指定することもできます。

モデルでは、テーブルはダウンロード専用にしか指定できず、ダウンロード専用パブリケーションを作成することはできません。これは、1 つのモデルには 1 つのパブリケーションしか設定できないためです。

♦ **テーブル・マッピングの方向を変更するには、次の手順に従います。**

1. モデル・モードで [マッピング] タブを開きます。
2. [テーブル・マッピング] ウィンドウ枠で、リモート・テーブルを選択します。
3. [マッピング方向] ドロップダウン・リストから、次のいずれかを選択します。
   - ♦ 双方向
   - ♦ 統合にのみアップロードします

- リモートにのみダウンロードします
- 同期されない

警告
デフォルトでは、シャドー・テーブルは同期されません。シャドー・テーブルは同期しないでください。

4. 必要に応じて、マッピングする統合テーブルを [統合テーブル] ドロップダウン・リストから選択します。

♦ カラムを同期しないようにするには、次の手順に従います。

1. モデル・モードで [マッピング] タブを開きます。
2. [テーブル・マッピング] ウィンドウ枠で、テーブルを選択します。

   テーブルのカラム情報が、下ウィンドウ枠の [カラム・マッピング] タブに表示されます。
3. カラムを選択します。
4. [マッピング方向] ドロップダウン・リストから、[同期されない] を選択します。

   プライマリ・キーは同期する必要があります。

テーブル・マッピングとカラム・マッピングの変更

既存のリモート・データベースに基づいてモデルを作成した場合は、カラム・マッピングは推測に基づいています。必ず確認し、必要に応じてカスタマイズする必要があります。

♦ テーブル・マッピングを変更するには、次の手順に従います。

1. モデル・モードで [マッピング] タブを開きます。
2. [テーブル・マッピング] ウィンドウ枠で、テーブルを選択します。
3. マッピングされた統合テーブルを変更するには、[統合テーブル] ドロップダウン・リストから別のテーブルを選択します。
4. マッピングされたリモート・テーブルを変更するには、次の手順に従います。

   - 統合データベースに基づいてリモート・ベースを作成した場合は、[リモート・テーブルの新規作成] ダイアログを使用します。「統合データベースに基づいてリモート・データベース・スキーマが作成されている場合にリモート・テーブルを作成する」 33 ページ を参照してください。
   - 既存のリモート・データベースに基づいてリモート・データベースを作成した場合は、[リモート・テーブル] ドロップダウン・リストから別のテーブルを選択します。
5. テーブルのカラム・マッピングを変更するには、テーブルを選択し、下ウィンドウ枠の [カラム・マッピング] タブを開きます。

モデルで作成するリモート・データベースの変更

モデル内のリモート・データベースのスキーマは、次のように変更できます。

統合データベースに基づいてリモート・データベース・スキーマが作成されている場合にリモート・テーブルを作成する

モデルのリモート・データベース・スキーマにテーブルを追加するには、[リモート・テーブルの新規作成] ダイアログを使用します。テーブルがモデルに追加され、同期できるようにマッピングされます。モデル・モードで [リモート・テーブルの新規作成] ダイアログを開くには、[ファイル] – [新規] – [リモート・テーブル] を選択します。

テーブルをリモート・データベースに追加するときに、テーブルが統合データベースに存在しない場合は、統合データベースにテーブルを追加し、[スキーマ更新] ウィザードを実行してから、[リモート・テーブルの新規作成] ダイアログを使用してテーブルをモデルに追加します。モデル・モードで [スキーマ更新] ウィザードを開くには、[ファイル] – [スキーマの更新] を選択します。

「モデル・モードでのスキーマの更新」 42 ページを参照してください。

既存のリモート・データベースに基づいてリモート・データベース・スキーマが作成されている場合にリモート・テーブルを作成する

既存のリモート・データベースに新しいテーブルを作成する場合は、モデル外でリモート・データベースを変更し、[スキーマ更新] ウィザードを使用して、モデル内のリモート・データベース・スキーマを更新します。次に、[マッピング] タブで新しいリモート・テーブルをマッピングします。次の項を参照してください。

- 「モデル・モードでのスキーマの更新」 42 ページ
- 「テーブル・マッピングとカラム・マッピングの変更」 31 ページ

リモート・テーブルとカラムの削除

モデル・モードでは、モデル内のリモート・データベース・スキーマからテーブルとカラムを削除できます。ローを右クリックして [削除] を選択すると、リモート・テーブルまたはカラムに削除対象のマークを付けることができます。リモート・テーブルまたはカラムは、モデルを保存するときに削除されます。リモート・テーブルまたはカラムを削除すると、新しいリモート・データベースに展開したときにも、リモート・テーブルまたはカラムはリモート・データベースで作成されません。

プライマリ・キーは削除できません。

モデル・モードでは、統合データベースからテーブルやローを削除することはできません。統合スキーマを変更するには、モデル・モード以外で統合データベースを変更してから [スキーマ更新] ウィザードを実行します。

ダウンロード・タイプの変更

ダウンロード・タイプには、タイムスタンプ、スナップショット、カスタムのいずれかを設定できます。ダウンロード・タイプは、[マッピング] タブの [テーブル・マッピング] ウィンドウ枠で変更できます。

- **[タイムスタンプベースのダウンロード]** このオプションを選択すると、タイムスタンプベースのダウンロードがデフォルトとして使用されます。最後の同期後に変更されたローのみがダウンロードされます。「タイムスタンプベースのダウンロード」『Mobile Link - サーバ管理』を参照してください。
- **[スナップショット・ダウンロード]** このオプションを選択すると、スナップショット・ダウンロードがデフォルトとして使用されます。最後の同期後に変更されていない場合でも、すべてのローがダウンロードされます。次の項を参照してください。
  - 「スナップショットを使った同期」『Mobile Link - サーバ管理』
  - 「スナップショットを使った同期をいつ行うか」『Mobile Link - サーバ管理』
- **[カスタム・ダウンロード・ロジック]** download_cursor スクリプトと download_delete_cursor スクリプトを自動的に生成せず、独自に作成する場合は、このオプションを選択します。次の項を参照してください。
  - 「同期スクリプトの作成」『Mobile Link - サーバ管理』
  - 「download_cursor スクリプトの作成」『Mobile Link - サーバ管理』
  - 「download_delete_cursor スクリプトの作成」『Mobile Link - サーバ管理』
- **ダウンロード・タイプを変更するには、次の手順に従います。**

1. モデル・モードで [マッピング] タブを開きます。
2. [テーブル・マッピング] ウィンドウ枠で、リモート・テーブルを選択します。
3. [ダウンロード・タイプ] ドロップダウン・リストで、[タイムスタンプ]、[スナップショット]、[カスタム] のいずれかを選択します。
4. [カスタム] を選択すると、[イベント] タブが開きます。
   - テーブルを右クリックし、[イベントに移動] を選択します。
5. 適切なビジネス論理を使用して、download_cursor script スクリプトと download_delete_cursor スクリプトを編集します。

削除の処理方法の変更

スナップショット・ダウンロードを使用している場合、リモート・データベース内のローは、スナップショットがダウンロードされる前にすべて削除されます。タイムスタンプベースのダウンロードを使用している場合は、統合データベースでの削除をリモート・データベースで処理する方法を指定することができます。

統合データベースからローを削除したときに、リモート・データベースからもローを削除する場合は、削除できるようにローを記録しておく必要があります。この処理は、シャドー・テーブルを使用するか、論理削除によって行うことができます。

♦ **削除の処理方法を変更するには、次の手順に従います。**

1. モデル・モードで [マッピング] タブを開きます。
2. [テーブル・マッピング] ウィンドウ枠で、リモート・テーブルを選択します。
3. 統合データベースから削除のダウンロードを行う場合は、[削除のダウンロード] ドロップダウン・リストでチェックボックスをオンにします。統合データベースから削除のダウンロードを行わない場合は、チェックボックスをオフにします。
4. 削除のダウンロードを行う場合は、下ウィンドウ枠で [削除のダウンロード] タブを開きます。

   削除を記録するには、シャドー・テーブルまたは論理削除の使用を設定します。

参照

- 「削除の処理」 『Mobile Link - サーバ管理』
- 「download_delete_cursor スクリプトの作成」 『Mobile Link - サーバ管理』
- 「download_cursor テーブル・イベント」 『Mobile Link - サーバ管理』

ダウンロード・サブセットの変更

各 Mobile Link リモート・データベースでは、統合データベースに格納されているデータのサブセットを同期できます。テーブルごとにダウンロード・サブセットをカスタマイズできます。

ダウンロード・サブセットのオプションは、次のとおりです。

- **[ユーザ]** このオプションを選択すると、Mobile Link ユーザ名によってデータが分割され、登録済みの Mobile Link ユーザごとに異なるデータがダウンロードされます。このオプションを使用するには、Mobile Link ユーザ名が統合データベース上に存在する必要があります。Mobile Link ユーザ名は展開時に選択するため、統合データベースの既存の値と一致する名前を選択できます。サブセットしているテーブルとは異なるテーブルに Mobile Link ユーザ名がある場合は、そのテーブルにジョインする必要があります。
- **[リモート]** このオプションを選択すると、リモート ID によってデータが分割され、リモート・データベースごとに異なるデータがダウンロードされます。このオプションを使用するには、リモート ID が統合データベースにあることが必要です。リモート ID はデフォルトでは GUID として作成されますが、統合データベースの既存の値に一致する ID を設定できます。サブセットしているテーブルとは異なるテーブルにリモート ID がある場合は、そのテーブルにジョインする必要があります。
- **[カスタム]** ダウンロードするローを指定する SQL 式を使用するには、このオプションを選択します。各同期は、SQL 式が true のローのみをダウンロードします。この SQL Anywhere 式は、download_cursor スクリプトで使用される式と同じです。この式は、一部が自動的に作成されます。

♦ **ダウンロード・サブセットを変更するには、次の手順に従います。**

1. モデル・モードで [マッピング] タブを開きます。
2. [テーブル・マッピング] ウィンドウ枠で、リモート・テーブルを選択します。
3. [サブセットのダウンロード] ダウンロード・リストで、[なし]、[ユーザ]、[リモート]、[カスタム] のいずれを選択します。
4. [ユーザ]、[リモート]、[カスタム] を選択する場合は、下ウィンドウ枠で [サブセットのダウンロード] タブを開きます。
5. [ユーザ] または [リモート] を選択した場合は、[サブセットのダウンロード] タブを使用して、Mobile Link ユーザ名またはリモート ID が格納された統合テーブル内のカラムを識別したり、テーブルのジョインを入力して、Mobile Link ユーザ名またはリモート ID を取得することができます。
6. [カスタム] を選択すると、[サブセットのダウンロード] タブに 2 つのテキスト・ボックスが表示されます。このテキスト・ボックスには情報を追加して download_cursor スクリプトを作成できます。完全な download_cursor を作成する必要はありません。ダウンロード・サブセットのジョインやその他の制限を指定するために追加情報を指定する必要があるだけです。

   ♦ download_cursor にその他のテーブルへのジョインが必要な場合は、最初のテキスト・ボックス ([ダウンロード・カーソルの FROM 句に追加するテーブル]) にテーブル名を入力します。ジョインで複数のテーブルが必要な場合は、カンマで区切ります。

   ♦ 2 つ目のテキスト・ボックス ([ダウンロード・カーソルの WHERE 句で使用する SQL 式]) に、ジョインとダウンロード・サブセットを指定する WHERE 条件を入力します。

参照

- 「Mobile Link ユーザの概要」 『Mobile Link - クライアント管理』
- 「リモート ID」 『Mobile Link - クライアント管理』
- 「リモート・データベース間でローを分割する」 『Mobile Link - サーバ管理』
- 「スクリプトでのリモート ID と Mobile Link ユーザ名の使用」 『Mobile Link - クライアント管理』
- 「スクリプトでの最終ダウンロード時間の使用」 『Mobile Link - サーバ管理』
- 「download_cursor スクリプトの作成」 『Mobile Link - サーバ管理』

例 (ユーザ)

たとえば、CustDB の ULOrder テーブルは、ユーザ間で共有できます。デフォルトで、注文は、作成した従業員に割り当てられています。しかし、別の従業員によって作成された注文を確認したい場合があります。たとえば、マネージャは、部署の従業員が作成したすべての注文を確認する必要があるかもしれません。CustDB データベースでは、このような状況に備えるために、ULEmpCust テーブルを使用します。これにより、顧客を従業員に割り当てることができます。マネージャは、ある従業員と顧客の関係における注文をダウンロードします。

どのように処理されたか確認するには、まずダウンロード・サブセットなしで、ULOrder の download_cursor スクリプトを表示します。次に、[マッピング] タブで ULEmpCust テーブルを選択します。タイムスタンプ・ベースのダウンロードを選択します。ダウンロード・サブセットは

選択しないでください。テーブルを右クリックし、[イベントに移動] を選択します。テーブルの download_cursor は、次のようになります。

```
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}
```

[マッピング] タブに戻ります。ULOrder の [サブセットのダウンロード] を [ユーザ] に変更します。下ウィンドウ枠で [サブセットのダウンロード] タブを開きます。[ジョインされている関係テーブル内のカラムを使用する] を選択します。ジョインするテーブルで、ULEmpCust を選択します。一致させるカラムで、emp_id を選択します。ジョイン条件には、emp_id = emp_id を選択します。

一番上のウィンドウ枠でテーブルを右クリックし、[イベントに移動] を選択します。テーブルの download_cursor は、次のようになります。

```
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}
```

例 (カスタム)

たとえば、Mobile Link ユーザによる Customer というテーブルのダウンロードをサブセットし、active=1 である場合だけローをダウンロードするとします。Mobile Link ユーザ名はサブセットしているテーブルに存在しないため、ユーザ名が含まれる SalesRep というテーブルへのジョインを作成する必要があります。

[マッピング] タブの Customer テーブルで、[ダウンロード・タイプ] は [タイムスタンプ]、[サブセットのダウンロード] は [カスタム] を選択します。下ウィンドウ枠で [サブセットのダウンロード] タブを開きます。最初のテキスト・ボックス ([ダウンロード・カーソルの FROM 句に追加するテーブル]) に、次のように入力します。

```
SalesRep
```

