SYBASE iAnywhere

Mobile Link
クイック・スタート

2009 年 2 月

バージョン 11.0.1

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 11 - 紹介』を参照してください。

SQL Anywhere のマニュアルについて

SQL Anywhere の完全なマニュアルは 4 つの形式で提供されており、いずれも同じ情報が含まれています。

- **HTML ヘルプ**　オンライン・ヘルプには、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。

  Microsoft Windows オペレーティング・システムを使用している場合は、オンライン・ヘルプは HTML ヘルプ (CHM) 形式で提供されます。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 11] - [マニュアル] - [オンライン・マニュアル]** を選択します。

  管理ツールのヘルプ機能でも、同じオンライン・マニュアルが使用されます。

- **Eclipse**　UNIX プラットフォームでは、完全なオンライン・ヘルプは Eclipse 形式で提供されます。マニュアルにアクセスするには、SQL Anywhere 11 インストール環境の *bin32* または *bin64* ディレクトリから *sadoc* を実行します。

- **DocCommentXchange**　DocCommentXchange は、SQL Anywhere マニュアルにアクセスし、マニュアルについて議論するためのコミュニティです。

  DocCommentXchange は次の目的に使用できます (現在のところ、日本語はサポートされておりません)。

  ○ マニュアルを表示する

  ○ マニュアルの項目について明確化するために、ユーザによって追加された内容を確認する

  ○ すべてのユーザのために、今後のリリースでマニュアルを改善するための提案や修正を行う

  http://dcx.sybase.com を参照してください。

- **PDF**　SQL Anywhere の完全なマニュアル・セットは、Portable Document Format (PDF) 形式のファイルとして提供されます。内容を表示するには、PDF リーダが必要です。Adobe Reader をダウンロードするには、http://get.adobe.com/reader/ にアクセスしてください。

  Microsoft Windows オペレーティング・システムで PDF 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 11] - [マニュアル] - [オンライン・マニュアル - PDF]** を選択します。

  UNIX オペレーティング・システムで PDF マニュアルにアクセスするには、Web ブラウザを使用して *install-dir/documentation/ja/pdf/index.html* を開きます。

マニュアル・セットに含まれる各マニュアルについて

SQL Anywhere のマニュアルは次の構成になっています。

- **『SQL Anywhere 11 - 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 11 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 11 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 11 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 データベースを実行、管理、構成する方法について説明します。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、管理ユーティリティとオプションについて説明します。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java、PHP、Perl、Python、および Visual Basic や Visual C# などの .NET プログラミング言語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。ADO.NET や ODBC などのさまざまなプログラミング・インタフェースについても説明します。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルでは、システム・プロシージャとカタログ (システム・テーブルとビュー) に関する情報について説明します。また、SQL Anywhere での SQL 言語の実装 (探索条件、構文、データ型、関数) についても説明します。
- **『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。
- **『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 - 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。また、dbmlsync API についても説明します。dbmlsync API を使用すると、同期を C++ または .NET のクライアント・アプリケーションにシームレスに統合できます。
- **『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 - サーバ起動同期』** このマニュアルでは、Mobile Link サーバ起動同期について説明します。この機能により、Mobile Link サーバは同期を開始したり、リモート・デバイス上でアクションを実行することができます。
- **『QAnywhere』** このマニュアルでは、モバイル・クライアント、ワイヤレス・クライアント、デスクトップ・クライアント、およびラップトップ・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、QAnywhere について説明します。
- **『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。

- **『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 - 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・デバイス、モバイル・デバイス、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 Mobile、または Windows を搭載しているハンドヘルド・デバイス、モバイル・デバイス、または埋め込みデバイスの Web ベースのデータベース・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 - .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・デバイス、モバイル・デバイス、または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 J』** このマニュアルでは、Ultra Light J について説明します。Ultra Light J を使用すると、Java をサポートしている環境用のデータベース・アプリケーションを開発し、配備することができます。Ultra Light J は、BlackBerry スマートフォンと Java SE 環境をサポートしており、iAnywhere Ultra Light データベース製品がベースになっています。
- **『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示し、その診断情報を説明します。