2 つ目のテキスト・ボックス ([ダウンロード・カーソルの WHERE 句で使用する SQL 式]) に、次のように入力します。

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


一番上のウィンドウ枠でテーブルを右クリックし、[イベントに移動] を選択します。テーブルの download_cursor は、次のようになります。

```
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 句の最後の行により、Customer から SalesRep へのキー・ジョインが作成されます。

競合の検出と解決の変更

リモート・データベースと統合データベースの両方でローが更新された場合は、次にデータベースを同期するときに競合が発生します。

競合の検出には、次のオプションがあります。

- **[競合検出を実行しない]** このオプションを選択すると、競合は検出されません。アップロードされた更新は、競合を確認しないで適用されます。これにより、統合データベースから現在のローの値をフェッチする必要がなくなるため、更新の同期が高速になることがあります。
- **[ローベースの競合検出]** 最後の同期後に、リモート・データベースと統合データベースの両方でローが更新されていた場合に競合が検出されます。

  このオプションは、upload_fetch スクリプトと upload_update スクリプトを定義します。「upload_fetch スクリプトによる競合の検出」『Mobile Link - サーバ管理』を参照してください。
- **[カラムベースの競合検出]** リモート・データベースと統合データベースの両方で、ローの同じカラムが更新されていた場合に競合が検出されます。

  このオプションは、upload_fetch_column_conflict スクリプトを定義します。「upload_fetch スクリプトによる競合の検出」『Mobile Link - サーバ管理』を参照してください。

  テーブルに BLOB があり、カラムベースの競合検出を選択した場合は、ローベースの競合検出が使用されます。

競合の解決には、次のオプションがあります。

- **[統合]** 先入れ勝ちです。アップロードされた更新が競合する場合は拒否されます。
- **[リモート]** 後入れ勝ちです。アップロードされた更新が常に適用されます。

  このオプションは、カラムベースの競合検出だけに使用してください。それ以外の場合に使用すると、競合検出を選択しない方がパフォーマンスがよくなったり、同程度になったりします。
- **[タイムスタンプ]** 最新の更新が適用されます。このオプションを使用するには、テーブルのタイムスタンプ・カラムを作成し、維持する必要があります。このタイムスタンプ・カラムに、ローが最後に変更された時刻が記録されます。このカラムは、統合データベースとリモー

ト・データベースの両方に存在する必要があります。これを機能させるには、リモート・データベースと統合データベースで同じタイム・ゾーン (UTC を推奨) を使用し、かつクロックが同期されている必要があります。

- **[カスタム]** 独自の resolve_conflict スクリプトを作成します。この処理は、ウィザード終了後に [イベント] タブで行います。

  「resolve_conflict スクリプトによる競合の解決」 『Mobile Link - サーバ管理』を参照してください。

- **競合の検出と解決をカスタマイズするには、次の手順に従います。**

1. モデル・モードで [マッピング] タブを開きます。
2. [テーブル・マッピング] ウィンドウ枠で、リモート・テーブルを選択します。
3. [競合の検出] ドロップダウン・リストで、[なし]、[ローベース]、[カラムベース] のいずれかを選択します。[なし] を選択した場合は、これで完了です。
4. [ローベース] または [カラムベース] を選択した場合は、[競合の解決] ドロップダウン・リストから [統合]、[リモート]、[タイムスタンプ]、[カスタム] のいずれかを選択します。
5. [タイムスタンプ] を選択した場合は、下ウィンドウ枠の [競合の解決] タブを開き、使用するタイムスタンプ・カラムの名前を入力します。
6. [カラム] を選択した場合は、[イベント] タブを開き、テーブルの resolve_conflict スクリプトを作成します。

参照

- 「競合の解決」 『Mobile Link - サーバ管理』

モデル内のスクリプトの変更

Mobile Link モデル・モードの [イベント] タブでは、次の操作を実行できます。

- [同期モデル作成] ウィザードで生成されたスクリプトを表示し、変更する。
- 新しいスクリプトを作成する。

[イベント] タブの上部には、選択したスクリプトが属すグループが表示されます。利便性のために、単一テーブルのスクリプトはすべて 1 つにまとめられています。[イベント] タブの上部には、選択したスクリプトの名前のほか、[同期モデル作成] ウィザードで生成されたかどうか、ユーザ定義かどうか、生成されたスクリプトが上書きされたかどうかも表示されます。また、同期論理が SQL、.NET、Java のどれで作成されているのかも表示されます。

スクリプトを追加または変更すると、スクリプトを完全に制御できるようになるので、モデル・モードで関連の設定を変更しても、変更が自動的に行われることはありません。たとえば、モデルの download_delete_cursor を変更した後にモデル・モードで [削除のダウンロード] の選択を解除しても、カスタマイズした download_delete_cursor に影響はありません。

[ファイル] メニューのオプションを使用すると、変更したデフォルトのスクリプトをリストアしたり、追加した新しいスクリプトを削除することができます。リストアするスクリプトを選択してから [ファイル] を選択して、オプションを表示します。

特定のテーブルのスクリプトを検索するには、[マッピング] タブを開いてローを選択し、[ファイル] – [イベントに移動] を選択します。適切なテーブルで、[イベント] タブが開きます。

モデル・モードでの外部サーバに対する認証

外部の POP3、IMAP、または LDAP サーバに対して認証するには、モデル・モードで [認証] タブを開き、[この同期モデルのカスタム認証を有効にする] を選択します。

ホストとポートの情報を入力するか、LDAP サーバの場合は LDAP サーバの URL を入力します。

これらのフィールドの詳細については、「外部認証識別符号プロパティ」 『Mobile Link - クライアント管理』を参照してください。

モデル・モードでのサーバ起動同期の設定

サーバ起動同期を使用すると、統合データベースで変更が行われたときに、クライアントで同期を起動することができます。モデル・モードには、サーバ起動同期を設定するための方法が用意されています。この方法では、サーバ起動同期の設定と実行を簡単に行うことができますが、バージョンが限定されています。

[通知] タブ

♦ サーバ起動同期を設定するには、次の手順に従います (Sybase Central モデル・モードと [同期モデル展開] ウィザードの場合)。

1. [同期モデル作成] ウィザードを使用して、Mobile Link モデルを作成します。
2. モデルがモデル・モードで開いたら、モデル上部の [通知] タブを開きます。
3. [サーバ起動同期を有効にする] を選択します。
4. 通知に使用する統合データベースのテーブルを選択します。

   このテーブル内のデータが変更されると、リモート・データベースに通知が送信され、同期がトリガされます。通知により、同期がトリガされます。

   この目的のためには、タイムスタンプベースのダウンロード・カーソルを定義したテーブルだけを選択できます (デフォルト)。通知はダウンロード・カーソルの内容に基づきます。

   「download_cursor スクリプトの作成」 『Mobile Link - サーバ管理』を参照してください。
5. ポーリング間隔を選択します。これは、ポーリングからポーリングまでの時間です。事前に定義されたポーリング間隔を選択するか、間隔を入力できます。デフォルトは 30 秒です。

   データベース接続が失われた場合、Notifier はデータベースが再び使用可能になった後に、この最初のポーリング間隔で自動的にリカバリを行います。

6. オプションで、Notifier のデータベース接続の独立性レベルを変更します。デフォルトは、コミットされた読み込みです。

   独立性レベルの設定結果に注意してください。レベルが高くなると競合が増加し、パフォーマンスが逆に低下することがあります。独立性レベル 0 に設定すると、コミットされていないデータ (最終的にロールバックされるデータ) を読み込むことができます。

7. オプションで、通知が送信されるゲートウェイを変更します。デフォルトは、default_device_tracker ゲートウェイです。「ゲートウェイと Carrier」 『Mobile Link - サーバ起動同期』を参照してください。

♦ サーバ起動同期でモデルを展開するには、次の手順に従います。

1. モデルを展開します。

   a. [ファイル] – [展開] を選択します。

   b. [同期モデル展開] ウィザードの指示に従います。

      「モデルの配備」 44 ページを参照してください。

   c. [サーバによって開始される同期のリスナ] ページで、Listener のオプションを選択します。

2. モデルが展開されます。作成されたファイルの詳細については、「展開したモデルの同期」 46 ページを参照してください。

3. サーバ起動同期を使用するには、次の操作を行います。

   a. Mobile Link サーバを起動します。

   b. 最初の同期を行います (まだ行っていない場合)。

   c. Listener を起動します。

4. [同期モデル作成] ウィザードを最初に開始したときに選択したディレクトリに移動します。このディレクトリには、拡張子 *.mlsm* のモデルが格納されています。また、次のサブディレクトリもあります。

   - *¥mlsrv*
   - *¥remote*
   - *¥consolidated*

サーバ起動同期について (モデル・モード以外の場合)

モデル・モードのバージョンのサーバ起動同期では、Mobile Link サーバがテーブルに対して download_cursor スクリプトを使用して、同期をいつ開始するかを判断します。これは、download_cursor スクリプトを使用して Notifier に request_cursor を生成することによって行われます。このバージョンのサーバ起動同期を使用する場合、request_cursor はカスタマイズできません。

「download_cursor スクリプトの作成」 『Mobile Link - サーバ管理』と「request_cursor プロパティ」 『Mobile Link - サーバ起動同期』を参照してください。

モデル・モードでは、通知を送信するために Device Tracker ゲートウェイも設定します。ゲートウェイはカスタマイズできます。「ゲートウェイと Carrier」 『Mobile Link - サーバ起動同期』を参照してください。

Sybase Central モデル・モードには、サーバ起動同期の簡略バージョンを設定するための方法が用意されていますが、完全なバージョンを設定することもできます。

サーバ起動同期の完全な実装については、Mobile Link - サーバ起動同期 『Mobile Link - サーバ起動同期』を参照してください。

モデル・モードを使用しないでサーバ起動同期を設定するときに何が必要かについては、「サーバ起動同期のクイック・スタート」 『Mobile Link - サーバ起動同期』を参照してください。

モデル・モードでのスキーマの更新

[スキーマ更新] ウィザードでは、モデル内の統合データベースとリモート・データベースのスキーマを更新できます。

[スキーマ更新] ウィザードは、モデルを展開した後と、次の場合に最適です。

- モデルに追加する必要があるリモート・データベースのスキーマを変更した場合。
- モデルに追加する必要がある統合データベースのスキーマを変更した場合。

  たとえば、1 つまたは複数のテーブルにタイムスタンプ・ベースのダウンロードを作成したモデルを再展開する前に、[スキーマ更新] を実行したとします。この場合は、以前に展開したときに、タイムスタンプ・カラムまたはシャドー・テーブルを追加して、統合データベースのスキーマを変更したために、スキーマを更新する必要があります。

[リモート・テーブルの新規作成] ダイアログはリモート・テーブルをモデルに追加しますが、[スキーマ更新] ウィザードはこれと異なり、テーブルをマッピングしません。[マッピング] タブを使用して、テーブル・マッピングを作成する必要があります。「モデルで作成するリモート・データベースの変更」 33 ページを参照してください。

♦ スキーマを更新するには、次の手順に従ってください。

1. モデル・モードで、[ファイル] – [スキーマの更新] を選択します。

   [スキーマ更新] ウィザードが表示されます。

2. 更新するスキーマを選択します。

   [スキーマ更新] ウィザードは、モデル内のスキーマとデータベースのスキーマを比較します。

   - **[統合データベース・スキーマ]** モデル内の統合スキーマは更新されます。モデル内のリモート・スキーマは変更されません。

- **[リモート・データベース・スキーマ]** モデル内のリモート・スキーマは更新されます。モデル内の統合スキーマは変更されません。
- **[統合データベース・スキーマとリモート・データベース・スキーマ]** 統合およびリモートのスキーマの両方が、既存のデータベースのスキーマに一致するようにモデル内で更新されます。

3. ウィザードの指示に従います。ページごとにオンライン・ヘルプがあります。
4. [完了] をクリックします。

   [完了] をクリックすると、統合データベースおよびリモート・データベースへの接続が切断されます。したがって、作業はオフラインになります。モデルを展開するまで、モデルの外で変更は行われません。そのときまで統合データベースは変更されず、またリモート・データベースは作成または変更されません。
5. [マッピング] タブで、新しいリモート・テーブルをマッピングします。「テーブル・マッピングとカラム・マッピングの変更」 31 ページを参照してください。

モデルの配備

モデルを試す準備ができたら、[同期モデル展開] ウィザードを使用して展開します。展開できるものは、次のように複数あります。

- 統合データベースの変更内容
- SQL Anywhere または Ultra Light リモート・データベース (データベースを作成するか、既存の空のデータベースにテーブルを追加するか、リモート・テーブルがすでに格納されている既存のデータベースを使用)
- モデルを展開するバッチ・ファイル
- Mobile Link サーバと Mobile Link クライアントを実行するためのバッチ・ファイル
- サーバによって開始される同期の設定

モデルを配備すると、モデル・ファイルは保存されます。

統合データベースへの配備

展開ウィザードでは、次の 2 つの方法で統合データベースに配備できます。

- Mobile Link システム・テーブルにデータを追加し、必要なシャドー・テーブル、カラム、トリガ、ストアド・プロシージャを作成することで、モデルを統合データベースに直接適用します。必要に応じて、Mobile Link アプリケーションを実行するバッチ・ファイルも作成できます。
- 同じ変更内容をすべて含む SQL ファイルを作成します。このファイルは、いつでも確認、変更、実行することができます。結果は変更を直接適用する場合と同じです。

> **注意**
> 配備時にシャドー・テーブルを作成した場合は、シャドー・テーブルが作成されるベース・テーブルの所有者または管理者として、統合データベースに接続する必要があります。

リモート・データベースの配備

既存のリモート・データベースを使用するか、ウィザードでリモート・データベースを作成できます。ウィザードでは、リモート・データベースを直接作成するか、リモート・データベースを作成する SQL ファイルを作成できます。

ウィザードでは、モデルで指定したデータベース所有者を使用して、デフォルトのデータベース作成オプションでリモート・データベース (SQL Anywhere または Ultra Light) が作成されます。また、[同期モデル作成] ウィザードを使用しないでカスタム設定でリモート・データベースを作成し、ウィザードを使用して必要なリモート・テーブルを追加するか、すでにリモート・テーブルがある既存のリモート・データベースに配備することもできます。

同期ツールを実行するバッチ・ファイルの配備

ウィザードでは、次のバッチ・ファイルを作成できます。

- 指定したオプションで Mobile Link サーバを実行するバッチ・ファイル

- SQL Anywhere のリモート・データベースの場合、指定したオプションで dbmlsync を実行するバッチ・ファイル
- Ultra Light のリモート・データベースの場合、指定したオプションで ulsync を実行するバッチ・ファイル。ulsync は、同期のテストに使用されるので、正常に機能する Ultra Light アプリケーションがない場合に初めて同期するときに役立ちます。
- サーバによって開始される同期を設定する場合は、Notifier と Listener を実行するバッチ・ファイル

♦ モデルを展開するには、次の手順に従います。

1. モデル・モードで、[ファイル] – [展開] を選択します。

   [同期モデル展開] ウィザードが表示されます。

2. ウィザードの指示に従います。ページごとにオンライン・ヘルプがあります。
3. 完了したら、選択した変更内容が展開されます。同じ名前のファイルがすでにあった場合は、上書きされます。
4. アプリケーションを同期するには、「展開したモデルの同期」 46 ページを参照してください。

モデルの再展開

モデルは、展開後に変更できます。これを実行するには、モデル・モードで変更してから再展開します。Sybase Central の管理モードで配備したアプリケーションを変更するか、システム・プロシージャやその他の方法でデータベースを直接変更することもできます。ただし、モデル・モード外で配備したモデルを変更した場合は、変更内容をリバース・エンジニアリングでモデルに戻すことはできません。モデルを再展開するときは、モデル外で作成された同期アプリケーションへの変更は上書きされます。

リモート・データベースまたは統合データベースのスキーマを変更する場合は、モデル内のスキーマを更新してから再展開する必要があります。配備によりスキーマが変更されてしまう場合があるので、その他の変更を行っていないときでも、スキーマを更新する必要があります。たとえば、統合データベース内の各同期テーブルにタイムスタンプ・カラムを追加するモデルを展開する場合は (モデル作成時のデフォルトの動作)、モデル内の統合スキーマを更新してから再展開する必要があります。同様に、統合データベースにテーブルを追加してから再展開する場合は、モデル内の統合スキーマを更新してから、新しいリモート・テーブルを作成する必要があります。

[スキーマ更新] ウィザードを実行するには、[ファイル] – [スキーマの更新] を選択します。

「モデル・モードでのスキーマの更新」 42 ページを参照してください。

展開したモデルの同期

モデルを展開するときは、[同期モデル作成] ウィザードの最初の画面で選択したロケーションにディレクトリやファイルがオプションで作成されます。このファイルやディレクトリは、このときに選択したモデル名に従って名前が付けられます。

モデルの名前を **MyModel** にし、*c:¥SyncModels* に保存したとします。この場合、次のようなファイルが作成されます。作成されるファイルは選択した展開オプションによって異なります。

ディレクトリ (この例の名前とロケーションに基づく)	説明と内容 (この例の名前に基づく)
c:¥SyncModels	*MyModel.mlsm* として保存されたモデル・ファイルがあります。
c:¥SyncModels¥MyModel	展開ファイルを含むフォルダがあります。
c:¥SyncModels¥MyModel¥consolidated	統合データベース用の展開ファイルがあります。 ◆ *MyModel_consolidated.sql* - 統合データベースの設定に使用する SQL ファイル ◆ *MyModel_consolidated.bat* - SQL ファイルを実行するためのバッチ・ファイル
c:¥SyncModels¥MyModel¥mlsrv	Mobile Link サーバ用の展開ファイルがあります。 ◆ *MyModel_mlsrv.bat* - Mobile Link サーバを実行するためのバッチ・ファイル。サーバ起動同期を設定した場合は、Notifier も起動されます。
c:¥SyncModels¥MyModel¥remote	リモート・データベース用の展開ファイルがあります。 ◆ *dblsn.txt* - サーバ起動同期を設定した場合の、Listener のオプション設定を含むテキスト・ファイル。*MyModel_dblsn.bat* によって使用されます。 ◆ *MyModel_dblsn.bat* - サーバ起動同期を設定した場合の、Listener を実行するためのバッチ・ファイル ◆ *MyModel_dbmlsync.bat* - SQL Anywhere リモート・データベースを展開した場合の、SQL Anywhere データベースを dbmlsync と同期するバッチ・ファイル ◆ *MyModel_remote.bat* - *MyModel_remote.sql* を実行するためのバッチ・ファイル ◆ *MyModel_remote.db* - 新しい SQL Anywhere リモート・データベースを作成する場合のデータベース ◆ *MyModel_remote.sql* - 新しい SQL Anywhere リモート・データベースを設定する場合の SQL ファイル ◆ *MyModel_remote.udb* - 新しい Ultra Light リモート・データベースを作成する場合のデータベース ◆ *MyModel_ulsync.bat* - Ultra Light データベースを展開した場合の、ulsync ユーティリティを使用して Ultra Light リモート・データベースとの同期をテストするためのバッチ・ファイル

バッチ・ファイルの実行

[同期モデル展開] ウィザードで作成されるバッチ・ファイルは、コマンド・ラインから実行する必要があります。このとき、接続情報を指定する必要があります。これらのバッチ・ファイルを実行する前に ODBC データ・ソースを作成する必要がある場合があります。

「ODBC データ・ソースの使用」 『SQL Anywhere サーバ - データベース管理』を参照してください。

♦ **バッチ・ファイルを使用してモデルを同期するには、次の手順に従います。**

1. 統合データベースで Mobile Link 設定スクリプトを実行していない場合は、実行してから展開します。

   「統合データベースの設定」 『Mobile Link - サーバ管理』を参照してください。

2. [同期モデル展開] ウィザードを実行したときに、([統合データベースの展開先] ページで) 後で実行できるようにファイルを作成する場合は、モデルの *consolidated* サブフォルダにあるバッチ・ファイルを実行します。このファイルによって、同期スクリプト、シャドー・テーブル、トリガなど、統合データベースに作成することを選択したオブジェクトがすべて作成されます。また、Mobile Link ユーザが統合データベースに登録される場合もあります。

   このファイルを実行するには、*consolidated* ディレクトリに移動し、*_consolidated.bat* で終わるファイルを実行します。このとき、接続情報を指定する必要があります。たとえば、次のように入力します。

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

3. [同期モデル展開] ウィザードを実行したときに、後で実行するためのファイルを作成する場合 ([新しい SQL Anywhere リモート・データベース] ページまたは [新しい Ultra Light リモート・データベース] ページ) は、*remote* ディレクトリにあるバッチ・ファイルを実行します。このファイルによって、テーブル、パブリケーション、サブスクリプション、Mobile Link ユーザなど、リモート・データベースに作成することを選択したオブジェクトがすべて作成されます。

   このファイルを実行するには、*remote* ディレクトリに移動し、*_remote.bat* で終わるファイルを実行します。たとえば、次のように入力します。

   ```
   MyModel_remote.bat
   ```

4. *mlsrv¥MyModel_mlsrv.bat* を実行して、Mobile Link サーバを起動します。サーバ起動同期を設定した場合は、このファイルによって Notifier も起動されます。このとき、接続情報を指定する必要があります。たとえば、次のように入力します。

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

5. 同期を実行します。

   SQL Anywhere リモート・データベースの場合

   ♦ DBA 以外のユーザ (推奨) に REMOTE DBA 権限を付与します。たとえば、Interactive SQL で次のコマンドを実行します。

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


REMOTE DBA 権限を持つユーザで接続します。

- *remote* ディレクトリにあるリモート・データベースを起動します。たとえば、次のように入力します。