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表記規則について説明します。

オペレーティング・システム

SQL Anywhere はさまざまなプラットフォームで稼働します。ほとんどの場合、すべてのプラットフォームで同じように動作しますが、いくつかの相違点や制限事項があります。このような相違点や制限事項は、一般に、基盤となっているオペレーティング・システム (Windows、UNIX など) に由来しており、使用しているプラットフォームの種類 (AIX、Windows Mobile など) またはバージョンに依存していることはほとんどありません。

オペレーティング・システムへの言及を簡素化するために、このマニュアルではサポートされているオペレーティング・システムを次のようにグループ分けして表記します。

- **Windows** Microsoft Windows ファミリを指しています。これには、主にサーバ、デスクトップ・コンピュータ、ラップトップ・コンピュータで使用される Windows Vista や Windows XP、およびモバイル・デバイスで使用される Windows Mobile が含まれます。

  特に記述がないかぎり、マニュアル中に Windows という記述がある場合は、Windows Mobile を含むすべての Windows ベース・プラットフォームを指しています。

- **UNIX**　特に記述がないかぎり、マニュアル中に UNIX という記述がある場合は、Linux および Mac OS X を含むすべての UNIX ベース・プラットフォームを指しています。

ディレクトリとファイル名

ほとんどの場合、ディレクトリ名およびファイル名の参照形式はサポートされているすべてのプラットフォームで似通っており、それぞれの違いはごくわずかです。このような場合は、Windows の表記規則が使用されています。詳細がより複雑な場合は、マニュアルにすべての関連形式が記載されています。

ディレクトリ名とファイル名の表記を簡素化するために使用されている表記規則は次のとおりです。

- **大文字と小文字のディレクトリ名**　Windows と UNIX では、ディレクトリ名およびファイル名には大文字と小文字が含まれている場合があります。ディレクトリやファイルが作成されると、ファイル・システムでは大文字と小文字の区別が維持されます。

  Windows では、ディレクトリおよびファイルを参照するとき、大文字と小文字は**区別されません**。大文字と小文字を混ぜたディレクトリ名およびファイル名は一般的に使用されますが、参照するときはすべて小文字を使用するのが通常です。SQL Anywhere では、*Bin32* や *Documentation* などのディレクトリがインストールされます。

  UNIX では、ディレクトリおよびファイルを参照するとき、大文字と小文字は**区別されます**。大文字と小文字を混ぜたディレクトリ名およびファイル名は一般的に使用されません。ほとんどの場合は、すべて小文字の名前が使用されます。SQL Anywhere では、*bin32* や *documentation* などのディレクトリがインストールされます。

  このマニュアルでは、ディレクトリ名に Windows の形式を使用しています。ほとんどの場合、大文字と小文字が混ざったディレクトリ名をすべて小文字に変換すると、対応する UNIX 用のディレクトリ名になります。

- **各ディレクトリおよびファイル名を区切るスラッシュ**　マニュアルでは、ディレクトリの区切り文字に円記号を使用しています。たとえば、PDF 形式のマニュアルは *install-dir¥Documentation¥ja¥pdf* にあります。これは Windows の形式です。

  UNIX では、円記号をスラッシュに置き換えます。PDF マニュアルは *install-dir/documentation/ja/pdf* にあります。

- **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、*.exe* や *.bat* などの拡張子が付きます。UNIX では、実行ファイルの名前に拡張子は付きません。

  たとえば、Windows でのネットワーク・データベース・サーバは *dbsrv11.exe* です。UNIX では *dbsrv11* です。

- ***install-dir***　インストール・プロセス中に、SQL Anywhere をインストールするロケーションを選択します。このロケーションを参照する環境変数 SQLANY11 が作成されます。このマニュアルでは、そのロケーションを *install-dir* と表します。