  ```
  dbeng10 MyModel_remote.db
  ```

- SQL Anywhere Mobile Link クライアント dbmlsync を起動します。*remote* ディレクトリに存在する *_dbmlsync.bat* で終わるファイルを実行します。 このとき、接続情報を指定する必要があります。たとえば、次のように入力します。

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

Ultra Light リモート・データベースの場合

- 同期をテストするには、*remote* ディレクトリに存在する *_ulsync.bat* で終わるファイルを実行します。
- Ultra Light アプリケーションを実行することもできます。

6. サーバ起動同期を設定した場合は、最初の同期を行ってから、Listener を起動します。最初の同期は、リモート ID ファイルを作成するために必要です。Listener を開始するには、*remote* ディレクトリに存在する *_dblsn.bat* で終わるファイルを実行します。たとえば、次のように入力します。

   ```
   MyModel_dblsn.bat
   ```

参照

- 「GRANT REMOTE DBA 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「dbmlsync のパーミッション」『Mobile Link - クライアント管理』

パート II. Mobile Link チュートリアル

パート II では、Mobile Link テクノロジの設定方法と使用方法を示すチュートリアルがあります。チュートリアルは、新しいユーザのための入門用チュートリアルから、高度な機能の使用方法の説明にまで及びます。

第 3 章

Mobile Link CustDB サンプルの解説

目次

Mobile Link CustDB チュートリアルの概要

CustDB は販売管理アプリケーションです。CustDB サンプルは、Mobile Link 開発者にとって貴重なリソースです。このサンプルは、Mobile Link アプリケーションの開発時に必要なさまざまな手法の実装例を示しています。

このアプリケーションを使用して、いくつかの一般的な同期方法を説明します。この章で説明することを最大限に活用するために、サンプル・アプリケーションのソース・コードを読んで理解してください。

サポートされるオペレーティング・システムとデータベースの種類ごとに、CustDB が用意されています。

CustDB のロケーションと設定手順については、「CustDB 統合データベースの設定」 54 ページを参照してください。

シナリオ

CustDB のシナリオを以下に示します。

統合データベースは、本社に配置されています。以下のデータが統合データベースに格納されます。

- 同期されるメタデータを格納する Mobile Link システム・テーブル。同期論理を実装する同期スクリプトが含まれています。
- すべての顧客、製品、注文の情報を含み、ベース・テーブルのローに格納される CustDB のデータ

リモート・データベースは、モバイル管理者用と営業担当者用の 2 種類があります。

各モバイル営業担当者のデータベースにはすべての製品とその営業担当者に対応する注文のみが格納されていますが、モバイル管理者のデータベースにはすべての製品と注文が格納されています。

同期の設計

CustDB サンプル・アプリケーションでは、以下の機能を使用して同期を行います。

- **完全なテーブルのダウンロード**　ULProduct テーブルのすべてのローとカラムが、リモート・データベースでも完全に共有される。
- **カラムのサブセット**　ULCustomer テーブルの一部のカラムのすべてのローが、リモート・データベースでも共有される。
- **ローのサブセット**　ULOrder テーブルから、リモート・ユーザごとに異なるロー・セットを取得する。

  ローのサブセットの詳細については、「リモート・データベース間でローを分割する」『Mobile Link - サーバ管理』を参照してください。

- **タイムスタンプベースの同期**　最後のデバイス同期以降に実行された統合データベースに対する変更を識別する方法。ULCustomer テーブルと ULOrder テーブルは、タイムスタンプベースの方法で同期される。

  「タイムスタンプベースのダウンロード」『Mobile Link - サーバ管理』を参照してください。

- **スナップショットを使った同期**　同期を実行するたびにすべてのローをダウンロードする単純な同期方法。ULProduct テーブルは、この方法で同期される。

  「スナップショットを使った同期」『Mobile Link - サーバ管理』を参照してください。

- **プライマリ・キー・プール (ユニークなプライマリ・キー管理用)**　プライマリ・キーの値を、Mobile Link インストール環境全体で確実にユニークにする必要がある。このアプリケーションで使われるプライマリ・キー・プール・メソッドは、プライマリ・キーをユニークにする方法の 1 つである。

  「プライマリ・キー・プールの使用」『Mobile Link - サーバ管理』を参照してください。

  プライマリ・キーをユニークにする他の方法については、「ユニークなプライマリ・キーの管理」『Mobile Link - サーバ管理』を参照してください。

CustDB テーブルの ER (実体関連) 図については、「CustDB サンプル・データベースについて」『SQL Anywhere 10 - 紹介』を参照してください。

詳細情報

- 「Ultra Light CustDB サンプルの解説」『Ultra Light - データベース管理とリファレンス』

CustDB の設定

この項では、CustDB サンプル・アプリケーションとサンプル・データベースを作成するコードを部分的に説明します。これらのコードを以下に示します。

- サンプル SQL スクリプト。*samples-dir¥MobiLink¥CustDB* にあります。
- アプリケーションのコード。*samples-dir¥UltraLite¥CustDB* にあります。
- プラットフォーム固有のユーザ・インタフェースのコード。*samples-dir¥UltraLite¥CustDB* の各オペレーティング・システムの名前が付いたサブディレクトリにあります。

注意
samples-dir の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

CustDB 統合データベースの設定

CustDB 統合データベースとして使用できるのは、SQL Anywhere、Sybase Adaptive Server Enterprise、Microsoft SQL Server、Oracle、または IBM DB2 です。

SQL Anywhere CustDB

SQL Anywhere CustDB 統合データベースは *samples-dir¥UltraLite¥CustDB¥custdb.db* にあります。SQL Anywhere 10 CustDB という DSN はインストール環境に含まれています。

データベースは、*samples-dir¥UltraLite¥CustDB¥newdb.bat* ファイルを使用して再構築できます。

CustDB サンプルの作りを詳しく調べるには、*samples-dir¥MobiLink¥CustDB¥syncsa.sql* ファイルを参照してください。

その他の RDBMS 用の CustDB

次の SQL スクリプトを使用すると、サポートされている RDBMS のいずれかで、CustDB 統合データベースを構築できます。これらのスクリプトは、*samples-dir¥MobiLink¥CustDB* にあります。

RDBMS	CustDB 設定スクリプト
Adaptive Server Enterprise	*custase.sql*
Microsoft SQL Server	*custmss.sql*
Oracle	*custora.sql*
IBM DB2	*custdb2.sql*

次の手順を実行すると、サポートされている各 RDBMS 用の CustDB 統合データベースが作成されます。

統合データベースとして使用するデータベースを準備する方法の詳細については、「統合データベースの設定」『Mobile Link - サーバ管理』を参照してください。

♦ 統合データベースを設定するには、次の手順に従います (Adaptive Server Enterprise、Oracle、または SQL Server)。

1. 使用している RDBMS でデータベースを作成します。
2. 以下の SQL スクリプトのいずれかを実行して Mobile Link システム・テーブルを追加します。これらのスクリプトは、SQL Anywhere 10 インストール環境の *MobiLink¥setup* サブディレクトリにあります。
   - Adaptive Server Enterprise 統合データベースの場合は、*syncase.sql* を実行します。
   - Oracle 統合データベースの場合は、*syncora.sql* を実行します。
   - SQL Server 統合データベースの場合は、*syncmss.sql* を実行します。
3. 以下の SQL スクリプトのいずれかを実行して CustDB データベースにサンプル・ユーザ・テーブルを追加します。これらのスクリプトは、SQL Anywhere 10 インストール環境の *Samples¥MobiLink¥CustDB* サブディレクトリにあります。
   - Adaptive Server Enterprise 統合データベースの場合は、*custase.sql* を実行します。
   - Oracle 統合データベースの場合は、*custora.sql* を実行します。
   - SQL Server 統合データベースの場合は、*custmss.sql* を実行します。
4. クライアント・マシン上で、データベースを参照する CustDB という ODBC データ・ソースを作成します。
   - [スタート] – [プログラム] – [SQL Anywhere 10] – [SQL Anywhere] – [ODBC アドミニストレータ] を選択します。
   - [追加] をクリックします。
   - リストから適切なドライバを選択します。

     [完了] をクリックします。
   - この ODBC データ・ソースに CustDB という名前を付けます。
   - [ログイン] タブをクリックします。データベースのユーザ ID とパスワードを入力します。

♦ 統合データベースを設定するには、次の手順に従います (IBM DB2)。

1. DB2 サーバ上に DB2 データベースを作成します。このチュートリアルでは、これを CsutDB と呼びます。
2. デフォルトのテーブル領域 (通常は USERSPACE1) が 8 KB ページを使用することを確認します。

デフォルトのテーブル領域が 8 KB ページを使用しない場合は、次の手順を行います。

- 1 つ以上のバッファ・プールに 8 KB ページがあることを確認します。ない場合は、8 KB ページのバッファ・プールを作成してください。
- 8 KB ページのある新しいテーブル領域とテンポラリ・テーブル領域を作成します。

  詳細については、DB2 のマニュアルを参照してください。

3. *MobiLink¥setup¥syncdb2long.sql* ファイルを使用して、Mobile Link システム・テーブルを DB2 統合データベースに追加します。

   - *syncdb2long.sql* ファイルの先頭にある接続コマンドを変更します。*DB2Database* を、お使いの DB2 データベース名 (またはそのエイリアス) に置き換えます。この例では、このデータベースを CustDB と呼びます。以下に示すように、DB2 のユーザ名とパスワードを追加することもできます。

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

   - サーバまたはクライアント・コンピュータで、DB2 コマンド・ウィンドウを開きます。次のコマンドを入力して *syncdb2long.sql* を実行します。

     ```
     db2 -c -ec -td~ +s -v -f syncdb2long.sql
     ```

4. *syncdb2long.sql* に定義されているストアド・プロシージャを DB2 で使用する場合は、SQL Anywhere インストール環境の *MobiLink¥setup* サブディレクトリにある *syncdb2long_version* という名前の Java ファイルとクラス・ファイルを、DB2 インストール環境の *FUNCTION* サブディレクトリにコピーします。
5. SQL Anywhere インストール環境の *Samples¥MobiLink¥CustDB* サブディレクトリにある *custdb2.class* を DB2 サーバ・マシンの *SQLLIB¥FUNCTION* ディレクトリにコピーします。このクラスには、CustDB サンプルで使用されるプロシージャが含まれています。
6. 以下のように、データ・テーブルを CustDB に追加します。

   - 必要に応じて、custdb2.sql の接続コマンドを変更します。たとえば、以下に示すように、ユーザ名とパスワードを追加できます。*userid* と *password* を、使用するユーザ名とパスワードに置き換えてください。

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

   - サーバまたはクライアント・コンピュータで、DB2 コマンド・ウィンドウを開きます。
   - 次のコマンドを入力して custdb2.sql を実行します。

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

   - 処理が完了したら、次のコマンドを入力してコマンド・ウィンドウを閉じます。

     ```
     exit
     ```

7. DB2 クライアント・マシン上で、DB2 データベースを参照する CustDB という ODBC データ・ソースを作成します。

- ODBC アドミニストレータを起動します。

  [スタート] – [プログラム] – [SQL Anywhere 10] – [SQL Anywhere] – [ODBC アドミニストレータ] を選択します。

  [ODBC データ ソース アドミニストレータ] が表示されます。

- [ユーザー DSN] タブで [追加] をクリックします。

  [データ ソースの新規作成] ダイアログが表示されます。

- DB2 データベース用の ODBC ドライバを選択します。たとえば、IBM DB2 UDB ODBC ドライバを選択します。[完了] をクリックします。

  DB2 の ODBC ドライバの設定方法については、次のマニュアルを参照してください。

  - DB2 のマニュアル
  - http://www.ianywhere.jp/developers/technotes/odbc_mobilink.html

8. 以下のようにして、DB2 クライアント・マシン上で *custdb2setuplong* Java アプリケーションを実行します。このアプリケーションは、DB2 データベースの CustDB サンプルをリセットします。初期設定が終了したら、同じコマンド・ラインを入力してこのアプリケーションを実行し、DB2 CustDB データベースをいつでもリセットできます。

   - データ・ソースに CustDB 以外の名前を使用する場合は、以下のように入力して *custdb2setuplong.java* の接続コードを変更し、再コンパイルする必要があります。システム変数 *%db2tempdir%* で指定するパスにスペースが含まれている場合は、パスを引用符で囲んでください。

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

   - 次のように入力します。ここでは、*userid* と *password* は CustDB ODBC データ・ソースに接続するためのユーザ名とパスワードです。

     ```
     java custdb2setuplong userid password
     ```

参照

- 「IBM DB2 UDB 統合データベース」 『Mobile Link - サーバ管理』

Ultra Light リモート・データベースの設定

以下の手順では、CustDB のリモート・データベースを作成します。CustDB リモート・データベースは、Ultra Light データベースでなければなりません。

リモート・データベースのアプリケーション論理は *samples-dir¥UltraLite¥CustDB* にあります。これには、以下のファイルが含まれています。

- **Embedded SQL の論理**　ファイル *custdb.sqc* には、Ultra Light データベースの情報を問い合わせて修正するための SQL 文と、統合データベースとの同期を開始するために必要な呼び出しが格納されています。
- **C++ API の論理**　ファイル *custdbcomp.cpp* には、C++ API の論理が格納されています。
- **ユーザ・インタフェース機能**　この機能は、*Samples¥UltraLite¥CustDB* のプラットフォーム固有のサブディレクトリに、別々に格納されています。

Ultra Light が実行されているリモート・デバイスにサンプル・アプリケーションをインストールするには、次の手順を実行します。

♦ サンプル・アプリケーションをリモート・デバイスにインストールするには、次の手順に従います。

1. 統合データベースを起動します。
2. Mobile Link サーバを起動します。
3. サンプル・アプリケーションをリモート・デバイスにインストールします。
4. リモート・デバイスでサンプル・アプリケーションを起動します。
5. サンプル・アプリケーションを同期します。

例

次の例では、DB2 統合データベースを対象として動作している Palm デバイスに CustDB サンプルをインストールします。

1. 次のようにして、統合データベースが稼働していることを確認します。

   DB2 コマンド・ウィンドウを開きます。次のコマンドを入力します。ここでは、*userid* と *password* は DB2 データベースに接続するためのユーザ ID とパスワードです。

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

2. Mobile Link サーバを起動します。

   DB2 データベースと同期できるようにするため、コマンド・プロンプトで次のコマンドを実行します。

   ```
   mlsrv10 -c "DSN=CustDB" -zp
   ```

3. 次のようにして、サンプル・アプリケーションを Palm デバイスにインストールします。
   - 使用中の PC で、Palm Desktop を起動します。
   - [Palm Desktop] ツールバーの [インストール] をクリックします。
   - [追加] をクリックし、SQL Anywhere 10 インストール環境の *UltraLite¥palm¥68k* サブディレクトリにある *custdb.prc* を検索します。
   - [開く] をクリックします。

- Palm デバイスで HotSync を実行します。

4. 次のようにして、Palm デバイスで CustDB サンプル・アプリケーションを起動します。
   - Palm デバイスをクレードルに置きます。

     サンプル・アプリケーションを初めて起動した場合は、データの初期コピーの同期とダウンロードを実行することを求めるメッセージが表示されます。この手順が必要なのは、アプリケーションを初めて起動する場合だけです。これによってダウンロードされたデータは、Ultra Light データベースに格納されます。
   - サンプル・アプリケーションを起動します。

     [Applications] ビューで、[CustDB] を選択します。

     初期ダイアログが表示され、従業員 ID を入力するプロンプトが表示されます。
   - 従業員 ID を入力します。

     チュートリアルとして実行するときは、値 50 を入力してください。このサンプル・アプリケーションでは、51、52、または 53 の値も使用できますが、これらの値を入力した場合は、動作が多少異なります。

     各ユーザ ID を使用したときの動作の詳細については、「CustDB サンプル内のユーザ」 63 ページを参照してください。

     次の処理に進む前に同期を行う必要があることを示すメッセージ・ボックスが表示されます。
   - アプリケーションを同期します。

     HotSync を使用して、データの初期コピーを取得します。
   - データが同期されたことを確認します。

     [Applications] ビューで、[CustDB] アプリケーションを選択します。顧客用の入力シートにデータが入力されて画面に表示されます。
5. リモート・アプリケーションを統合データベースと同期します。データベースを変更したときは、この手順のみを行ってください。
   - 統合データベースと Mobile Link サーバが動作中であることを確認します。
   - Palm デバイスをクレードルに置きます。
   - HotSync ボタンを押して同期を行います。

CustDB データベース内のテーブル

CustDB データベースのテーブル定義は、*samples-dir¥MobiLink¥CustDB* にあるプラットフォーム固有のファイル内に格納されています。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

CustDB テーブルの ER (実体関連) 図については、「CustDB サンプル・データベースについて」『SQL Anywhere 10 - 紹介』を参照してください。

次の 5 つのテーブルは統合データベースとリモート・データベースの両方にありますが、その定義は両者で少し異なります。

ULCustomer

ULCustomer テーブルには、顧客リストがあります。

リモート・データベースの ULCustomer には、次のカラムがあります。

- **cust_id**　顧客を識別するユニークな整数を保持するプライマリ・キー・カラム。
- **cust_name**　顧客の名前を保持する 30 バイトの文字列。

統合データベースの ULCustomer には、次のカラムもあります。

- **last_modified**　ローが最後に変更されたときのタイムスタンプ。このカラムは、タイムスタンプベースの同期に使用されます。

ULProduct

ULProduct テーブルには、製品リストがあります。

リモート・データベースと統合データベースの両方の ULProduct に、次のカラムがあります。

- **prod_id**　製品を識別するユニークな整数を保持するプライマリ・キー・カラム。
- **price**　単価を識別する整数。
- **prod_name**　製品の名前を保持する 30 バイトの文字列。

ULOrder

ULOrder テーブルには受注リストがあります。注文した顧客の情報、受注した従業員、注文された製品についての詳細が含まれています。

リモート・データベースの ULOrder には、次のカラムがあります。

- **order_id**　注文を識別するユニークな整数を保持するプライマリ・キー・カラム。
- **cust_id**　ULCustomer を参照する外部キー・カラム。
- **prod_id**　ULProduct を参照する外部キー・カラム。
- **emp_id**　ULEmployee を参照する外部キー・カラム。

- **disc** 注文に適用される値引きを保持する整数。
- **quant** 注文された製品の数を保持する整数。
- **notes** 注文に関する注記を保持する 50 バイトの文字列。
- **status** 注文のステータスが記述された 20 バイトの文字列。

統合データベースの ULOrder には、次のカラムもあります。

- **last_modified** ローが最後に変更されたときのタイムスタンプ。このカラムは、タイムスタンプベースの同期に使用されます。

ULOrderIDPool

ULOrderIDPool テーブルは、ULOrder のプライマリ・キー・プールです。

リモート・データベースの ULOrderIDPool には、次のカラムがあります。

- **pool_order_id** 注文 ID を識別するユニークな整数を保持するプライマリ・キー・カラム。

統合データベースの ULOrderIDPool には、次のカラムもあります。

- **pool_emp_id** 注文 ID が割り当てられたリモート・データベースの所有者の従業員 ID を保持する整数カラム。
- **last_modified** ローが最後に変更されたときのタイムスタンプ。

ULCustomerIDPool

ULCustomerIDPool テーブルは、ULCustomer のプライマリ・キー・プールです。

リモート・データベースの ULCustomerIDPool には、次のカラムがあります。

- **pool_cust_id** 顧客 ID を識別するユニークな整数を保持するプライマリ・キー・カラム。

統合データベースの ULCustomerIDPool には、次のカラムもあります。

- **pool_emp_id** リモート・データベースで生成された新しい従業員に使用される従業員 ID を保持する整数カラム。
- **last_modified** ローが最後に変更されたときのタイムスタンプ。

以下のテーブルは、統合データベースにのみ存在します。

ULIdentifyEmployee_nosync

ULIdentifyEmployee_nosync テーブルは、統合データベースにのみ存在します。これには、次の 1 つのカラムがあります。

- **emp_id** このプライマリ・キー・カラムには、従業員 ID を示す整数が保持されています。

ULEmployee

ULEmployee テーブルは、統合データベースにのみ存在します。これには、営業担当者リストが格納されています。

ULEmployee には、次のカラムがあります。

- **emp_id**　従業員を識別するユニークな整数を保持するプライマリ・キー・カラム。
- **emp_name**　従業員の名前を保持する 30 バイトの文字列。

ULEmpCust

ULEmpCust テーブルは、どの顧客の注文をダウンロードするかを制御します。従業員が新しい顧客の注文を必要とする場合は、従業員 ID と顧客 ID を挿入すると、その顧客の注文がダウンロードされます。

- **emp_id**　ULEmployee.emp_id の外部キー。
- **cust_id**　ULCustomer.cust_id の外部キー。プライマリ・キーは、emp_id と cust_id で構成されています。
- **action**　従業員のレコードをリモート・データベースから削除するかどうかを決定するのに使用される文字。従業員が顧客の注文を必要としなくなった場合は、D (削除) に設定します。注文が必要な場合、action は NULL に設定してください。

  この場合は、ULOrder テーブルから削除するローを統合データベースが識別できるようにするため、論理削除を使用する必要があります。削除情報がダウンロードされると、action が D に設定されたその従業員のすべてのレコードは統合データベースからも削除できます。
- **last_modified**　ローが最後に変更されたときのタイムスタンプ。このカラムは、タイムスタンプベースの同期に使用されます。

ULOldOrder と ULNewOrder

これらのテーブルは、統合データベースにのみ存在します。また、競合を解決するために使用され、ULOrder と同じカラムが含まれています。SQL Anywhere と Microsoft SQL Server では、これらはテンポラリ・テーブルです。Adaptive Server Enterprise の場合、これらのテーブルは通常のテーブルであり、@@spid です。DB2 と Oracle にはテンポラリ・テーブルがないため、同期ユーザに属するローを Mobile Link が識別できるようになっている必要があります。これらのテーブルはベース・テーブルであるため、5 人のユーザが同期している場合は、これらのテーブルのローを各ユーザが同時に保持することがあります。

@@spid の詳細については、「変数」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

CustDB サンプル内のユーザ

CustDB サンプルのユーザには、営業担当者とモバイル管理者の 2 つのタイプがあります。相違点は次のとおりです。

- **営業担当者**　営業担当者に関連付けられたリモート・データベースは、ユーザ ID 51、52、53 で識別されます。営業担当者は、次のタスクを実行できます。
  - 顧客と製品のリスト表示
  - 新規顧客の追加
  - 注文の追加や削除
  - 未処理の注文リストのスクロール
  - 変更内容に関する統合データベースとの同期
- **モバイル管理者**　モバイル管理者に関連付けられたリモート・データベースは、ユーザ ID 50 で識別されます。モバイル管理者は、営業担当者と同じタスクを実行できます。 このほか、モバイル管理者は次のタスクを実行できます。
  - 注文の承認や拒否

CustDB の同期

以下の項では、CustDB サンプルの同期論理を説明します。

同期論理のソース・コード

Sybase Central を使用すると、統合データベース内の同期スクリプトを調べることができます。

スクリプト・タイプとイベント

custdb.sql ファイルは、ml_add_connection_script または ml_add_table_script を呼び出して、各同期スクリプトを統合データベースに追加します。

例

custdb.sql の次の行は、ULProduct テーブル用のテーブル・レベルのスクリプトを追加します。このスクリプトは、download_cursor イベントで実行されます。スクリプトには、SELECT 文が 1 つあります。

```
call ml_add_table_script(
'CustDB 10.0',
'ULProduct', 'download_cursor',
'SELECT prod_id, price, prod_name FROM ULProduct' )
go
```

CustDB サンプルでの注文の同期

ビジネス・ルール

ULOrder テーブルのビジネス・ルールは、次のとおりです。

- 注文は、承認されていないか、ステータスが NULL である場合にかぎりダウンロードされる。
- 注文は、統合データベースでもリモート・データベースでも修正できる。
- 各リモート・データベースには、従業員に対応する注文のみが保持される。

ダウンロード

統合データベースでは、注文を挿入、削除、更新できます。これらの操作に対応するスクリプトは、次のとおりです。

- **download_cursor** download_cursor スクリプトの最初のパラメータは、最終ダウンロード・タイムスタンプです。これは、最後の同期以後にリモート・データベースまたは統合データベースのいずれかで修正されたローのみをダウンロードするために使用されます。2 番目のパラメータは従業員 ID です。この ID は、ダウンロードするローを判断するために使用されます。

  CustDB の download_cursor スクリプトを次に示します。

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

CustDB の ULOrderDownload プロシージャを次に示します。

```
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 の download_delete_cursor スクリプトを次に示します。

```
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}
```

アップロード

リモート・データベースでは、注文を挿入、削除、更新できます。これらの操作に対応するスクリプトは、次のとおりです。

- **upload_insert**　CustDB の upload_insert スクリプトを次に示します。

```
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 の upload_update スクリプトを次に示します。

```
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 の upload_delete スクリプトを次に示します。

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

- **upload_fetch**　CustDB の upload_fetch スクリプトを次に示します。

```
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 の upload_old_row_insert スクリプトを次に示します。

```
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 の upload_new_row_insert スクリプトを次に示します。

```
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 } )
```

競合解決

- **resolve_conflict**　CustDB の resolve_conflict スクリプトを次に示します。

```
CALL ULResolveOrderConflict
```

CustDB の ULResolveOrderConflict プロシージャを次に示します。

```
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
```

CustDB サンプルの顧客の同期

ビジネス・ルール

顧客を規定するビジネス・ルールは、次のとおりです。

- 顧客情報は、統合データベースでもリモート・データベースでも修正できる。
- リモート・データベースと統合データベースの両方に、完全な顧客リストがある。

ダウンロード

統合データベースでは、顧客情報を挿入または更新できます。これらの操作に対応するスクリプトは、次のとおりです。

- **download_cursor**　次の download_cursor スクリプトは、ユーザが最後に情報をダウンロードした後で情報が変更された顧客をすべてダウンロードします。

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

アップロード

リモート・データベース側で顧客情報を挿入、更新、または削除できます。これらの操作に対応するスクリプトは、次のとおりです。

- **upload_insert**　CustDB の upload_insert スクリプトを次に示します。

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

- **upload_update**　CustDB の upload_update スクリプトを次に示します。

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

  このテーブルに対する競合検出は実行されません。

- **upload_delete**　CustDB の upload_delete スクリプトを次に示します。

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

CustDB サンプルの製品の同期

ビジネス・ルール

ULProduct に関するすべてのローがダウンロードされます。これはスナップショット同期と呼ばれます。

「スナップショットを使った同期」『Mobile Link - サーバ管理』を参照してください。

ULProduct テーブルのビジネス・ルールは、次のとおりです。

- 統合データベースでは、製品の修正のみが可能。
- 各リモート・データベースには、すべての製品が含まれている。

ダウンロード

統合データベースでは、製品情報を挿入、削除、更新できます。これらの操作に対応するスクリプトは、次のとおりです。

- **download_cursor**　次の download_cursor スクリプトは、同期が行われるたびに ULProduct テーブルのすべてのローとカラムをダウンロードします。

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


顧客と注文のプライマリ・キー・プールの管理

CustDB サンプル・データベースでは、ULCustomer テーブルと ULOrder テーブルのユニークなプライマリ・キーを管理するために、プライマリ・キー・プールが使用されます。プライマリ・キー・プールは、ULCustomerIDPool テーブルと ULOrderIDPool テーブルです。

ULCustomerIDPool

以下のスクリプトは、ULCustomerIDPool テーブルで定義されています。

ダウンロード

- **download_cursor** CustDB の download_cursor スクリプトを次に示します。

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

アップロード

- **upload_insert** CustDB の upload_insert スクリプトを次に示します。

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

- **upload_delete** CustDB の upload_delete スクリプトを次に示します。

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

- **end_upload** 次の end_upload スクリプトは、各アップロードの完了後に 20 個の顧客 ID が顧客 ID プールに残るように処理を行います。

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

CustDB の UL_CustomerIDPool_maintain プロシージャを次に示します。

```
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

以下のスクリプトは、ULOrderIDPool テーブルで定義されています。

ダウンロード

- **download_cursor**　CustDB の download_cursor スクリプトを次に示します。

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

アップロード

- **end_upload**　次の end_upload スクリプトは、各アップロードの完了後に 20 個の注文 ID が注文 ID プールに残るように処理を行います。

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

CustDB の UL_OrderIDPool_maintain プロシージャを次に示します。

```
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 の upload_insert スクリプトを次に示します。

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

- **upload_delete**　CustDB の upload_delete スクリプトを次に示します。

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

クリーンアップ

サンプルを再起動するには、CustDB データベースのデータをリセットします。

♦ **CustDB データベースのデータをリセットするには、次の手順に従います。**

1. 次のようにして、ULDBUtil をデバイスにインストールします。
   - Palm デバイス用に、PC で Palm Desktop を起動します。
   - [Palm Desktop] ツールバーの [インストール] をクリックします。
   - [追加] をクリックし、SQL Anywhere 10 インストール環境の *UltraLite¥palm¥68k* サブディレクトリにある *uldbutil.prc* を検索します。
   - [完了] をクリックします。
   - Palm デバイスで HotSync を実行します。
2. 次のように、ULDBUtil を使用してデータを削除します。
   - Palm デバイスで [ULDBUtil] アイコンを選択します。
   - CustDB を選択し、[データを削除します] を選択します。
   - Palm デバイスで HotSync を実行します。

詳細情報

スクリプトの種類の詳細については、「スクリプトの種類」 『Mobile Link - サーバ管理』を参照してください。

各スクリプトやそのパラメータなどのリファレンス情報については、「同期イベント」 『Mobile Link - サーバ管理』を参照してください。

第 4 章

Mobile Link Contact サンプルの解説

目次

Contact サンプル・チュートリアルの概要

Contact サンプルは、Mobile Link 開発者にとって貴重なリソースです。このサンプルを使用して、Mobile Link アプリケーションの開発時に必要なさまざまな方法の実装例を紹介します。

Contact サンプル・アプリケーションは、1 つの SQL Anywhere 統合データベースと、2 つの SQL Anywhere リモート・データベースから構成されています。このサンプルでは、同期の一般的な方法をいくつか説明します。この章で説明することを最大限に活用するために、サンプル・アプリケーションのソース・コードを読んで理解してください。

統合データベースは SQL Anywhere データベースですが、同期スクリプトは他のデータベース管理システムの最小限の変更を処理する SQL 文で構成されています。

Contact サンプルは *samples-dir¥MobiLink¥Contact* にあります。概要については、同じディレクトリにある readme ファイルを参照してください。*samples-dir* の詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

同期の設計

Contact サンプル・アプリケーションの同期の設計では、以下の機能を使用します。

- **カラムのサブセット**　統合データベース上の Customer、Product、SalesRep、Contact の各テーブルのカラムのサブセットが、リモート・データベースで共有される。
- **ローのサブセット**　統合データベース上の SalesRep テーブルの単一のローのすべてのカラムが各リモート・データベースで共有される。

  「リモート・データベース間でローを分割する」『Mobile Link - サーバ管理』を参照してください。
- **タイムスタンプベースの同期**　最後のデバイス同期以降に実行された統合データベースに対する変更を識別する方法。Customer、Contact、Product の各テーブルは、タイムスタンプベースの方法を使用して同期される。

  「タイムスタンプベースのダウンロード」『Mobile Link - サーバ管理』を参照してください。

Contact サンプルの設定

Contact サンプル・データベースを構築できるように、Windows バッチ・ファイル *build.bat* が用意されています。UNIX システムでは *build.sh* です。このバッチ・ファイルの内容を調べることができます。このバッチ・ファイルによって次のアクションが実行されます。

- 統合データベースと 2 つのリモート・データベース用に、ODBC データ・ソース定義が作成されます。
- *consol.db* という統合データベースが作成され、Mobile Link システム・テーブル、データベース・スキーマ、データ、同期スクリプト、Mobile Link ユーザ名がデータベースにロードされます。
- *remote.db* という名前の 2 つのリモート・データベースが、サブディレクトリ *remote_1* と *remote_2* に作成されます。両方のデータベースに共通の情報がロードされ、カスタマイズ内容が適用されます。カスタマイズ内容には、グローバル・データベース識別子、Mobile Link ユーザ名、2 つのパブリケーションに対するサブスクリプションが含まれます。

♦ Contact サンプルを構築するには、次の手順に従います。

1. コマンド・プロンプトで、*samples-dir¥MobiLink¥Contact* に移動します。
2. *build.bat* (Windows) または *build.sh* (UNIX) を実行します。

samples-dir の詳細については、「サンプル・ディレクトリ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

Contact サンプルの実行

Contact サンプルには最初の同期を実行するバッチ・ファイルが含まれており、Mobile Link サーバと dbmlsync のコマンド・ラインが例示されています。*samples-dir¥MobiLink¥Contact* に次のバッチ・ファイルがあり、その内容はテキスト・エディタで確認できます。

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

♦ Contact サンプルを実行するには、次の手順に従います。

1. Mobile Link サーバを起動します。
   - コマンド・プロンプトで、*samples-dir¥MobiLink¥Contact* に移動し、次のコマンドを実行します。

     ```
     step1
     ```


このコマンドは、Mobile Link サーバを冗長モードで起動するバッチ・ファイルを実行します。このモードは、開発中やトラブルシューティング中には便利ですが、パフォーマンスが低下するため、通常の運用環境では使用しません。

2. 両方のリモート・データベースを同期します。

   - コマンド・プロンプトで、*samples-dir¥MobiLink¥Contact* に移動します。
   - 次のコマンドを実行します。

     ```
     step2
     ```

     これは、両方のリモート・データベースを同期するバッチ・ファイルです。

3. Mobile Link サーバを停止します。

   - コマンド・プロンプトで、*samples-dir¥MobiLink¥Contact* に移動します。
   - 次のコマンドを実行します。

     ```
     step3
     ```

     これは、Mobile Link サーバを停止させるバッチ・ファイルです。

Contact サンプルで同期の動作方法を調べるには、Interactive SQL を使用してリモート・データベースと統合データベースでデータを修正し、バッチ・ファイルを使用して同期を行います。

Contact データベース内のテーブル

Contact データベースのテーブル定義は、次のファイルにあります。これらのファイルはすべてサンプルのディレクトリにあります。

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

次の 3 つのテーブルは統合データベースとリモート・データベースの両方にありますが、その定義は両者で少し異なります。

SalesRep

SalesRep テーブルには、営業担当者ごとに 1 つのローがあります。リモート・データベースは、それぞれ 1 人の営業担当者が所持します。

各リモート・データベースの SalesRep には、次のカラムがあります。

- **rep_id**　営業担当者の識別番号が格納されるプライマリ・キー・カラム。
- **name**　担当者の名前。

統合データベース側には、この他に営業担当者の Mobile Link ユーザ名を保持する ml_username カラムもあります。

Customer

このテーブルには、顧客ごとに 1 つのローがあります。顧客は、それぞれ 1 人の営業担当者が担当する会社です。SalesRep テーブルと Customer テーブルは 1 対多の関係になっています。

各リモート・データベースの Customer には、次のカラムがあります。

- **cust_id**　顧客の識別番号を保持するプライマリ・キー・カラム。
- **name**　顧客名。これは会社名です。
- **rep_id**　SalesRep テーブルを参照する外部キー・カラム。顧客に割り当てられた営業担当者を識別します。

統合データベースには、この他に last_modified カラムと active カラムがあります。

- **last_modified**　ローを最後に変更した時刻。このカラムは、タイムスタンプベースの同期に使用されます。
- **active**　顧客が現在アクティブであるか (1)、またはこの顧客との取引がなくなったか (0) を示すビット・カラム。このカラムに非アクティブ (0) のマークが付いている場合は、この顧客に対応するすべてのローがリモート・データベースから削除されます。

Contact

このテーブルには、窓口ごとに 1 つのローがあります。窓口担当者は、顧客の会社の従業員です。Customer テーブルと Contact テーブルは 1 対多の関係になっています。

各リモート・データベースの Contact には、次のカラムがあります。

- **contact_id**　窓口担当者の識別番号を保持するプライマリ・キー・カラム。
- **name**　各窓口担当者の氏名。
- **cust_id**　窓口担当者が所属する顧客の識別子。

統合データベースでは、このテーブルに次のカラムもあります。

- **last_modified**　ローを最後に変更した時刻。このカラムは、タイムスタンプベースの同期に使用されます。
- **active**　窓口が現在アクティブであるか (1)、またはこの窓口との取引がなくなったか (0) を示すビット・カラム。このカラムに非アクティブ (0) のマークが付いている場合は、この窓口に対応するローがリモート・データベースから削除されます。

Product

Product テーブルには、会社で販売される製品ごとに 1 つのローがあります。Product テーブルは別のパブリケーションに保持されるため、リモート・データベースはこのテーブルを別途同期できます。

各リモート・データベースの Product には、次のカラムがあります。

- **id**　製品を識別するユニークな数値を保持するプライマリ・キー・カラム。
- **name**　品目の名前。
- **size**　品目のサイズ。
- **quantity**　品目の在庫数量。営業担当者が注文を受け取った時点で、このカラムは更新されます。
- **unit_price**　製品の単価。

統合データベースの Product テーブルには、次のカラムもあります。

- **supplier**　製品を製造している会社。
- **last_modified**　ローを最後に変更した時刻。このカラムは、タイムスタンプベースの同期に使用されます。
- **active**　製品が現在アクティブ (1) であるかどうかを示すビット・カラム。このカラムに非アクティブ (0) のマークが付いている場合は、この製品に対応するローがリモート・データベースから削除されます。

統合データベースには、これらのテーブルに加えてテーブル・セットが作成されます。これには、競合解決中に使用されるテンポラリ・テーブル product_conflict と、ユーザ mlmaint が所有する Mobile Link アクティビティをモニタリングするためのテーブル・セットが含まれます。Mobile Link モニタリング・テーブルを作成するためのスクリプトは、*samples-dir¥MobiLink¥Contact ¥mlmaint.sql* ファイルにあります。

Contact サンプル内のユーザ

Contact サンプルには、複数のデータベース・ユーザ ID と Mobile Link ユーザ名が含まれています。

データベース・ユーザ ID

2 つのリモート・データベースは、営業担当者 Samuel Singer (rep_id 856) と Pamela Savarino (rep_id 949) が所持しています。

この 2 人のユーザはどちらも、それぞれのリモート・データベースへの接続時に、デフォルトの SQL Anywhere ユーザ ID **dba** とパスワード **SQL** を使用します。

また、各リモート・データベースには、ユーザ ID **sync_user** (パスワードは **sync_user**) もあります。このユーザ ID は、dbmlsync コマンド・ラインでのみ使用します。これは REMOTE DBA 権限を持つユーザなので、dbmlsync からの接続時にはあらゆる操作を実行できますが、他のアプリケーションからの接続時には何の権限もありません。したがって、このユーザ ID とパスワードは広範囲で使用できますが問題にはなりません。

統合データベースには、**mlmaint** というユーザが存在します。このユーザは Mobile Link 同期統計とエラーのモニタリングに使用されるテーブルの所有者です。このユーザは接続権限を持っていません。テーブルを個々のユーザ ID に割り当てるには、スキーマ内でオブジェクトを他のオブジェクトから分離するだけであり、Sybase Central や他のユーティリティで管理しやすくなっています。

Mobile Link ユーザ名

Mobile Link ユーザ名は、データベース・ユーザ ID とは異なります。各リモート・デバイスには、データベースへの接続時に使用するユーザ ID の他に、Mobile Link ユーザ名があります。Samuel Singer の Mobile Link ユーザ名は SSinger です。Pamela Savarino の Mobile Link ユーザ名は PSavarino です。Mobile Link ユーザ名は、次のロケーションで格納または使用されています。

- リモート・データベース。Mobile Link ユーザ名が、CREATE SYNCHRONIZATION USER 文を使用して追加されています。
- 統合データベース。Mobile Link ユーザ名とパスワードが、mluser ユーティリティを使用して追加されています。
- *MobiLink¥Contact¥step2.bat* 内の dbmlsync コマンド・ライン。同期時に、接続ユーザの Mobile Link パスワードが指定されます。
- Mobile Link サーバ。同期時、Mobile Link ユーザ名がパラメータとして多数のスクリプトに指定されます。
- 統合データベース側の SalesRep テーブル。ml_username カラムがあります。同期スクリプトは、このカラムの値と Mobile Link ユーザ名パラメータを比較します。

Contact サンプルの同期

以下の項では、Contact サンプルの同期論理を説明します。

Contact サンプルの営業担当者の同期

SalesRep テーブルの同期スクリプトは、**スナップショットを使った同期**の例を示しています。営業担当者の情報は、変更されたかどうかに関係なくダウンロードされます。

「スナップショットを使った同期」『Mobile Link - サーバ管理』を参照してください。

ビジネス・ルール

SalesRep テーブルのビジネス・ルールは、次のとおりです。

- リモート・データベース側ではテーブルを修正しない。
- 営業担当者の Mobile Link ユーザ名と rep_id 値を変更しない。
- 各リモート・データベースの SalesRep テーブルには、リモート・データベース所有者の Mobile Link ユーザ名に対応するローが 1 つだけ存在する。

ダウンロード

- **download_cursor**　各リモート・データベースの SalesRep テーブルには、ローが 1 つだけ存在します。単一のローのダウンロードに伴うオーバヘッドはほとんどないため、単純なスナップショットの download_cursor スクリプトを使用します。

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

  このスクリプトの最初のパラメータは、最終ダウンロード・タイムスタンプですが、これは使用されません。IS NOT NULL は、パラメータを使用するために指定されたダミー式です。2 番目のパラメータは Mobile Link ユーザ名です。

アップロード

このテーブルはリモート・テーブル側では更新しないため、アップロード・スクリプトはありません。

Contact サンプルの顧客の同期

Customer テーブルの同期スクリプトは、**タイムスタンプベースの同期**とローの分割の例を示しています。これらの方法では、同期中に転送されるデータの量が最小限になり、テーブル・データの整合性が保持されます。

次の項を参照してください。

- 「タイムスタンプベースのダウンロード」『Mobile Link - サーバ管理』

- 「リモート・データベース間でローを分割する」 『Mobile Link - サーバ管理』

ビジネス・ルール

顧客を規定するビジネス・ルールは、次のとおりです。

- 顧客情報は、統合データベースでもリモート・データベースでも修正できる。
- 営業担当者間で、顧客の再割り当てを定期的に変更できる。このプロセスは、一般に領域の再編成と呼ばれる。
- 各リモート・データベースには、割り当てられている顧客のみが保持される。

ダウンロード

- **download_cursor** 次の download_cursor スクリプトは、最後の正常なダウンロード以後に情報が変更されたアクティブな顧客のみをダウンロードします。また、営業担当者別に顧客をフィルタリングします。

  ```
  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** 次の download_delete_cursor スクリプトは、最後の正常なダウンロード以後に情報が変更された顧客のみをダウンロードします。また、非アクティブのマークが付いているか、指定された営業担当者に割り当てられていない顧客を、すべて削除します。

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

  統合データベースにある Customer テーブルからローが削除されると、この結果セットには表示されないため、リモート・データベースからは削除されません。代わりに、顧客には非アクティブのマークが付きます。

  領域が再編成されると、このスクリプトは営業担当者への割り当てから外れた顧客を削除します。また、他の営業担当者に移された顧客も削除します。このような追加の削除にはフラグとして SQLCODE 100 が設定されますが、同期の妨げにはなりません。より複雑なスクリプトを作成すれば、現在の営業担当者から外された顧客のみを識別できます。

  Mobile Link クライアントはリモート・データベースでカスケード削除を実行するため、このスクリプトによって、他の営業担当者に割り当てられた顧客のすべての窓口が削除されます。

アップロード

リモート・データベース側で顧客情報を挿入、更新、または削除できます。これらの操作に対応するスクリプトは、次のとおりです。

- **upload_insert** 次の upload_insert スクリプトは、Customer テーブルにローを 1 つ追加して、顧客にアクティブのマークを付けます。


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

- **upload_update**　次の upload_update スクリプトは、統合データベースにある顧客情報を修正します。このテーブルに対する競合検出は実行されません。

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

- **upload_delete**　次の upload_delete スクリプトは、統合データベースで顧客に非アクティブのマークを付けます。ローは削除されません。

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

Contact サンプルの顧客窓口の同期

Contact テーブルには、顧客の会社の社員名、顧客を参照するための外部キー、窓口を識別するユニークな整数が含まれています。また、last_modified タイムスタンプと、窓口がアクティブであるかどうかを示すマーカもあります。

ビジネス・ルール

このテーブルのビジネス・ルールは、次のとおりです。

- 窓口情報は、統合データベースでもリモート・データベースでも修正できる。
- 各リモート・データベースには、営業担当者が割り当てられている顧客の窓口のみが含まれる。
- 営業担当者間で顧客を再割り当てした場合は、窓口の再割り当てもする。

トリガ

Customer テーブルのトリガは、顧客情報に変更があったときに窓口が確実に選択されるようにするために使用されます。トリガは、各窓口の last_modified カラムを、その窓口に対応する顧客の情報が変更されるたびに明示的に変更します。

```
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
```

顧客が修正されるたびにすべての窓口レコードを更新することで、トリガは顧客とその関連窓口を結合します。そのため、顧客が修正されるとすべての関連窓口も修正され、次回の同期時に顧客とその関連窓口が一括してダウンロードされます。

ダウンロード

- **download_cursor**　Contact の download_cursor スクリプトを次に示します。

```
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
```

  このスクリプトは、アクティブな窓口、営業担当者が最後にダウンロードした後に (明示的に、または対応する顧客の修正によって) 変更された窓口、営業担当者に割り当てられている窓口をすべて取り出します。この営業担当者に関連付けられている窓口を識別するには、Customer テーブルと SalesRep テーブルのジョインが必要です。

- **download_delete_cursor**　Contact の download_delete_cursor スクリプトを次に示します。

```
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
```

  リモート・データベースから顧客が削除されると、Mobile Link クライアントでは、カスケード参照整合性が自動的に使用され、対応する窓口が削除されます。このため、download_delete_cursor スクリプトは、非アクティブのマークが付いている窓口のみを削除します。

アップロード

リモート・データベース側で窓口情報を挿入、更新、または削除できます。これらの操作に対応するスクリプトは、次のとおりです。

- **upload_insert**　次の upload_insert スクリプトは、Contact テーブルにローを 1 つ追加して、窓口にアクティブのマークを付けます。

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

- **upload_update**　次の upload_update スクリプトは、統合データベースにある窓口情報を修正します。

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

  このテーブルに対する競合検出は実行されません。

- **upload_delete**　次の upload_delete スクリプトは、統合データベースで窓口に非アクティブのマークを付けます。ローは削除されません。


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

Contact サンプルの製品の同期

Product テーブル用のスクリプトは、競合の検出と解決の例を示しています。

Product テーブルは他のテーブルとは別のパブリケーションに保管されているため、別個にダウンロードできます。たとえば、価格変更と営業担当者が低速リンク経由で同期している場合は、それぞれ顧客と窓口の変更をアップロードしなくても、製品変更をダウンロードできます。

ビジネス・ルール

リモート・データベース側で可能な変更は、注文を受けた時点で quantity カラムの値を変更することだけです。

ダウンロード

- **download_cursor** 次の download_cursor スクリプトは、最後のリモート・データベースの同期以後に変更されたすべてのローをダウンロードします。

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

- **download_delete_cursor** 次の download_delete_cursor スクリプトは、会社が販売を中止した製品をすべて削除します。これらの製品には、統合データベース内で非アクティブのマークが付きます。

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

アップロード

リモート・データベースからは UPDATE 操作のみがアップロードされます。これらのアップロード・スクリプトの主な機能は、競合の検出と解決のためのプロシージャです。

2 人の営業担当者が注文を受けて同期を行うと、それぞれの注文が Product テーブルの quantity カラムから減算されます。たとえば、Samuel Singer が野球帽 (製品 ID 400) 20 個の注文を受けると、数量は 90 から 70 に変化します。Pamela Savarino が Samuel Singer の変更を受け取る前に野球帽 10 個の注文を受けると、自分のデータベース内のカラムの値が 90 から 80 に変化します。

Samuel Singer が自分の変更を同期すると、統合データベース内の quantity カラムは 90 から 70 に変更されます。Pamela Savarino が自分の変更を同期した場合の正しい値は 60 です。この設定は、競合を検出することで行われます。

競合検出スキーマには、次のスクリプトが含まれています。

- **upload_update** 次の upload_update スクリプトは、統合データベース側で単純な UPDATE を実行します。

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

- **upload_fetch**　次の upload_fetch スクリプトは、Product テーブルから単一のローをフェッチして、アップロードされるローの古い値と比較します。2 つのローが異なる場合は、競合が検出されます。

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

- **upload_old_row_insert**　競合が検出されると、古い値が product_conflict テーブルに挿入されます。これは resolve_conflict スクリプトによって使用されます。row_type カラムに、Old を表す値 O を持つローが追加されます。

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

- **upload_new_row_insert**　次のスクリプトは、アップロードされるローの新しい値を product_conflict テーブルに追加します。これは、resolve_conflict スクリプトによって使用されます。

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

競合解決

- **resolve_conflict**　次のスクリプトは、統合データベース内の数量値に新しい値と古い値の差を加算して、競合を解決します。

```
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'
```

Contact サンプルの統計とエラーのモニタリング

Contact サンプルには、単純なエラー・レポート・スクリプトとモニタリング・スクリプトがいくつか用意されています。これらのスクリプトを作成するための SQL 文は、*MobiLink¥Contact ¥mlmaint.sql* ファイルにあります。

各スクリプトは、値を保持するように作成されたテーブルにローを挿入します。便宜上、各テーブルの所有者は個別ユーザ mlmaint となっています。

第 5 章

チュートリアル：Oracle 10g 統合データベースでの Mobile Link の使用

目次

Mobile Link Oracle チュートリアルの概要

このチュートリアルでは、Oracle 統合データベースと SQL Anywhere リモート・データベースを準備します。

必要なソフトウェア

- SQL Anywhere の完全なインストール環境
- Oracle Enterprise Edition 10g の完全なインストール環境
- iAnywhere Solutions - Oracle ドライバ

前提知識と経験

このチュートリアルの前提となる知識と経験は、次のとおりです。

- Sybase Central インタフェースと機能の使用経験
- Interactive SQL と Oracle SQL Plus の知識
- Oracle のプログラミングの知識

目的

このチュートリアルの目的は、次のとおりです。

- Oracle で使用する Mobile Link サーバと関連コンポーネントに関する知識の習得
- Oracle 統合データベースに関連する Mobile Link サーバとクライアントのコマンドの実行に関する知識の習得

具体的には、次のタスクを実行します。

- リモート・データベースとして機能する新しい SQL Anywhere データベースの作成
- Oracle 統合データベースで動作する Mobile Link サーバの起動
- Mobile Link 同期クライアントの起動、リモート・データベースと Oracle 統合データベースの同期処理

関連項目

Mobile Link サーバの実行の詳細については、「Mobile Link 同期について」 3 ページを参照してください。

レッスン 1：データベースの作成

Mobile Link 同期を実行するには、統合データベース (このチュートリアルでは Oracle) が 1 つ、リモート・データベース (このチュートリアルでは SQL Anywhere) が 1 つ、これらのデータベースごとに ODBC データ・ソースが 1 つ必要です。

SQL Anywhere のリモート・データベースの作成

♦ **リモート・データベースを作成するには、次の手順に従います。**

1. このチュートリアル用のディレクトリを作成します (ここでは *OracleTut* とします)。コマンド・プロンプトを開き、*OracleTut* に移動します。
2. 次のコマンドを入力します。

   ```
   dbinit remote.db
   ```

3. *OracleTut* の内容一覧を表示して、データベースが正常に作成されたかどうかを確認します。

♦ **リモート・データベース用の ODBC データ・ソースを作成するには、次の手順に従います。**

- *OracleTut* ディレクトリで、次のコマンドを 1 行で入力します。

  ```
  dbdsn -w test_remote -y
    -c "uid=DBA;pwd=sql;dbf=path¥OracleTut¥remote.db;eng=remote"
  ```

  path を *OracleTut* ディレクトリがあるロケーションに置き換えてください。

♦ **リモート・データベース内にオブジェクトを作成するには、次の手順に従います。**

1. Interactive SQL を起動します。

   [スタート] – [プログラム] – [SQL Anywhere 10] – [Interactive SQL] を選択します。
2. リモート・データベースに接続します。
3. リモート・テーブル、パブリケーション、ユーザ、サブスクリプションを作成します。

   次のコードを Interactive SQL にコピーして実行します。

   ```
   CREATE TABLE emp ( emp_id int primary key ,emp_name varchar( 128 ) );
   CREATE TABLE cust(
     cust_id int primary key,
     emp_id int references emp ( emp_id ),
     cust_name varchar( 128 ) );
   CREATE PUBLICATION emp_cust ( TABLE cust, TABLE emp );
   CREATE SYNCHRONIZATION USER ml_user;
   CREATE SYNCHRONIZATION SUBSCRIPTION
    TO emp_cust FOR ml_user TYPE TCPIP ADDRESS 'host=localhost';
   ```

詳細情報

リモート・データベースの作成の詳細については、「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバ - データベース管理』を参照してください。

SQL Anywhere データベース用 ODBC データ・ソースの作成の詳細については、「データ・ソース・ユーティリティ (dbdsn)」『SQL Anywhere サーバ - データベース管理』を参照してください。

Interactive SQL の詳細については、「Interactive SQL ユーティリティ (dbisql)」『SQL Anywhere サーバ - データベース管理』を参照してください。

このチュートリアルで SQL Anywhere オブジェクトを作成する詳細については、次の項を参照してください。

- 「CREATE TABLE 文」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』

Oracle 統合データベースの作成

Oracle データベースには、さまざまな方法でデータを入力できます。このチュートリアルでは、Oracle SQL Plus を使用します。

♦ Oracle 統合データベースを作成するには、次の手順に従います。

1. SQL Plus を起動します。

   [スタート] – [プログラム] – [Oracle - OraDb10g_home1] – [Application Development] – [SQL Plus] を選択します。

2. 統合データベースに接続します。

3. 次のコードを SQL Plus にコピーして実行します。これらの SQL 文は、統合データベースでテーブルの削除、作成、移植を行います。削除するテーブルがない場合は、SQL Plus の出力にエラーが表示されますが、このエラーは処理には影響しません。

```
CREATE SEQUENCE emp_sequence;
CREATE SEQUENCE cust_sequence;
DROP TABLE emp;
CREATE TABLE emp(
  emp_id int primary key,
  emp_name varchar( 128 ) );
DROP TABLE cust;
CREATE TABLE cust(
  cust_id int primary key,
  emp_id int references emp(emp_id),
  cust_name varchar( 128 ) );
INSERT INTO emp
  ( emp_id, emp_name ) VALUES ( emp_sequence.nextval, 'emp1' );
INSERT INTO emp
  ( emp_id, emp_name ) VALUES ( emp_sequence.nextval, 'emp2' );
INSERT INTO emp
  ( emp_id, emp_name ) VALUES ( emp_sequence.nextval, 'emp3' );
COMMIT;
```

```
INSERT INTO cust
  ( cust_id, emp_id, cust_name ) VALUES ( cust_sequence.nextval, 1, 'cust1' );
INSERT INTO cust
  ( cust_id, emp_id, cust_name ) VALUES ( cust_sequence.nextval, 1, 'cust2' );
INSERT INTO cust
  ( cust_id, emp_id, cust_name ) VALUES ( cust_sequence.nextval, 2, 'cust3' );
COMMIT;
```

4. テーブルごとに次の SQL 文を実行して、テーブルが正常に作成されたことを確認します。

   ```
   SELECT * FROM emp;
   SELECT * FROM cust;
   ```

   統合データベースを稼働したままにします。

統合データベース用の ODBC データ・ソースの作成

Mobile Link でデータ同期を実行するには、ODBC データ・ソースが必要です。バージョン 10.0.0 を使用するには、Oracle ODBC ドライバをダウンロードする必要があります。詳細については、http://www.ianywhere.jp/developers/technotes/odbc_mobilink.html を参照してください。

インストール環境の ODBC 部分に必要なインスタンス名、サービス名、データベース名が判明していることを確認します。これらの値は Oracle のインストール時に設定されます。

Oracle 統合データベースの ODBC 設定を行う手順は、次のとおりです。

♦ Oracle 用の ODBC データ・ソースを設定するには、次の手順に従います。

1. [スタート] – [プログラム] – [SQL Anywhere 10] – [SQL Anywhere] – [ODBC アドミニストレータ] を選択します。

   [ODBC データ ソース アドミニストレータ] が開きます。

2. [ユーザー DSN] タブで [追加] をクリックします。[データ ソースの新規作成] ウィンドウが表示されます。

3. **[iAnywhere Solutions 10 - Oracle]** を選択し、[完了] をクリックします。

   [ODBC Oracle ドライバ設定] ウィンドウが表示されます。

4. [General] タブをクリックし、データ・ソース名 **ora_consol** を入力します。これは、Oracle データベースに接続するための DSN です。このデータ・ソース名は後で必要になります。

5. サーバ名を入力します。この値は Oracle インストール環境に応じて異なります。サーバが自分のコンピュータで稼働している場合、このフィールドは空白でかまいません。

6. [Advanced] タブをクリックし、デフォルトのユーザ名を入力します。このチュートリアルでは、**system** を使用するか、オブジェクトの作成権限を持つ任意のユーザ名を使用できます。

7. [OK] をクリックします。

8. [OK] をクリックして、[ODBC データ ソース アドミニストレータ] を閉じます。

詳細情報

Oracle ODBC ドライバの詳細については、「iAnywhere Solutions Oracle ドライバ」『Mobile Link - サーバ管理』を参照してください。

Oracle の詳細については、Oracle のマニュアルを参照してください。

統合データベースの設定

Mobile Link には、スクリプト *syncora.sql* が付属しており、SQL Anywhere インストール環境の *MobiLink¥setup* サブディレクトリにあります。このスクリプトは、Mobile Link で Oracle データベースを使用できるように設定するために実行します。

Syncora.sql には、Oracle SQL で記述された SQL 文が格納されています。この文は、Oracle データベースを統合データベースとして使用する準備を整えます。この文で、Mobile Link で使用する一連の Mobile Link システム・テーブル、トリガ、プロシージャを作成します。システム・テーブル名は ML_ で始まります。Mobile Link は、同期処理中にこれらのテーブルを使用します。

♦ **Oracle 内に Mobile Link システム・テーブルを作成するには、次の手順に従います。**

1. SQL Plus を起動します。[スタート] – [プログラム] – [Oracle - OraDb10g_home1] – [Application Development] – [SQL Plus] を選択します。

   Oracle SQL Plus を使用して Oracle データベースに接続します。**system** スキーマとパスワード **manager** を使用してログオンします。

2. 次のコマンドを入力して *syncora.sql* を実行します。

   ```
   @path¥syncora.sql;
   ```

   path は、SQL Anywhere 10 インストール環境の *MobiLink¥setup* サブディレクトリです。パスにスペースが含まれる場合は、パスとファイル名を引用符で囲んでください。

♦ **システム・テーブルがインストールされたかどうかを確認するには、次の手順に従います。**

1. SQL Plus を起動します。[スタート] – [プログラム] – [Oracle - OraDb10g_home1] – [Application Development] – [SQL Plus] を選択します。
2. 次の SQL 文を実行して、Mobile Link システム・テーブル、プロシージャ、トリガのリストを生成します。

   ```
   SELECT object_name
   FROM all_objects
   WHERE object_name
   LIKE 'ML_%';
   ```

   ML_ で始まるオブジェクトがない場合は、実行したプロシージャが失敗しています。この場合は、Mobile Link エラー・メッセージで問題を確認して訂正し、Mobile Link システム・テーブルを次の手順で削除してください。

♦ **必要に応じて、Mobile Link システム・テーブルを削除するには、次の手順に従います。**

1. SQL Plus で次の SQL 文を実行します。

   ```
   select 'drop ' || object_type || ' ' || object_name || ';'
   from all_objects
   where object_name like 'ML_%';
   ```

削除されるテーブル、プロシージャ、トリガのリストが生成されます。

2. このリストをテキスト・ファイルにコピーし、*drop.sql* として *OracleTut* ディレクトリに保存します。DROP 文を含まない行をすべて削除します。

3. 次のコマンドを実行して、*drop.sql* の SQL 文を実行します。

   ```
   @path¥OracleTut¥drop.sql;
   ```

   path を *OracleTut* ディレクトリがあるロケーションに置き換えてください。依存関係のために 1 回目の実行時に削除されなかったテーブルを再度、*drop.sql* を実行して削除します。

これで、上で説明した Oracle 内に Mobile Link システム・テーブルを作成する手順をもう一度実行できます。

Oracle 統合データベースの設定の詳細については、「Oracle 統合データベース」『Mobile Link - サーバ管理』を参照してください。

レッスン 2：Mobile Link サーバの起動

この時点で、Mobile Link サーバをコマンド・プロンプトから起動できます。Mobile Link サーバは統合データベースのクライアントであるため、統合データベースを起動してから Mobile Link サーバを起動してください。

♦ **Mobile Link サーバを起動するには、次の手順に従います。**

1. 統合データベースが稼働していることを確認します。
2. コマンド・プロンプトで、*OracleTut* ディレクトリに移動します。
3. Mobile Link サーバを起動します。

   次のコマンドを入力します。

   ```
   mlsrv10 -c "dsn=ora_consol;pwd=manager;uid=system" -o mlsrv.mls -v+ -zu+
   ```

このコマンド・ラインは、以下に示す mlsrv10 のオプションを指定します。

- **-c** 接続パラメータを指定します。上記の例では、DSN にユーザ ID が含まれているため、パスワードのみを指定することに注意してください。「-c オプション」 『Mobile Link - サーバ管理』を参照してください。
- **-o** メッセージ・ログ・ファイルを指定します。「-o オプション」 『Mobile Link - サーバ管理』を参照してください。
- **-v+** 冗長ロギングをオンに設定します。「-v オプション」 『Mobile Link - サーバ管理』を参照してください。
- **-dl** ログ表示機能をオンに設定します。「-dl オプション」 『Mobile Link - サーバ管理』を参照してください。
- **-zu+** ユーザ認証処理を自動化します。「-zu オプション」 『Mobile Link - サーバ管理』を参照してください。

詳細情報

mlsrv10 の詳細については、「Mobile Link サーバ」 『Mobile Link - サーバ管理』と「mlsrv10 の構文」 『Mobile Link - サーバ管理』を参照してください。

レッスン 3：Mobile Link 同期クライアントの起動

この時点で、Mobile Link クライアントをコマンド・プロンプトから起動できます。Mobile Link 同期を開始するのはクライアントです。

dbmlsync コマンド・ラインで -c オプションを使用して、リモート・データベースの接続パラメータを指定できます。

♦ **Mobile Link クライアントを起動するには、次の手順に従います。**

1. Mobile Link サーバが起動されていることを確認します。
2. コマンド・プロンプトで、*OracleTut* ディレクトリに移動します。
3. 次のコマンドを入力します。

   ```
   dbmlsync -c "dsn=test_remote" -o dbmlsync.out -v+ -e "SendColumnNames=ON"
   ```

   このコマンド・ラインは以下に示す dbmlsync のオプションを指定します。

   - **-c** データベース接続パラメータを指定します。「-c オプション」 『Mobile Link - クライアント管理』を参照してください。
   - **-o** メッセージ・ログ・ファイルを指定します。「-o オプション」 『Mobile Link - クライアント管理』を参照してください。
   - **-v+** 冗長オペレーション。「-v オプション」 『Mobile Link - クライアント管理』を参照してください。
   - **-e** 拡張オプション。"SendColumnNames=ON" を指定すると、カラム名が Mobile Link に送信されます。「Mobile Link SQL Anywhere クライアントの拡張オプション」 『Mobile Link - クライアント管理』を参照してください。

詳細情報

dbmlsync の詳細については、「dbmlsync 構文」 『Mobile Link - クライアント管理』を参照してください。

詳細情報

Mobile Link サーバの実行の詳細については、「Mobile Link サーバ」 『Mobile Link - サーバ管理』を参照してください。

同期スクリプトの作成の詳細については、「同期スクリプトの作成」 『Mobile Link - サーバ管理』と「同期イベント」 『Mobile Link - サーバ管理』を参照してください。

タイムスタンプベースの同期などのその他の同期方法の概要については、「同期の方法」 『Mobile Link - サーバ管理』を参照してください。

第 6 章

チュートリアル：Java 同期論理の使用

目次

Java 同期チュートリアルの概要

このチュートリアルでは、Java 同期論理を使用するための基本的な手順について説明します。SQL Anywhere 統合データベースとして CustDB サンプルを使用して、Mobile Link のテーブル・レベル・イベント用に簡単なクラス・メソッドを指定します。また、この処理では、コンパイルされた Java クラスのパスを設定するオプションを使用して、Mobile Link サーバ (mlsrv10) を実行します。

必要なソフトウェア

- SQL Anywhere 10.0
- Java ソフトウェア開発キット

前提知識と経験

次の知識と経験が必要です。

- Java の知識
- Mobile Link イベント・スクリプトの基本的な知識

目的

次の項目について、知識と経験を得ることができます。

- Mobile Link テーブル・レベル・イベント用の簡単な Java クラス・メソッドの利用

主要な概念

この項では、次の手順に従って、Mobile Link CustDB サンプル・データベースを使用した基本的な Java ベースの同期を実装します。

- Mobile Link サーバ API リファレンスを使用して、ソース・ファイルをコンパイルします。
- 特定のテーブル・レベル・イベント用のクラス・メソッドを指定します。
- -sl java オプションを使用して、Mobile Link サーバ (mlsrv10) を実行します。
- サンプル Windows クライアント・アプリケーションを使用して、同期をテストします。

関連項目

同期スクリプトの詳細については、「同期スクリプトの概要」『Mobile Link - サーバ管理』を参照してください。

レッスン 1：CustdbScripts Java クラスのコンパイル

Java クラスは、メソッドに同期論理をカプセル化します。

このレッスンでは、CustDB サンプル・データベースに関連するクラスをコンパイルします。

Mobile Link データベース・サンプル

SQL Anywhere には、同期できるように設定された SQL Anywhere サンプル・データベース (CustDB) が付属しています。このデータベースには、同期に必要な SQL スクリプトなどが含まれています。たとえば、CustDB の ULCustomer テーブルは、さまざまなテーブル・レベル・イベントをサポートする同期テーブルです。

CustDB は、Ultra Light クライアントと SQL Anywhere クライアントの両方の統合データベース・サーバとなるように設計されています。CustDB データベースには、SQL Anywhere 10 CustDB という DSN が含まれています。

CustdbScripts クラス

この項では、ULCustomer の upload_insert イベントと download_cursor イベントを処理するための論理を含む Java クラス CustdbScripts を作成します。テキスト・エディタに CustdbScripts コードを入力し、ファイルを *CustdbScripts.java* として保存します。

♦ CustdbScripts.java を作成するには、次の手順に従います。

1. Java クラスとアセンブリ用のディレクトリを作成します。

   このチュートリアルでは、パスを *c:¥mljava* とします。

2. テキスト・エディタを使用して、CustdbScripts コードを入力します。

```
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：**
> クラスと関連するメソッドは、パブリックとして設定する必要があります。

3. ファイルを *CustdbScripts.java* として *c:¥mljava* に保存します。

Java ソース・コードのコンパイル

Java 同期論理を実行するには、Mobile Link サーバが *mlscript.jar* のクラスにアクセスできることが必要です。この jar ファイルには、Java メソッドで利用する Mobile Link サーバ API クラスのレポジトリが入っています。

Mobile Link 用 Java ソース・コードをコンパイルするときは、Mobile Link サーバ API を使用するための *mlscript.jar* を含める必要があります。この項では、javac ユーティリティの -classpath オプションを使用して、*mlscript.jar* を CustdbScripts クラス用に指定します。

♦ **Java ソース・コードをコンパイルするには、次の手順に従います (Windows の場合)。**

1. コマンド・プロンプトで、*CustdbScripts.java* を含むディレクトリ (*c:¥mljava*) に移動します。
2. 次のコマンドを入力します。*install-dir* は、SQL Anywhere インストール環境のディレクトリ名に置き換えてください。

```
javac custdbscripts.java -classpath "install-dir¥java¥mlscript.jar"
```

CustdbScripts.class ファイルが生成されます。

詳細情報

Java 用 Mobile Link サーバ API の詳細については、「Java 用 Mobile Link サーバ API リファレンス」『Mobile Link - サーバ管理』を参照してください。

Java メソッドの詳細については、「メソッド」『Mobile Link - サーバ管理』を参照してください。

CustDB サンプル・データベースの詳細と代替 RDBMS サーバの使用については、「CustDB 統合データベースの設定」 54 ページを参照してください。

レッスン 2：イベントを処理するクラス・メソッドの指定

前のレッスンで作成された *CustdbScripts.class* は、メソッド UploadInsert と DownloadCursor をカプセル化します。これらのメソッドには、それぞれ ULCustomer の upload_insert イベントと download_cursor イベントの実装が含まれています。

この項では、次の 2 つの方法を使用して、テーブル・レベル・イベント用のクラス・メソッドを指定します。

- Sybase Central の Mobile Link の管理モードを使用する方法

  Sybase Central を使用して CustDB データベースに接続し、upload_insert スクリプトの言語を Java に変更して、イベントを処理する CustdbScripts.UploadInsert を指定します。

- ml_add_java_table_script ストアド・プロシージャを使用する方法

  Interactive SQL を使用して CustDB データベースに接続し、ml_add_java_table_script を実行して、download_cursor イベントを処理する CustdbScripts.DownloadCursor を指定します。

♦ ULCustomer の upload_insert イベントを処理する CustdbScripts.UploadInsert を指定するには、次の手順に従います。

1. Sybase Central の Mobile Link の管理モードを使用してサンプル・データベースに接続します。
   - Sybase Central を起動します。
   - [ビュー] メニューをクリックし、[フォルダ] が選択されていることを確認します。
   - [接続] メニューから [Mobile Link 10 に接続] を選択します。
   - [ID] タブで、[ODBC データ・ソース名] に [SQL Anywhere 10] を選択します。
   - [OK] をクリックして接続します。
   - これで、Sybase Central は Mobile Link 10 プラグインで CustDB データ・ソースを表示します。
2. ULCustomer テーブル用の既存の upload_insert イベントを削除します。
   - 左ウィンドウ枠で、[同期テーブル] フォルダを開き、ULCustomer テーブルを選択します。テーブル・レベル・スクリプトのリストが、右ウィンドウ枠に表示されます。
   - custdb 10.0 の upload_insert イベントに関連するテーブル・スクリプトを右クリックします。[編集] メニューから [削除] を選択します。オブジェクトの削除を確認する必要があります。
3. ULCustomer テーブル用に新規の upload_insert イベントを作成します。
   - [同期テーブル] フォルダで ULCustomer テーブルが選択された状態で、[ファイル] – [新規] – [テーブル・スクリプト] を選択します。

- 作成するイベントとして upload_insert を選択し、[次へ] をクリックします。
- Java 言語を使用して新規スクリプトの定義を作成することを選択します。
- [完了] をクリックします。

4. upload_insert スクリプトをダブルクリックして開きます。スクリプトの内容を、完全に修飾されたメソッド名 **CustdbScripts.UploadInsert** に置き換え、スクリプトを保存します。[ファイル] – [保存] を選択します。
5. Sybase Central を終了します。

この手順では、Sybase Central を使用して、Java メソッドを ULCustomer の upload_insert イベント用のスクリプトとして指定しました。

別の方法として、ml_add_java_connection_script ストアド・プロシージャや ml_add_java_table_script ストアド・プロシージャも使用できます。これらのストアド・プロシージャは、特に同期イベントを処理するのに多数の Java メソッドが必要な場合に使用すると、より効率的です。

「ml_add_java_connection_script」『Mobile Link - サーバ管理』と「ml_add_java_table_script」『Mobile Link - サーバ管理』を参照してください。

♦ ULCustomer の download_cursor イベントを処理する CustdbScripts.DownloadCursor() を指定するには、次の手順に従います。

1. Interactive SQL を使用して、サンプル・データベースに接続します。
   - Interactive SQL を開きます。

     [接続] ダイアログが表示されます。
   - [ID] タブで、[ODBC データ・ソース名] に [SQL Anywhere 10] を選択します。
   - [データベース] タブで、[ネットワーク上でデータベース・サーバを検索] オプションが選択されていないことを確認します。
   - [OK] をクリックして接続します。
2. Interactive SQL で次のコマンドを実行します。

```
CALL ml_add_java_table_script(
'custdb 10.0',
'ULCustomer',
'download_cursor',
'CustdbScripts.DownloadCursor');
COMMIT;
```

次に、各パラメータの説明を示します。

パラメータ	説明
custdb 10.0	スクリプト・バージョン
ULCustomer	同期テーブル

パラメータ	説明
download_cursor	イベント名
CustdbScripts.DownloadCursor	完全に修飾された Java メソッド

3. Interactive SQL を終了します。

このレッスンでは、ULCustomer テーブル・レベル・イベントを処理するための Java メソッドを指定しました。次のレッスンでは、Mobile Link サーバが確実に適切なクラス・ファイルと Mobile Link サーバ API をロードするように指定します。

詳細情報

ストアド・プロシージャを使用してスクリプトを追加する方法の詳細については、「ml_add_java_connection_script」『Mobile Link - サーバ管理』と「ml_add_java_table_script」『Mobile Link - サーバ管理』を参照してください。

同期スクリプトの追加方法と削除方法の概要については、「スクリプトの追加と削除」『Mobile Link - サーバ管理』を参照してください。

レッスン 3：-sl java を使用した Mobile Link サーバの実行

-sl java -cp オプションを使用して Mobile Link サーバを実行すると、クラス・ファイルを検索するための一連のディレクトリを指定して、Java 仮想マシンをサーバ起動時にロードできます。

♦ Mobile Link サーバ (mlsrv10) を起動し、Java アセンブリをロードするには、次の手順に従います。

- コマンド・プロンプトで、次のコマンドを 1 行で入力します。

  ```
  mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -sl java (-cp c:¥mljava)
  ```

  サーバが起動されたことを示すメッセージが表示されます。これで、同期中に ULCustomer テーブルの upload_insert イベントがトリガされると Java メソッドが実行されるようになりました。

詳細情報

詳細については、「-sl java オプション」『Mobile Link - サーバ管理』を参照してください。

レッスン 4：同期のテスト

Ultra Light には、サンプル Windows クライアントが用意されています。このクライアントは、ユーザが同期を発行したときに自動的に dbmlsync ユーティリティを起動します。これは簡単な販売状況アプリケーションで、前のレッスンで起動した CustDB 統合データベースに対して実行できます。

アプリケーションの起動 (Windows)

♦ **サンプル・アプリケーションを起動して同期するには、次の手順に従います。**

1. サンプル・アプリケーションを起動します。

   [スタート] – [プログラム] – [SQL Anywhere 10] – [Ultra Light] – [Windows アプリケーションのサンプル] を選択します。

2. 従業員 ID を入力します。

   50 の値を入力し、[Enter] を押します。

アプリケーションは自動的に同期を実行し、一連の顧客、製品、注文が CustDB 統合データベースからアプリケーションにダウンロードされます。

注文情報の追加 (Windows)

♦ **注文情報を追加するには、次の手順に従います。**

1. [Order] – [New] を選択します。

   [Add New Order] 画面が表示されます。

2. 新しい顧客名を入力します。

   たとえば、**Frank Javac** と入力します。

3. 製品を選択し、数量と割引率を入力します。

4. [Enter] を押して、新しい注文を追加します。

これで、ローカルの Ultra Light データベースでデータが修正されました。このデータは、同期が行われるまで統合データベースとは共有されません。

♦ **統合データベースと同期し、upload_insert イベントをトリガするには、次の手順に従います。**

· [File] – [Synchronize] を選択します。

統合データベースに挿入が正常にアップロードされたことを示すウィンドウが表示されます。

詳細情報

CustDB Windows アプリケーションの詳細については、「Mobile Link CustDB サンプルの解説」 51 ページを参照してください。

クリーンアップ

チュートリアルをコンピュータから削除します。

♦ **チュートリアルをコンピュータから削除するには、次の手順に従います。**

1. ULCustomer テーブルの upload_insert スクリプトと download_cursor スクリプトを元の SQL 論理に戻します。

   ♦ Interactive SQL を開きます。

   [接続] ダイアログ・ボックスが表示されます。

   ♦ [ID] タブで、[ODBC データ・ソース名] に [SQL Anywhere 10] を選択します。

   ♦ [OK] をクリックして接続します。

   ♦ Interactive SQL で次のコマンドを実行します。

```
CALL ml_add_table_script( 'custdb 10.0',
  'ULCustomer',
  'upload_insert',
  'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

CALL ml_add_table_script( 'custdb 10.0',
  'ULCustomer',
  'download_cursor',
  'SELECT "cust_id", "cust_name"
   FROM "ULCustomer" WHERE "last_modified" >= ?' );
COMMIT;
```

2. タスクバー上で、SQL Anywhere、Mobile Link、同期クライアントの各項目を右クリックし、[閉じる] を選択して閉じます。

3. チュートリアルに関連するすべての Java ソースを削除します。

   CustdbScripts.java ファイルと *CustdbScripts.class* ファイルを含むフォルダ (*c:¥mljava*) を削除します。

> **注意：**
> *c:¥mljava* に他の重要なファイルが含まれていないことを確認してください。

詳細情報

Java で Mobile Link 同期スクリプトを作成する方法の詳細については、「Java 同期論理の設定」『Mobile Link - サーバ管理』を参照してください。

カスタム認証用の Java 同期スクリプトの使用例については、「Java 同期の例」 『Mobile Link - サーバ管理』を参照してください。

同期スクリプトの作成方法の詳細については、「同期スクリプトの作成」 『Mobile Link - サーバ管理』と「同期イベント」 『Mobile Link - サーバ管理』を参照してください。

タイムスタンプベースの同期などのその他の同期方法の概要については、「同期の方法」 『Mobile Link - サーバ管理』を参照してください。

第 7 章

チュートリアル：.NET 同期論理の使用

目次

.NET 同期チュートリアルの概要

このチュートリアルでは、.NET 同期論理を使用するための基本的な手順について説明します。SQL Anywhere 統合データベースとして CustDB サンプルを使用して、Mobile Link のテーブル・レベル・イベント用に簡単なクラス・メソッドを指定します。また、この処理では、.NET アセンブリのパスを設定するオプションを使用して、Mobile Link サーバ (mlsrv10) を実行します。

必要なソフトウェア

- SQL Anywhere 10.0
- Microsoft .NET Framework SDK

前提知識と経験

次の知識と経験が必要です。

- .NET の知識
- Mobile Link イベント・スクリプトの基本的な知識

目的

次の項目について、知識と経験を得ることができます。

- Mobile Link テーブル・レベル・イベント・スクリプト用の .NET クラス・メソッドの利用

主要な概念

この項では、次の手順に従って、Mobile Link CustDB サンプル・データベースを使用した基本的な .NET 同期を実装します。

- Mobile Link 参照を含む *CustdbScripts.dll* プライベート・アセンブリをコンパイルします。
- テーブル・レベル・イベント用のクラス・メソッドを指定します。
- -sl dnet オプションを使用して、Mobile Link サーバ (mlsrv10) を実行します。
- サンプル Windows クライアント・アプリケーションを使用して、同期をテストします。

関連項目

同期スクリプトの詳細については、「同期スクリプトの概要」 『Mobile Link - サーバ管理』を参照してください。

レッスン 1：Mobile Link 参照を含む CustdbScripts.dll アセンブリのコンパイル

.NET クラスは、メソッドに同期論理をカプセル化します。

このレッスンでは、CustDB サンプル・データベースに関連するクラスをコンパイルします。

Mobile Link データベース・サンプル

SQL Anywhere には、同期できるように設定された SQL Anywhere サンプル・データベース (CustDB) が付属しています。このデータベースには、同期に必要な SQL スクリプトなどが含まれています。たとえば、CustDB の ULCustomer テーブルは、さまざまなテーブル・レベル・イベントをサポートする同期テーブルです。

CustDB は、Ultra Light クライアントと SQL Anywhere クライアントの両方の統合データベース・サーバとなるように設計されています。CustDB データベースには、SQL Anywhere 10 CustDB という DSN が含まれています。

CustdbScripts アセンブリ

この項では、ULCustomer の upload_insert イベントと download_cursor イベントを処理するための論理を含む .NET クラス CustdbScripts を作成します。

Mobile Link サーバ API

.NET 同期論理を実行するには、Mobile Link サーバが *iAnywhere.MobiLink.Script.dll* 内のクラスにアクセスできることが必要です。*iAnywhere.MobiLink.Script.dll* には、.NET メソッドで利用する .NET クラス用 Mobile Link サーバ API のリポジトリが入っています。

.NET 用 Mobile Link サーバ API の詳細については、「.NET 用 Mobile Link サーバ API リファレンス」『Mobile Link - サーバ管理』を参照してください。

この API を利用する場合、CustdbScripts クラスのコンパイル時にここで示したアセンブリを含める必要があります。クラスのコンパイルは、Visual Studio .NET を使用するか、コマンド・プロンプトから実行できます。

- Visual Studio .NET を使用する場合、新しいクラス・ライブラリを作成し、CustdbScripts コードを入力します。*iAnywhere.MobiLink.Script.dll* をリンクし、クラスのアセンブリを構築します。
- コマンド・プロンプトでは、テキスト・エディタに CustdbScripts コードを入力し、ファイルを *CustdbScripts.cs* (Visual Basic .NET の場合は *CustdbScripts.vb*) として保存します。コマンド・ライン・コンパイラを使用して *iAnywhere.MobiLink.Script.dll* を参照し、クラスのアセンブリを構築します。

♦ Visual Studio .NET を使用して CustdbScripts アセンブリを作成するには、次の手順に従います。

1. 新しい Visual C# または Visual Basic .NET クラス・ライブラリ・プロジェクトを起動します。

プロジェクト名として CustdbScripts を使用し、適切なパスを入力します。このチュートリアルでは、パスを *c:¥mldnet* とします。

2. CustdbScripts コードを入力します。

   C# の場合、次のように入力します。

```
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") +"'");
}
}
}
```

   Visual Basic .NET の場合、次のように入力します。

```
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. Mobile Link サーバ API への参照を追加します。

   - Visual Studio .NET で、[プロジェクト] – [既存項目の追加] を選択します。
   - SQL Anywhere インストール環境の *Assembly¥v1* サブディレクトリにある *iAnywhere.MobiLink.Script.dll* を選択します。Visual Studio .NET の場合は、[開く] ドロップダウン・メニューから [リンク ファイル] を選択します。Visual Studio 2005 の場合は、[Add] – [Add Link] を選択します。

4. CustdbScripts プロジェクトを右クリックし、テーブルの [共通プロパティ] – [全般] を選択します。すべてのテキストについて、[ルート名前空間] テキスト・ボックスがオフになっていることを確認してください。

5. *CustdbScripts.dll* をビルドします。

   [ビルド] – [CustdbScripts のビルド] を選択します。

これにより、*C:¥mldnet¥CustdbScripts¥CustdbScripts¥bin¥Debug* に *CustdbScripts.dll* が作成されます。

♦ **コマンド・プロンプトで、CustdbScripts アセンブリを作成するには、次の手順に従います。**

1. .NET クラスとアセンブリ用のディレクトリを作成します。

   このチュートリアルでは、パスを *c:¥mldnet* とします。

2. テキスト・エディタを使用して、CustdbScripts コードを入力します。

   C# の場合、次のように入力します。

```
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") +"'");
 }
}
}
```

   Visual Basic .NET の場合、次のように入力します。

```
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. ファイルを *c:¥mldnet* に *CustdbScripts.cs* (Visual Basic .NET の場合は *CustdbScripts.vb*) として保存します。

4. 次のコマンドを使用して、ファイルをコンパイルします。

   C# の場合、次のように入力します。

```
csc /out:c:¥mldnet¥custdbscripts.dll /target:library /reference:"%sqlany10%¥Assembly¥v1
¥iAnywhere.MobiLink.Script.dll" c:¥mldnet¥CustdbScripts.cs
```

   Visual Basic .NET の場合、次のように入力します。


```
vbc /out:c:¥mldnet¥custdbscripts.dll /target:library /reference:"%sqlany10%¥Assembly¥v1
¥iAnywhere.MobiLink.Script.dll" c:¥mldnet¥CustdbScripts.vb
```

CustdbScripts.dll アセンブリが生成されます。

詳細情報

.NET 用 Mobile Link サーバ API の詳細については、「.NET 用 Mobile Link サーバ API リファレンス」『Mobile Link - サーバ管理』を参照してください。

.NET メソッドの詳細については、「メソッド」『Mobile Link - サーバ管理』を参照してください。

レッスン 2：イベント用のクラス・メソッドの指定

CustDB サンプル・データベースの詳細については、「CustDB 統合データベースの設定」 54 ページを参照してください。

前のレッスンで作成された *CustdbScripts.dll* は、メソッド UploadInsert() と DownloadCursor() をカプセル化します。これらのメソッドには、それぞれ ULCustomer の upload_insert イベントと download_cursor イベントの実装が含まれています。

この項では、次の 2 つの方法を使用して、テーブル・レベル・イベント用のクラス・メソッドを指定します。

1. Mobile Link 同期プラグインを使用する方法

   Sybase Central を使用して CustDB データベースに接続し、upload_insert スクリプトの言語を .NET に変更して、イベントを処理するための MLExample.CustdbScripts.UploadInsert を指定します。

2. ml_add_dnet_table_script ストアド・プロシージャを使用する方法

   Interactive SQL を使用して CustDB データベースに接続し、ml_add_dnet_table_script を実行して、download_cursor イベント用の MLExample.CustdbScripts.DownloadCursor を指定します。

♦ ULCustomer テーブル用の upload_insert イベントに CustdbScripts.uploadInsert() をサブスクライブするには、次の手順に従います。

1. Mobile Link 同期プラグインを使用して、サンプル・データベースに接続します。

   - Sybase Central を起動します。
   - [ビュー] メニューで、[フォルダ] が選択されていることを確認します。
   - [接続] メニューから [Mobile Link 10 に接続] を選択します。
   - [ID] タブで、[ODBC データ・ソース名] に [SQL Anywhere 10] を選択します。
   - [OK] をクリックして接続します。
   - これで、Sybase Central は Mobile Link 10 プラグインで CustDB データ・ソースを表示します。

2. ULCustomer テーブルの upload_insert イベントの言語を .NET に変更します。

   - 左ウィンドウ枠で、[同期テーブル] フォルダを開き、ULCustomer テーブルを選択します。テーブル・レベル・スクリプトのリストが、右ウィンドウ枠に表示されます。
   - custdb 10.0 の upload_insert イベントに関連するテーブル・スクリプトを右クリックします。[ファイル] – [言語] を選択し、言語を .NET に変更します。

3. upload_insert スクリプト用として、完全に修飾された .NET メソッド名を入力します。

   - upload_insert イベントに関連するテーブル・スクリプトをダブルクリックします。

スクリプトの内容を示すウィンドウが表示されます。

- スクリプトの内容を、完全に修飾されたメソッド名 **MLExample.CustdbScripts.UploadInsert** に変更します。

  注意：
  完全に修飾されたメソッド名では、大文字と小文字が区別されます。

- [ファイル] – [保存] を選択して、スクリプトを保存します。

4. Sybase Central を終了します。

この手順では、Sybase Central を使用して、.NET メソッドを ULCustomer の upload_insert イベント用のスクリプトとして指定しました。

別の方法として、ml_add_dnet_connection_script ストアド・プロシージャや ml_add_dnet_table_script ストアド・プロシージャも使用できます。これらのストアド・プロシージャは、特に同期イベントを処理するのに多数の .NET メソッドが必要な場合に使用すると、より効率的です。

「ml_add_dnet_connection_script」『Mobile Link - サーバ管理』と「ml_add_dnet_table_script」『Mobile Link - サーバ管理』を参照してください。

次の項では、Interactive SQL を使用して CustDB に接続し、ml_add_dnet_table_script を実行して、download_cursor イベントに MLExample.CustdbScripts.DownloadCursor を割り当てます。

♦ ULCustomer の download_cursor イベント用の MLExample.CustdbScripts.DownloadCursor を指定するには、次の手順に従います。

1. Interactive SQL を使用して、サンプル・データベースに接続します。

   - Interactive SQL を起動します。

     [スタート] – [プログラム] – [SQL Anywhere 10] – [Interactive SQL] を選択するか、コマンド・プロンプトで次のコマンドを入力します。

     ```
     dbisql
     ```

     [接続] ダイアログが表示されます。

   - [ID] タブで、[ODBC データ・ソース名] に [SQL Anywhere 10] を選択します。
   - [データベース] タブで、ネットワーク・データベース・サーバの検索オプションが選択されていないことを確認します。
   - [OK] をクリックして接続します。

2. Interactive SQL で次のコマンドを実行します。

   ```
   CALL ml_add_dnet_table_script(
   'custdb 10.0',
   'ULCustomer',
   'download_cursor',
   ```

```
'MLExample.CustdbScripts.DownloadCursor');
COMMIT;
```

次に、各パラメータの説明を示します。

パラメータ	説明
custdb 10.0	スクリプト・バージョン
ULCustomer	同期テーブル
download_cursor	イベント名
MLExample.CustdbScripts.DownloadCursor	完全に修飾された .NET メソッド

3. Interactive SQL を終了します。

このレッスンでは、ULCustomer テーブル・レベル・イベントを処理するための .NET メソッドを指定しました。次のレッスンでは、Mobile Link サーバが確実に適切なクラス・ファイルと Mobile Link サーバ API をロードするように指定します。

詳細情報

同期スクリプトの追加方法と削除方法の詳細については、「スクリプトの追加と削除」『Mobile Link - サーバ管理』を参照してください。

このレッスンで使用したスクリプトの詳細については、「ml_add_dnet_connection_script」『Mobile Link - サーバ管理』と「ml_add_dnet_table_script」『Mobile Link - サーバ管理』を参照してください。

レッスン 3：-sl dnet を使用した Mobile Link の実行

-sl dnet オプションを使用して Mobile Link サーバを実行すると、.NET アセンブリのロケーションを指定して、CLR をサーバ起動時にロードできます。

コンパイルに Visual Studio .NET を使用した場合、*CustdbScripts.dll* のロケーションは *c:¥mldnet¥CustdbScripts¥CustdbScripts¥bin¥Debug* になります。コマンド・プロンプトを使用した場合、*CustdbScripts.dll* のロケーションは *c:¥mldnet* になります。

♦ Mobile Link サーバ (mlsrv10) を起動し、.NET アセンブリをロードするには、次の手順に従います。

- -sl dnet オプションを使用して、Mobile Link サーバを起動します。

  Visual Studio .NET を使用してアセンブリをコンパイルした場合は、次の手順に従います。

  コマンド・プロンプトで、次のコマンドを 1 行で入力します。

  ```
  mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -dl -o cons1.txt -v+ -sl dnet(-MLAutoLoadPath=c:¥mldnet¥CustdbScripts¥CustdbScripts¥bin¥Debug)
  ```

  コマンド・プロンプトでアセンブリをコンパイルした場合は、次の手順に従います。

  コマンド・プロンプトで、次のコマンドを 1 行で入力します。

  ```
  mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -dl -o cons1.txt -v+ -sl dnet(-MLAutoLoadPath=c:¥mldnet)
  ```

  サーバが要求を処理する準備ができたことを示すメッセージ・ダイアログが表示されます。これで、同期中に upload_insert イベントがトリガされると .NET メソッドが実行されるようになりました。

詳細情報

詳細については、「-sl dnet オプション」 『Mobile Link - サーバ管理』を参照してください。

レッスン 4：同期のテスト

Ultra Light には、サンプル Windows クライアントが用意されています。このクライアントは、ユーザが同期を発行したときに自動的に dbmlsync ユーティリティを起動します。これは簡単な販売状況アプリケーションで、前のレッスンで起動した CustDB 統合データベースに対して実行できます。

アプリケーションの起動

♦ **サンプル・アプリケーションを起動して同期するには、次の手順に従います。**

1. サンプル・アプリケーションを起動します。

   [スタート] – [プログラム] – [SQL Anywhere 10] – [Ultra Light] – [Windows アプリケーションのサンプル] を選択します。

2. 従業員 ID を入力します。

   50 の値を入力し、[Enter] を押します。

アプリケーションは自動的に同期を実行し、一連の顧客、製品、注文が CustDB 統合データベースからアプリケーションにダウンロードされます。

次の項では、新しい顧客名と注文情報を入力します。この情報は、今後発生する同期中に CustDB 統合データベースにアップロードされ、ULCustomer テーブルの upload_insert イベントと download_cursor イベントがトリガされます。

注文の追加

♦ **注文情報を追加するには、次の手順に従います。**

1. [Order] – [New] を選択します。

   [Add New Order] ダイアログが表示されます。

2. 新しい顧客名を入力します。

   たとえば、Frank DotNET と入力します。

3. 製品を選択し、数量と割引率を入力します。

4. [Enter] を押して、新しい注文を追加します。

これで、ローカルの Ultra Light データベースでデータが修正されました。このデータは、同期が行われるまで統合データベースとは共有されません。

♦ **統合データベースと同期し、upload_insert イベントをトリガするには、次の手順に従います。**

1. [File] – [Synchronize] を選択します。

   統合データベースに挿入が正常にアップロードされたことを示すウィンドウが表示されます。

詳細情報

CustDB Windows アプリケーションの詳細については、「Mobile Link CustDB サンプルの解説」 51 ページを参照してください。

クリーンアップ

チュートリアルをコンピュータから削除します。

♦ チュートリアルをコンピュータから削除するには、次の手順に従います。

1. ULCustomer テーブルの upload_insert スクリプトと download_cursor スクリプトを元の SQL 論理に戻します。

   - Interactive SQL を開きます。

     [スタート] – [プログラム] – [SQL Anywhere 10] – [Interactive SQL] を選択するか、コマンド・プロンプトで次のコマンドを入力します。

     ```
     dbisql
     ```

     [接続] ダイアログ・ボックスが表示されます。
   - [ID] タブで、[ODBC データ・ソース名] に [SQL Anywhere 10] を選択します。
   - [OK] をクリックして接続します。
   - Interactive SQL で次のコマンドを実行します。

     ```
     CALL ml_add_table_script( 'custdb 10.0',
       'ULCustomer',
       'upload_insert',
       'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

     CALL ml_add_table_script( 'custdb 10.0',
       'ULCustomer',
       'download_cursor',
       'SELECT "cust_id", "cust_name"
        FROM "ULCustomer"
        WHERE "last_modified" >= ?' );
     ```

2. タスクバー上で、SQL Anywhere、Mobile Link、同期クライアントの各項目を右クリックし、[閉じる] を選択して閉じます。
3. チュートリアルに関連するすべての .NET ソースを削除します。

   CustdbScripts.cs ファイルと *CustdbScripts.dll* ファイルを含むフォルダ (*c:¥mldnet*) を削除します。

> **注意：**
> *c:¥mldnet* に他の重要なファイルが含まれていないことを確認してください。

詳細情報

.NET で Mobile Link 同期スクリプトを作成する方法の詳細については、「.NET 同期論理の設定」『Mobile Link - サーバ管理』を参照してください。

.NET 同期論理をデバッグする方法の詳細については、「.NET 同期論理のデバッグ」『Mobile Link - サーバ管理』を参照してください。

カスタム認証用の .NET 同期スクリプトの使用例については、「.NET 同期のサンプル」『Mobile Link - サーバ管理』を参照してください。

同期スクリプトの作成方法の詳細については、「同期スクリプトの作成」『Mobile Link - サーバ管理』と「同期イベント」『Mobile Link - サーバ管理』を参照してください。

タイムスタンプベースの同期などのその他の同期方法の概要については、「同期の方法」『Mobile Link - サーバ管理』を参照してください。

第 8 章

チュートリアル：カスタム認証用の .NET と Java の使用

目次

Mobile Link カスタム認証の概要

Mobile Link 同期スクリプトは、SQL、Java、または .NET で作成できます。Java または .NET を使用すると、同期処理の任意の時点にカスタム・アクションを追加できます。

このチュートリアルでは、authenticate_user 接続イベントに対する Java または .NET メソッドを追加します。authenticate_user イベントを使用すると、カスタム認証スキームを指定して、Mobile Link の組み込みクライアント認証を無効にできます。

必要なソフトウェア

- SQL Anywhere 10.0
- Java ソフトウェア開発キット

前提知識と経験

次の知識と経験が必要です。

- Java の知識
- Mobile Link イベント・スクリプトの基本的な知識

目的

次の項目について、知識と経験を得ることができます。

- Mobile Link カスタム認証

主要な概念

この項では、次の手順に従って、Mobile Link CustDB サンプル・データベースを使用した基本的な Java ベースの同期を実装します。

- Mobile Link サーバ API リファレンスを使用して、ソース・ファイルをコンパイルします。
- 特定のテーブル・レベル・イベント用のクラス・メソッドを指定します。
- -sl java オプションを使用して、Mobile Link サーバ (mlsrv10) を実行します。
- サンプル Windows クライアント・アプリケーションを使用して、同期をテストします。

関連項目

Mobile Link クライアントの認証の詳細については、「ユーザ認証メカニズムの選択」 『Mobile Link - クライアント管理』を参照してください。

POP3、IMAP、または LDAP 認証の統合の詳細については、「外部サーバに対する認証」 『Mobile Link - クライアント管理』を参照してください。

.NET または Java の同期スクリプトの詳細については、「.NET での同期スクリプトの作成」 『Mobile Link - サーバ管理』 (.NET の場合)、または「Java による同期スクリプトの作成」 『Mobile Link - サーバ管理』 (Java の場合) を参照してください。

レッスン 1：カスタム認証用の Java または .NET クラスの作成 (サーバ側)

このレッスンでは、カスタム認証用の Java または .NET 同期論理を記述するクラスをコンパイルします。

.NET 用 Mobile Link サーバ API

.NET 同期論理を実行するには、Mobile Link サーバが *iAnywhere.MobiLink.Script.dll* 内のクラスにアクセスできることが必要です。*iAnywhere.MobiLink.Script.dll* には、.NET メソッドで利用する Mobile Link .NET サーバ API クラスのリポジトリが入っています。.NET クラスをコンパイルするときは、*iAnywhere.MobiLink.Script.dll* を参照します。

Java 用 Mobile Link サーバ API

Java 同期論理を実行するには、Mobile Link サーバが *mlscript.jar* 内のクラスにアクセスできることが必要です。*mlscript.jar* には、Java メソッドで利用する Mobile Link Java サーバ API クラスのリポジトリが入っています。Java クラスをコンパイルするときは、*mlscript.jar* を参照します。

♦ **カスタム認証用の Java または .NET クラスを作成するには、次の手順に従います。**

1. Java または .NET を使用して、MobiLinkAuth というクラスを作成します。

   MobiLinkAuth クラスには、authenticate_user 同期イベントで使用する authenticateUser メソッドが含まれています。authenticate_user イベントは、ユーザ・パラメータとパスワード・パラメータを提供します。認証結果は、authentication_status inout パラメータを使用して返します。

   Java の場合は、次のように入力します。

```
import ianywhere.ml.script.*;

public class MobiLinkAuth
{

 public void authenticateUser (
  ianywhere.ml.script.InOutInteger authentication_status,
  String user,
  String pwd,
  String newPwd )
 {
   // to do...

 }

}
```

   .NET の場合は、次のように入力します。

```
using iAnywhere.MobiLink.Script;

public class MobiLinkAuth
{
  public void authenticateUser  (
    ref int authentication_status,
    string user,
```



```
    string pwd,
    string newPwd)
  {

   // to do...

  }
 }
```

2. authenticateUser メソッドを作成します。

   このチュートリアルでは、カスタム・ユーザ認証の非常に簡単な例を取り上げて説明します。ユーザ名が "ML" で始まる場合は、認証が成功します。

   注意：
   authenticate_user 同期イベントの authenticateUser メソッドは、「レッスン 2：authenticate_user イベント用の Java または .NET スクリプトの登録」 128 ページに登録します。

   Java の場合は、次のように入力します。

```
public void authenticateUser (
 ianywhere.ml.script.InOutInteger authentication_status,
 String user,
 String pwd,
 String newPwd ) {

if(user.substring(0,1)=="ML") {
      // success: an auth status code of 1000
      authentication_status.setValue(1000);
   } else {
      // fail: an authentication_status code of 4000
      authentication_status.setValue(4000);
   }
}
```

   .NET の場合は、次のように入力します。

```
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. MobiLinkAuth クラスをコンパイルします。

   - .NET の場合は、*MobiLinkAuth.cs* という名前を付けて、ファイルを *c:¥mlauth* に保存します。

- コマンド・プロンプトで、*c:¥mlauth* に移動します。次のコマンドを 1 行で入力し、ファイルをコンパイルします。*install-dir* は、SQL Anywhere インストール環境のディレクトリ名に置き換えてください。

  ```
  csc /out:c:¥mlauth¥MobiLinkAuth.dll /target:library
  /reference:install-dir¥Assembly¥v1¥iAnywhere.MobiLink.Script.dll" MobiLinkAuth.cs
  ```

  MobiLinkAuth.dll アセンブリが生成されます。

- Java の場合は、*MobiLinkAuth.java* という名前を付けて、ファイルを *c:¥mlauth* に保存します。コマンド・プロンプトで、*c:¥mlauth* に移動します。次のコマンドを 1 行で入力し、ファイルをコンパイルします。*install-dir* は、SQL Anywhere インストール環境のディレクトリ名に置き換えてください。

  ```
  javac MobiLinkAuth.java -classpath "install-dir/java/mlscript.jar"
  ```

  MobiLinkAuth.class ファイルが生成されます。

詳細情報

authenticate_user イベントの詳細 (authentication_status のリターン・コード表も含む) については、「authenticate_user 接続イベント」『Mobile Link - サーバ管理』を参照してください。

Java または .NET を使用したカスタム認証の実装の詳細については、「Java と .NET のユーザ認証」『Mobile Link - クライアント管理』を参照してください。

Java カスタム認証の詳細な例については、「Java 同期の例」『Mobile Link - サーバ管理』を参照してください。

.NET カスタム認証の詳細な例については、「.NET 同期のサンプル」『Mobile Link - サーバ管理』を参照してください。

レッスン 2：authenticate_user イベント用の Java または .NET スクリプトの登録

このレッスンでは、authenticate_user 同期イベント用の MobiLinkAuth authenticateUser メソッドを登録します。このスクリプトは、Mobile Link サンプル・データベースである CustDB に追加します。

Mobile Link データベース・サンプル

SQL Anywhere には、同期できるように設定された SQL Anywhere サンプル・データベース (CustDB) が付属しています。たとえば、CustDB の ULCustomer テーブルは、さまざまなテーブル・レベル・スクリプトをサポートする同期テーブルです。

CustDB は、Ultra Light クライアントと SQL Anywhere クライアントの両方の統合データベース・サーバとなるように設計されています。CustDB データベースには、SQL Anywhere 10 CustDB という DSN が含まれています。

♦ authenticate_user イベント用の authenticateUser メソッドを登録するには、次の手順に従います。

1. Interactive SQL を使用してサンプル・データベースに接続します。

   コマンド・プロンプトで次のコマンドを入力します。

   ```
   dbisql -c "dsn=SQL Anywhere 10 CustDB"
   ```

2. ml_add_java_connection_script または ml_add_dnet_connection_script ストアド・プロシージャを使用して、authenticate_user イベント用の authenticateUser メソッドを登録します。

   Java の場合は、Interactive SQL で次のコマンドを実行します。

   ```
   call ml_add_java_connection_script(
   'custdb 10.0',
   'authenticate_user',
   'MobiLinkAuth.authenticateUser');
   commit;
   ```

   .NET の場合は、Interactive SQL で次のコマンドを実行します。

   ```
   call ml_add_dnet_connection_script(
   'custdb 10.0',
   'authenticate_user',
   'MobiLinkAuth.authenticateUser');
   commit;
   ```

次のレッスンでは、Mobile Link サーバを起動して、クラス・ファイルまたはアセンブリをロードします。

詳細情報

同期スクリプトの追加方法と削除方法の概要については、「スクリプトの追加と削除」『Mobile Link - サーバ管理』を参照してください。

ml_add_java_connection_script の詳細については、「ml_add_java_connection_script」『Mobile Link - サーバ管理』を参照してください。

ml_add_dnet_connection_script の詳細については、「ml_add_dnet_connection_script」『Mobile Link - サーバ管理』を参照してください。

レッスン 3：Java または .NET 用に Mobile Link サーバを起動

-sl java オプションまたは -sl dnet オプションを指定して Mobile Link サーバを起動すると、コンパイル済みファイルを検索するための一連のディレクトリを指定できます。

♦ **Mobile Link サーバ (mlsrv10) を起動するには、次の手順に従います。**

1. Mobile Link サンプル・データベースに接続して、 mlsrv10 コマンド・ラインで Java クラスまたは .NET アセンブリをロードします。

   Java の場合は、コマンド・プロンプトで次のように入力します。

   ```
   mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -sl java(-cp c:¥mlauth)
   ```

   .NET の場合は、コマンド・プロンプトで次のように入力します。

   ```
   mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -sl dnet(-MLAutoLoadPath=c:¥mlauth)
   ```

   Mobile Link サーバ・ウィンドウが表示されます。これで、authenticate_user 同期イベントの発生時に Java または .NET メソッドが実行されます。

詳細情報

Java 用に Mobile Link サーバを起動する詳細については、「-sl java オプション」『Mobile Link - サーバ管理』を参照してください。

.NET 用に Mobile Link サーバを起動する詳細については、「-sl dnet オプション」『Mobile Link - サーバ管理』を参照してください。

レッスン 4：認証のテスト

Ultra Light には、サンプル Windows クライアントが用意されています。このクライアントは、ユーザが同期を発行したときに自動的に dbmlsync ユーティリティを起動します。これは簡単な販売状況アプリケーションで、前のレッスンで起動した CustDB 統合データベースに対して実行できます。

♦ **サンプル・アプリケーションを起動して認証をテストするには、次の手順に従います。**

1. サンプル・アプリケーションを起動します。

   [スタート] – [プログラム] – [SQL Anywhere 10] – [Ultra Light] – [Windows アプリケーションのサンプル] を選択します。

2. 無効な従業員 ID を入力して同期します。

   このアプリケーションでは、従業員 ID は Mobile Link ユーザ名でもあります。"ML" で始まらないユーザ名の場合、Java または .NET 論理により同期が失敗します。従業員 ID に **50** の値を入力し、[Enter] を押します。

   Ultra Light CustDB デモのダイアログ・ボックスが開き、SQL コード -103 の同期エラーが発生したことが示されます。SQL コード -103 は、ユーザ ID またはパスワードが無効であることを示します。

詳細情報

CustDB Windows アプリケーションの詳細については、「Mobile Link CustDB サンプルの解説」 51 ページを参照してください。

クリーンアップ

チュートリアルをコンピュータから削除します。

♦ **チュートリアルをコンピュータから削除するには、次の手順に従います。**

1. authenticate_user スクリプトを統合データベースから削除します。

   ♦ Interactive SQL を使用して、Mobile Link サンプル・データベースに接続します。

   コマンド・プロンプトで次のコマンドを入力します。

   ```
   dbisql -c "dsn=SQL Anywhere 10 CustDB"
   ```

   ♦ authenticate_user スクリプトを削除します。

   Java の場合は、次のコマンドを実行して authenticate_user スクリプトを削除します。

   ```
   call ml_add_java_connection_script(
   'custdb 10.0',
   'authenticate_user',
   null);
   commit;
   ```

   .NET の場合は、次のコマンドを実行して authenticate_user スクリプトを削除します。

   ```
   call ml_add_dnet_connection_script(
   'custdb 10.0',
   'authenticate_user',
   null);
   commit;
   ```

2. Java または .NET のソース・ファイルを削除します。

   たとえば、*c:¥mlauth* ディレクトリを削除します。

   警告
   このディレクトリには、チュートリアル関連のファイルのみが含まれていることを確認してください。

3. Interactive SQL と Ultra Light Windows クライアント・アプリケーションを終了します。

   各アプリケーションの [ファイル] メニューから ［終了］ を選択します。

4. SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを閉じます。

   それぞれのタスクバーを右クリックして、［閉じる］ を選択します。

詳細情報

Java 同期論理の詳細については、「Java による同期スクリプトの作成」 『Mobile Link - サーバ管理』を参照してください。

.NET 同期論理の詳細については、「.NET での同期スクリプトの作成」 『Mobile Link - サーバ管理』を参照してください。

カスタム認証用の Java または .NET 同期スクリプトの使用例については、「Java 同期の例」 『Mobile Link - サーバ管理』 (Java の場合)、または「.NET 同期のサンプル」 『Mobile Link - サーバ管理』 (.NET の場合) を参照してください。

Java または .NET 同期論理のデバッグの詳細については、「Java クラスのデバッグ」 『Mobile Link - サーバ管理』 (Java の場合)、または「.NET 同期論理のデバッグ」 『Mobile Link - サーバ管理』 (.NET の場合) を参照してください。

同期スクリプトの作成方法の詳細については、「同期スクリプトの作成」 『Mobile Link - サーバ管理』と「同期イベント」 『Mobile Link - サーバ管理』を参照してください。

第 9 章

チュートリアル：ダイレクト・ロー・ハンドリングの使用

目次

ダイレクト・ロー・ハンドリングのチュートリアルの概要

このチュートリアルでは、Mobile Link ダイレクト・ロー・ハンドリングを実装して、サポートされている統合データ・ソース以外のデータ・ソースを使用できるようにする方法について説明します。

ダイレクト・ロー・ハンドリングを使用して、リモート・データと中央のデータ・ソース、アプリケーション、または Web サービスとの通信ができます。

このチュートリアルでは、Java 用と .NET 用の Mobile Link サーバ API を使用して簡単なダイレクト・ロー・ハンドリングを行う方法を示します。ここでは、クライアントの RemoteOrders テーブルを統合データベースと同期し、OrderComments テーブル用に特別なダイレクト・ロー・ハンドリング処理を追加します。

RemoteOrders テーブルには簡単な同期を設定します。

必要なソフトウェア

- SQL Anywhere 10.0.0
- Java ソフトウェア開発キットまたは Microsoft .NET Framework

前提知識と経験

次の知識と経験が必要です。

- Java または .NET の知識
- Mobile Link イベント・スクリプトと Mobile Link 同期の基礎知識

目的

次の項目について、知識と経験を得ることができます。

- Java 用と .NET 用の Mobile Link サーバ API
- Mobile Link ダイレクト・ロー・ハンドリング用のメソッドの作成

関連項目

Mobile Link 同期の詳細については、「Mobile Link 同期について」 3 ページを参照してください。

同期方法の詳細については、「同期の方法」 『Mobile Link - サーバ管理』を参照してください。

ダイレクト・ロー・ハンドリングの詳細については、「ダイレクト・ロー・ハンドリング」 『Mobile Link - サーバ管理』を参照してください。

レッスン 1：Mobile Link 統合データベースの設定

統合データベースには、SQL Anywhere、Adaptive Server Enterprise、Oracle、Microsoft SQL Server、または IBM DB2 の Mobile Link 統合データベースを使用できます。Mobile Link 統合データベースはデータの中央レポジトリであり、同期処理の管理に使用する Mobile Link のシステム・テーブルとストアド・プロシージャが含まれます。ダイレクト・ロー・ハンドリングでは、統合データベース以外のデータ・ソースと同期しますが、Mobile Link サーバが使用する情報を保持するために統合データベースも必要です。

このレッスンでは、次の作業を行います。

- データベースを作成し、ODBC データ・ソースを定義します。
- 同期するデータ・テーブルをリモート・クライアントに追加します。
- Mobile Link のシステム・テーブルとストアド・プロシージャをインストールします。

注意
Mobile Link 統合データベースを Mobile Link システム・オブジェクトと DSN を使用して設定済みの場合は、このレッスンは省略できます。

統合データベースの作成

このチュートリアルでは、Sybase Central の [データベース作成] ウィザードを使用して SQL Anywhere データベースを作成します。

♦ SQL Anywhere RDBMS を作成するには、次の手順に従います。

1. Sybase Central を起動します。

   [スタート] – [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択します。

   Sybase Central が表示されます。

2. Sybase Central から、[ツール] – [SQL Anywhere 10] – [データベースの作成] を選択します。

   [データベース作成] ウィザードが表示されます。

3. [次へ] をクリックします。

4. [このコンピュータにデータベースを作成] をデフォルトのままにし、[次へ] をクリックします。

5. データベースのファイル名とパスを入力します。たとえば、次のように入力します。

   c:¥MLdirect¥MLconsolidated.db

6. ウィザードの指示に従って残りの操作を進めます。次の設定を除き、デフォルト値をそのまま使用してください。

   - [jConnect メタ情報サポートをインストール] チェックボックスをオフにします。

- [最終切断後にデータベースを停止] チェックボックスをオフにします。

7. [完了] をクリックします。

   MLconsolidated というデータベースが Sybase Central に表示されます。

統合データベース用の ODBC データ・ソースの定義

SQL Anywhere 10 ドライバを使用して、MLconsolidated データベース用の ODBC データ・ソースを定義します。

♦ 統合データベース用の ODBC データ・ソースを定義するには、次の手順に従います。

1. ODBC アドミニストレータを起動します。

   Sybase Central の [ツール] メニューから、[SQL Anywhere 10] – [ODBC アドミニストレータを開く] を選択します。

   [ODBC データ ソース アドミニストレータ] が表示されます。

2. [ユーザー DSN] タブで [追加] をクリックします。

   [データ ソースの新規作成] ダイアログが表示されます。

3. [SQL Anywhere 10] を選択して、[完了] をクリックします。

   [SQL Anywhere 10 の ODBC 設定] ダイアログが表示されます。

4. [ODBC] タブで、データ・ソース名 **mobilink_db** を入力します。[ログイン] タブで、ユーザ ID として **DBA**、パスワードとして **sql** を入力します。[データベース] タブで、[サーバ名] に **MLconsolidated** と入力し、[データベース・ファイル] に *c:¥MLdirect¥MLconsolidated.db* と入力します。

5. [OK] をクリックしてデータ・ソースを定義し、もう一度 [OK] をクリックして ODBC アドミニストレータを閉じます。

同期用テーブルの作成

この手順では、Mobile Link 統合データベースに RemoteOrders テーブルを作成します。RemoteOrders テーブルには次のカラムが含まれます。

カラム	説明
order_id	注文のユニークな識別子です。
product_id	製品のユニークな識別子です。
quantity	品目の販売数です。
order_status	注文のステータスです。
last_modified	ローが最後に変更された日です。このカラムはタイムスタンプベースのダウンロードに使用します。このダウンロード方法は、効率的な同期のためにローをフィルタする一般的な方法です。

♦ **RemoteOrders テーブルを作成するには、次の手順に従います。**

1. Interactive SQL を使用してデータベースに接続します。

   Interactive SQL は、Sybase Central またはコマンド・プロンプトから起動できます。

   - Sybase Central から Interactive SQL を起動するには、データベース **MLconsolidated - DBA** をクリックします。Sybase Central で [ファイル] – [Interactive SQL を開く] を選択します。
   - コマンド・プロンプトで Interactive SQL を起動するには、次のコマンドを入力します。

     ```
     dbisql -c "dsn=mobilink_db"
     ```

2. Interactive SQL で次のコマンドを実行して RemoteOrders テーブルを作成します。

   ```
   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 によって、統合データベースに RemoteOrders テーブルが作成されます。

   Interactive SQL はこの後の手順でも使用するので、接続したままにします。

Mobile Link 設定スクリプトの実行

SQL Anywhere 10 インストール環境の *MobiLink/setup* サブディレクトリに、サポートされている各統合データベース (SQL Anywhere、Adaptive Server Enterprise、Oracle、Microsoft SQL Server、IBM DB2) の設定スクリプトがあります。

この手順では、SQL Anywhere 統合データベースを設定します。設定するには、*syncsa.sql* 設定スクリプトを使用します。*syncsa.sql* を実行すると、前に **ml_** が付いた一連のシステム・テーブルとストアド・プロシージャが作成されます。これらのテーブルとストアド・プロシージャは、同期処理中に Mobile Link サーバによって使用されます。

♦ **Mobile Link のシステム・テーブルをインストールするには、次の手順に従います。**

1. 統合データベースに接続していない場合は、Interactive SQL で接続します。

   コマンド・プロンプトで次のコマンドを入力します。

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. Interactive SQL で次のコマンドを実行して Mobile Link のシステム・テーブルとストアド・プロシージャを作成します。*c:¥Program Files¥SQL Anywhere 10¥* は、SQL Anywhere 10 インストール環境のロケーションに置き換えてください。

   ```
   read "c:¥Program Files¥SQL Anywhere 10¥MobiLink¥setup¥syncsa.sql"
   ```

   Interactive SQL によって *syncsa.sql* が統合データベースに適用されます。

Interactive SQL は次のレッスンでも使用するので、接続したままにします。

詳細情報

SQL Anywhere データベースの作成の詳細については、「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバ - データベース管理』を参照してください。

テーブルの作成については、「CREATE TABLE 文」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

Mobile Link 統合データベースの設定については、「Mobile Link 統合データベース」『Mobile Link - サーバ管理』を参照してください。

レッスン 2：同期スクリプトの追加

このレッスンでは、SQL ロー・ハンドリングとダイレクト・ロー・ハンドリング用のスクリプトを統合データベースに追加します。

SQL ロー・ハンドリング

SQL ロー・ハンドリングを使用すると、リモート・データを、Mobile Link 統合データベース内のテーブルと同期できます。SQL ベースのスクリプトでは、次の情報を定義します。

- Mobile Link クライアントからアップロードするデータを統合データベースに適用する方法。
- 統合データベースからダウンロードするデータ。

このレッスンでは、次の SQL ベースのアップロード・イベントとダウンロード・イベント用の同期スクリプトを作成します。

- **upload_insert**　リモート・クライアント・データベースに挿入された新しい注文を統合データベースに適用する方法を定義します。
- **download_cursor**　Mobile Link 統合データベースで更新された注文のうち、リモート・クライアントにダウンロードするものを定義します。

この手順では、ストアド・プロシージャを使用して、Mobile Link 統合データベースに同期スクリプト情報を追加します。

♦ SQL ベースのスクリプトを Mobile Link のシステム・テーブルに追加するには、次の手順に従います。

1. 統合データベースに接続していない場合は、Interactive SQL で接続します。

   コマンド・プロンプトで次のコマンドを入力します。

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. ml_add_table_script ストアド・プロシージャを使用して、upload_insert と download_cursor の各イベント用の SQL ベースのテーブル・スクリプトを追加します。

   Interactive SQL で次のコマンドを実行します。upload_insert のスクリプトでは、アップロードされた order_id、product_id、quantity、order_status を Mobile Link 統合データベースに挿入します。download_cursor のスクリプトでは、タイムスタンプベースのフィルタを使用して、更新されたローをリモート・クライアントにダウンロードします。

   ```
   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
   ```

ダイレクト・ロー・ハンドリングの処理

このチュートリアルでは、ダイレクト・ロー・ハンドリングを使用して特別な処理を SQL ベースの同期システムに追加します。この手順では、handle_UploadData、handle_DownloadData、end_download の各イベントに対応するメソッド名を登録します。独自の Java または .NET クラスを「レッスン 3：ダイレクト・ロー・ハンドリングを処理する Java または .NET の論理の記述」 144 ページで作成します。

♦ Mobile Link のシステム・テーブルにダイレクト・ロー・ハンドリングの情報を追加するには、次の手順に従います。

1. Interactive SQL の統合データベースに接続します。

   コマンド・プロンプトで次のコマンドを入力します。

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. end_download イベント用の Java または .NET メソッドを登録します。

   このメソッドを使用して、end_download 接続イベントが起動したときに入出力リソースを解放します。

   Java の場合は、Interactive SQL で次のコマンドを実行します。

   ```
   CALL ml_add_java_connection_script( 'default',
     'end_download',
       'MobiLinkOrders.EndDownload' );
   ```

   .NET の場合は、Interactive SQL で次のコマンドを実行します。

   ```
   CALL ml_add_dnet_connection_script( 'default',
     'end_download',
       'MobiLinkOrders.EndDownload' );
   ```

   Interactive SQL によって、ユーザ定義の EndDownload メソッドが end_download イベント用に登録されます。

3. handle_UploadData と handle_DownloadData の各同期イベント用の Java または .NET メソッドを登録します。

   Java の場合は、Interactive SQL で次のコマンドを実行します。

   ```
   CALL ml_add_java_connection_script( 'default',
     'handle_UploadData',
       'MobiLinkOrders.GetUpload' );

   CALL ml_add_java_connection_script( 'default',
     'handle_DownloadData',
       'MobiLinkOrders.SetDownload' );

   commit
   ```

   .NET の場合は、Interactive SQL で次のコマンドを実行します。

   ```
   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 によって、ユーザ定義の GetUpload と SetDownload の各メソッドが、それぞれ handle_UploadData と handle_DownloadData の各イベント用に登録されます。

詳細情報

SQL ベースのイベントを使用したリモート・クライアントから Mobile Link 統合データベースへのデータのアップロードについては、次の項を参照してください。

- 「ローをアップロードするスクリプトの作成」 『Mobile Link - サーバ管理』
- 「upload_insert テーブル・イベント」 『Mobile Link - サーバ管理』
- 「upload_update テーブル・イベント」 『Mobile Link - サーバ管理』
- 「upload_delete テーブル・イベント」 『Mobile Link - サーバ管理』

統合データベース以外のデータ・ソースへのデータのアップロードについては、「ダイレクト・アップロードの処理」 『Mobile Link - サーバ管理』を参照してください。

SQL ベースのイベントを使用した Mobile Link 統合データベースからのデータのダウンロードについては、次の項を参照してください。

- 「ローをダウンロードするスクリプトの作成」 『Mobile Link - サーバ管理』
- 「download_cursor テーブル・イベント」 『Mobile Link - サーバ管理』
- 「download_delete_cursor テーブル・イベント」 『Mobile Link - サーバ管理』

統合データベース以外のデータ・ソースへのデータのダウンロードについては、「ダイレクト・ダウンロードの設定」 『Mobile Link - サーバ管理』を参照してください。

同期イベントの順序については、「Mobile Link イベントの概要」 『Mobile Link - サーバ管理』を参照してください。

ダウンロードをフィルタする同期の方法については、「タイムスタンプベースのダウンロード」 『Mobile Link - サーバ管理』と「リモート・データベース間でローを分割する」 『Mobile Link - サーバ管理』を参照してください。

スクリプトの管理については、「スクリプトの追加と削除」 『Mobile Link - サーバ管理』を参照してください。

ダイレクト・ロー・ハンドリングについては、「ダイレクト・ロー・ハンドリング」 『Mobile Link - サーバ管理』を参照してください。

レッスン 3：ダイレクト・ロー・ハンドリングを処理する Java または .NET の論理の記述

このレッスンでは、ダイレクト・ロー・ハンドリングを使用して、クライアント・データベース内の OrderComments テーブルのローを処理します。ダイレクト・ロー・ハンドリング用に次のメソッドを追加します。

- **GetUpload**　このメソッドは handle_UploadData イベントに使用します。GetUpload では、アップロードされたコメントを *orderComments.txt* というファイルに書き込みます。
- **SetDownload**　このメソッドは handle_DownloadData イベントに使用します。SetDownload では、*orderResponses.txt* ファイルを使用してリモート・クライアントに応答をダウンロードします。

また、end_download イベントを処理する EndDownload メソッドも追加します。

次の手順では、処理用メソッドを含む Java または .NET のクラスを作成する方法を示します。完全なリストについては、「MobiLinkOrders の全リスト (Java)」 150 ページまたは「MobiLinkOrders の全リスト (.NET)」 152 ページを参照してください。

♦ ダイレクト・ロー・ハンドリング用の Java または .NET のクラスを作成するには、次の手順に従います。

1. Java または .NET を使用して、MobiLinkOrders というクラスを作成します。

   Java の場合は、テキスト・エディタまたは開発環境で次のコードを入力します。

   ```
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   public class  MobiLinkOrders{

   // to do...
   ```

   .NET の場合は、次のコードを入力します。

   ```
   using iAnywhere.MobiLink.Script;
   using System.IO;
   using System.Data;
   using System.Text;

   public class MobiLinkOrders

   // to do...
   ```

2. クラス・レベルの DBConnectionContext インスタンスを宣言します。

   Java の場合：

   ```
   // class level DBConnectionContext
      DBConnectionContext _cc;
   ```

.NET の場合：

```
// class level DBConnectionContext
   private DBConnectionContext _cc = null;
```

Mobile Link サーバによって DBConnectionContext のインスタンスがクラス・コンストラクタに渡されます。DBConnectionContext には、Mobile Link 統合データベースとの現在の接続に関する情報がカプセル化されます。

3. クラス・コンストラクタを作成します。

   クラス・コンストラクタが、クラスレベルの DBConnectionContext インスタンスを設定します。

   Java の場合：

```
public MobiLinkOrders( DBConnectionContext cc )
{
 // set your class-level DBConnectionContext
 _cc = cc;
}
```

   .NET の場合：

```
public MobiLinkOrders( DBConnectionContext cc )
{
   _cc = cc;
}
```

4. ファイル入出力に使用するオブジェクトを宣言します。

   Java の場合、java.io.FileWriter と java.io.BufferedReader を宣言します。

```
// java objects for file i/o
   FileWriter my_writer;
   BufferedReader my_reader;
```

   .NET の場合、Stream Writer と Stream Reader を宣言します。

```
// instances for file I/O
private static StreamWriter   my_writer = null;
private static StreamReader   my_reader = null;
```

5. EndDownload メソッドを作成します。

   このメソッドでは、end_download 接続イベントを処理し、またリソースを解放できます。

   Java の場合：

```
public void EndDownload() throws IOException
{
  // free i/o resources
    if (my_reader!=null) my_reader.close();
    if (my_writer!=null) my_writer.close();
}
```

   .NET の場合：

```
public void EndDownload()
{
```



```
if( my_writer != null ) {
    my_writer.Close();
    my_writer = null;
}
```

6. GetUpload メソッドを作成します。

   GetUpload メソッドでは、OrderComments テーブルを表す UploadedTableData クラス・インスタンスを取得します。OrderComments テーブルには、遠隔地の営業部員による特別なコメントが含まれます。このテーブルは「レッスン 5：Mobile Link クライアントの設定」 157 ページで作成します。UploadedTableData の getInserts メソッドでは、注文に対する新しいコメントの結果セットを返します。writeOrderComment メソッドでは、結果セット内の各ローをテキスト・ファイルに書き出します。

   Java の場合：

```
//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();

  }
```

   .NET の場合：

```
// 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 on the remote.
        // in this example you just use the known column order to determine the column
        // indexes

        // only process this insert of 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. SetDownload メソッドを作成します。

    a. OrderComments テーブルを表すクラス・インスタンスを取得します。

    DBConnectionContext の getDownloadData メソッドを使用して DownloadData のインスタンスを取得します。DownloadData の getDownloadTableByName メソッドを使用して、OrderComments テーブルの DownloadTableData インスタンスを返します。

    Java の場合：

    ```
    DownloadData download_d = _cc.getDownloadData();
    DownloadTableData download_td = download_d.getDownloadTableByName
    ( "OrderComments" );
    ```

    .NET の場合：

    ```
    DownloadTableData comments_for_download =
        _cc.GetDownloadData().GetDownloadTableByName( "OrderComments" );
    ```

    注意：
    このテーブルは、「レッスン 5：Mobile Link クライアントの設定」 157 ページでリモート・データベースに作成します。

    b. 準備文または IDBCommand を取得します。これを使用すると、ダウンロードに挿入操作や更新操作を追加できます。

Java の場合は、DownloadTableData の getUpsertPreparedStatement メソッドを使用して java.sql.PreparedStatement のインスタンスを返します。

```
PreparedStatement update_ps = download_td.getUpsertPreparedStatement();
```

.NET の場合、DownloadTableData の GetUpsertCommand メソッドを使用します。

```
// you will 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. 応答をリモート・クライアントに返します。

orderResponses.txt というテキスト・ファイルを *c:¥MLdirect* に作成します。このファイルには、コメントに対する応答が含まれています。*orderResponses.txt* には、たとえば次のエントリが含まれる場合があります。これには、タブで区切った comment_id、order_id、order_comment の値が含まれます。

```
...
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. 各ローのダウンロード・データを設定します。

Java の場合、次の例では、*orderResponses.txt* を参照して、Mobile Link ダウンロードにデータを追加しています。

Java の場合：

```
//   a BufferedReader for writing out 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();
         }
```

.NET の場合：

```
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. ダウンロードに挿入操作または更新操作を追加する準備文を終了します。

Java の場合：

```
update_ps.close();
```

.NET の場合、IDBCommand を閉じる必要はありません。これは、ダウンロードの終わりに Mobile Link によって破棄されます。

8. クラス・ファイルをコンパイルします

- Java または .NET のソース・ファイルが含まれるディレクトリに移動します。
- Java または .NET 用の Mobile Link サーバ API ライブラリを参照して MobiLinkOrders をコンパイルします。

Java の場合は、SQL Anywhere インストール環境の *java* サブディレクトリにある *mlscript.jar* を参照する必要があります。次のコマンド・ラインを入力して Java クラスをコンパイルします。*c:¥Program Files¥SQL Anywhere 10¥* は SQL Anywhere 10 の実際のディレクトリに置き換えてください。

```
javac -classpath "c:¥Program Files¥SQL Anywhere 10¥java¥mlscript.jar" MobiLinkOrders.java
```

.NET の場合は、次のコマンドを入力します。

```
csc /out:MobiLinkServerCode.dll /target:library /reference:"c:¥Program Files¥SQL Anywhere 10
¥Assembly¥v1¥iAnywhere.MobiLink.Script.dll" MobiLinkOrders.cs
```

詳細情報

クラス・コンストラクタと DBConnectionContext の詳細については、以下を参照してください。

- .NET 同期論理：「コンストラクタ」『Mobile Link - サーバ管理』
- Java 同期論理：「コンストラクタ」『Mobile Link - サーバ管理』

Java 同期論理の詳細については、「Java による同期スクリプトの作成」『Mobile Link - サーバ管理』を参照してください。

.NET 同期論理の詳細については、「.NET での同期スクリプトの作成」『Mobile Link - サーバ管理』を参照してください。

ダイレクト・ロー・ハンドリングの詳細については、「ダイレクト・ロー・ハンドリング」『Mobile Link - サーバ管理』を参照してください。

MobiLinkOrders の全リスト (Java)

Java でダイレクト・ロー・ハンドリングを行う場合の MobiLinkOrders の全リストを次に示します。手順を追った説明については、「レッスン 3：ダイレクト・ロー・ハンドリングを処理する Java または .NET の論理の記述」 144 ページを参照してください。

```
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 writing out 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();
    }

}
```

MobiLinkOrders の全リスト (.NET)

.NET オブジェクトベースのアップロードとダウンロードを行う場合の MobiLinkOrders の全リストは次のとおりです。手順を追った説明については、「レッスン 3：ダイレクト・ロー・ハンドリングを処理する Java または .NET の論理の記述」 144 ページを参照してください。

```
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:¥¥mlobjbased¥¥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 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 on the remote.
```

```
    // in this example you just use the known column order to determine the column
    // indexes

    // only process this insert of 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:¥¥mlobjbased¥¥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( "Nothing to do in SetDownload() there is no file to read." );
    return;
}
DownloadTableData comments_for_download =
    _cc.GetDownloadData().GetDownloadTableByName( "OrderComments" );

// you will 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;
```



```
        }
        }
    }
```

レッスン 4：Mobile Link サーバの起動

このレッスンでは、Mobile Link サーバを起動します。-c オプションを使用して Mobile Link サーバ (mlsrv10) を起動して統合データベースに接続し、-sl java オプションまたは -sl dnet オプションを使用してそれぞれ Java または .NET のクラスをロードします。

♦ ダイレクト・ロー・ハンドリング用に Mobile Link サーバを起動するには、次の手順に従います。

1. 統合データベースに接続し、mlsrv10 のコマンド・ラインで .NET または Java クラスをロードします。

   Java の場合は、次のコマンドを入力します。*c:¥MLdirect* は、Java ソース・ファイルがある実際のディレクトリに置き換えてください。

   ```
   mlsrv10 -c "dsn=mobilink_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl java (-cp c:¥MLdirect)
   ```

   .NET の場合：

   ```
   mlsrv10 -c "dsn=mobilink_db;uid=DBA;pwd=sql" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl dnet (-
   Dautoloadpath=c:¥MLdirect)
   ```

   Mobile Link サーバ・ウィンドウが表示されます。

   このチュートリアルで使用している Mobile Link サーバの各オプションの説明を次に示します。オプション -o、-v、-dl は、デバッグとトラブルシューティングの情報を提供します。これらのロギング・オプションは、開発環境での使用に適しています。パフォーマンス上の理由から、一般的に -v と -dl は運用環境では使用しません。

オプション	説明
-c	続いて接続文字列を指定します。
-o	メッセージ・ログ・ファイル *serverOut.txt* を指定します。
-v+	-v オプションは、ログを取る情報を指定します。-v+ を使用して、最大冗長ロギングをオンに設定します。
-dl	画面にすべてのログ・メッセージを表示します。
-zu+	自動的に新しいユーザを追加します。
-x	Mobile Link クライアントの通信プロトコルとパラメータを設定します。
-sl java	クラス・ファイルを検索する一連のディレクトリを指定し、またサーバ起動時に Java 仮想マシンをロードします。
-sl dnet	.NET アセンブリのロケーションを指定し、またサーバ起動時に CLR をロードします。

詳細情報

Mobile Link サーバ・オプションの完全なリストについては、「Mobile Link サーバのオプション」『Mobile Link - サーバ管理』を参照してください。

Java と .NET のクラスのロードの詳細については、それぞれ「-sl java オプション」『Mobile Link - サーバ管理』と「-sl dnet オプション」『Mobile Link - サーバ管理』を参照してください。

レッスン 5：Mobile Link クライアントの設定

このチュートリアルでは、SQL Anywhere データベースを統合データベースと Mobile Link クライアントに使用します。また、このチュートリアルの目的上、Mobile Link クライアント、統合データベース、および Mobile Link サーバは同じコンピュータに置きます。

Mobile Link クライアント・データベースを設定するには、RemoteOrders と OrderComments の各テーブルを作成します。RemoteOrders テーブルは、統合データベースの RemoteOrders テーブルに対応します。Mobile Link サーバでは、SQL ベースのスクリプトを使用してリモート注文が同期されます。OrderComments テーブルは、クライアント・データベースだけで使用されます。Mobile Link サーバでは、特別なイベントを使用して OrderComments テーブルが処理されます。

また、クライアント・データベースに同期ユーザ、パブリケーション、サブスクリプションも作成します。

♦ **Mobile Link クライアント・データベースを設定するには、次の手順に従います。**

1. Mobile Link クライアント・データベースを作成します。

   このレッスンでは、dbinit コマンド・ライン・ユーティリティを使用して SQL Anywhere データベースを作成します。

   - SQL Anywhere データベースを作成するには、コマンド・プロンプトで次のコマンドを入力します。

     ```
     dbinit -i -k remote1
     ```

     -i オプションと -k オプションは、それぞれ Sybase jConnect のサポートと Watcom SQL の互換ビューを省略するように dbinit に指定しています。

   - データベース・エンジンを起動するには、次のように入力します。

     ```
     dbeng10 remote1
     ```

2. Interactive SQL を使用して Mobile Link クライアント・データベースに接続します。

   コマンド・プロンプトで次のように入力します。

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

3. RemoteOrders テーブルを作成します。

   Interactive SQL で次のコマンドを実行します。

   ```
   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. Interactive SQL で次のコマンドを実行して OrderComments テーブルを作成します。

   ```
   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. Mobile Link 同期ユーザ、パブリケーション、サブスクリプションを作成します。

```
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：**
> Mobile Link サーバに接続する方法は、CREATE SYNCHRONIZATION SUBSCRIPTION 文の TYPE 句と ADDRESS 句を使用して指定します。

パブリケーションを使用して、同期するデータを指定できます。この例では、entire RemoteOrders と OrderComments の各テーブルを指定します。

詳細情報

SQL Anywhere データベースの作成については、「初期化ユーティリティ (dbinit)」『SQL Anywhere サーバ - データベース管理』を参照してください。

Mobile Link クライアントについては、「Mobile Link クライアントの紹介」『Mobile Link - クライアント管理』を参照してください。

クライアントでの Mobile Link オブジェクトの作成については、次の項を参照してください。

- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE PUBLICATION 文 [Mobile Link] [SQL Remote]」『SQL Anywhere サーバ - SQL リファレンス』
- 「CREATE SYNCHRONIZATION SUBSCRIPTION 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』

レッスン 6：同期

dbmlsync ユーティリティを使用して、SQL Anywhere リモート・データベースの Mobile Link 同期を開始します。dbmlsync を起動する前に、注文データとコメントをリモート・データベースに追加します。

♦ **リモート・データを設定するには、次の手順に従います (クライアント側)。**

1. Interactive SQL で Mobile Link クライアント・データベースに接続します。

   コマンド・プロンプトで次のコマンドを入力します。

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

2. クライアント・データベース内の RemoteOrders テーブルに注文を追加します。

   Interactive SQL で次のコマンドを実行します。

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. クライアント・データベース内の OrderComments テーブルにコメントを追加します。

   Interactive SQL で次のコマンドを実行します。

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. これまでの変更内容をコミットします。

   Interactive SQL で次のコードを実行します。

   ```
   COMMIT;
   ```

♦ **同期クライアントを起動するには、次の手順に従います (クライアント側)。**

· コマンド・プロンプトで、次のコマンドを 1 行で入力します。

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

次に、各オプションの説明を示します。

オプション	説明
-c	接続文字列を指定します。
-e scn	SendColumnNames をオンに設定します。これは、カラムを名前で参照する場合にダイレクト・ロー・ハンドリングで必要となります。
-o	メッセージ・ログ・ファイル *rem1.txt* を指定します。
-v+	-v オプションは、ログを取る情報を指定します。-v+ を使用して、最大冗長ロギングをオンに設定します。

Mobile Link 同期クライアントの起動が完了すると、同期が成功したことを示す出力画面が表示されます。SQL ベースの同期によって、クライアントの RemoteOrders テーブル内のローが、統合データベース内の RemoteOrders テーブルに転送されました。

Java または .NET の処理によってコメントが *orderComments.txt* に挿入されました。次の手順では、応答を *orderResponses.txt* に挿入してリモート・データベースにダウンロードします。

♦ ダイレクト・ロー・ハンドリングによるダウンロードを使用してコメントを返すには、次の手順に従います (サーバ側とクライアント側)。

1. 返すコメントを挿入します。この操作はサーバ側で行います。

   次のテキストを *orderResponses.txt* に追加します。エントリはタブ文字で区切ります。行末で、[Enter] キーを押します。

   ```
   1  1  Promotional material shipped
   ```

2. dbmlsync クライアント・ユーティリティを使用して同期を実行します。

   この操作はクライアント側で行います。

   コマンド・プロンプトで、次のコマンドを 1 行で入力します。

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

   Mobile Link クライアント・ユーティリティが表示されます。

   Interactive SQL で、OrderComments テーブルを選択して、ローがダウンロードされたことを確認します。

注意
ダイレクト・ロー・ハンドリングを使用してダウンロードされたローは、mlsrv10 -v+ オプションによっては出力されず、リモートの -v+ オプションによってリモート・ログに出力されます。

詳細情報

dbmlsync の詳細については、「SQL Anywhere クライアント」『Mobile Link - クライアント管理』を参照してください。

クリーンアップ

チュートリアルをコンピュータから削除します。

♦ **チュートリアルを削除するには、次の手順に従います。**

1. Interactive SQL のすべてのインスタンスを閉じます。
2. SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを閉じます。
3. チュートリアルに関連するすべての DSN を削除します。
   - ODBC アドミニストレータを起動します。

     コマンド・プロンプトで次のように入力します。

     ```
     odbcad32
     ```
   - mobilink_db データ・ソースを削除します。
4. 統合データベースとリモート・データベースを削除します。
   - 統合データベースとリモート・データベースが保存されているディレクトリに移動します。
   - *MLconsolidated.db*、*MLconsolidated.log*、*remote1.db*、および *remote1.log* を削除します。

詳細情報

Mobile Link サーバ API の詳細については、次の項を参照してください。

- 「Java による同期スクリプトの作成」『Mobile Link - サーバ管理』
- 「.NET での同期スクリプトの作成」『Mobile Link - サーバ管理』

Mobile Link のダイレクト・ロー・ハンドリングの詳細については、「ダイレクト・ロー・ハンドリング」『Mobile Link - サーバ管理』を参照してください。

索引

記号

A

C

D

G

I

J

L

M

N

O

P

S

し

す

せ

た

ち

つ

て

と

に

は

ひ

ふ

へ

ま

も

り

れ

ろ