  たとえば、マニュアルではファイルを install-dir¥readme.txt のように参照します。これは、Windows では、%SQLANY11%¥readme.txt に対応します。UNIX では、$SQLANY11/readme.txt または ${SQLANY11}/readme.txt に対応します。

install-dir のデフォルト・ロケーションの詳細については、「SQLANY11 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・プロセス中に、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します。このロケーションを参照する環境変数 SQLANYSAMP11 が作成されます。このマニュアルではそのロケーションを *samples-dir* と表します。

  Windows エクスプローラ・ウィンドウで *samples-dir* を開くには、**[スタート] - [プログラム] - [SQL Anywhere 11] - [サンプル・アプリケーションとプロジェクト]** を選択します。

  samples-dir のデフォルト・ロケーションの詳細については、「SQLANYSAMP11 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。

コマンド・プロンプトとコマンド・シェル構文

ほとんどのオペレーティング・システムには、コマンド・シェルまたはコマンド・プロンプトを使用してコマンドおよびパラメータを入力する方法が、1 つ以上あります。Windows のコマンド・プロンプトには、コマンド・プロンプト (DOS プロンプト) および 4NT があります。UNIX のコマンド・シェルには、Korn シェルおよび bash があります。各シェルには、単純コマンドからの拡張機能が含まれています。拡張機能は、特殊文字を指定することで起動されます。特殊文字および機能は、シェルによって異なります。これらの特殊文字を誤って使用すると、多くの場合、構文エラーや予期しない動作が発生します。

このマニュアルでは、一般的な形式のコマンド・ラインの例を示します。これらの例に、シェルにとって特別な意味を持つ文字が含まれている場合、その特定のシェル用にコマンドを変更することが必要な場合があります。このマニュアルではコマンドの変更について説明しませんが、通常、その文字を含むパラメータを引用符で囲むか、特殊文字の前にエスケープ文字を記述します。

次に、プラットフォームによって異なるコマンド・ライン構文の例を示します。

- **カッコと中カッコ** 一部のコマンド・ライン・オプションは、詳細な値を含むリストを指定できるパラメータを要求します。リストは通常、カッコまたは中カッコで囲まれています。このマニュアルでは、カッコを使用します。次に例を示します。

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

  カッコによって構文エラーになる場合は、代わりに中カッコを使用します。

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

  どちらの形式でも構文エラーになる場合は、シェルの要求に従ってパラメータ全体を引用符で囲む必要があります。

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

- **引用符** パラメータの値として引用符を指定する必要がある場合、その引用符はパラメータを囲むために使用される通常の引用符と競合する可能性があります。たとえば、値に二重引用符を含む暗号化キーを指定するには、キーを引用符で囲み、パラメータ内の引用符をエスケープします。

  ```
  -ek "my ¥"secret¥" key"
  ```


多くのシェルでは、キーの値は my "secret" key のようになります。

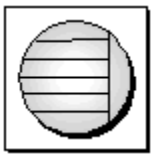
- **環境変数**　マニュアルでは、環境変数設定が引用されます。Windows のシェルでは、環境変数は構文 *%ENVVAR%* を使用して指定されます。UNIX のシェルでは、環境変数は構文 *$ENVVAR* または *${ENVVAR}* を使用して指定されます。

グラフィック・アイコン

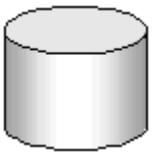
このマニュアルでは、次のアイコンを使用します。

- 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。



- データベース。ハイレベルの図では、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。

- プログラミング・インタフェース。

ドキュメンテーション・チームへのお問い合わせ

このヘルプに関するご意見、ご提案、フィードバックをお寄せください。

SQL Anywhere ドキュメンテーション・チームへのご意見やご提案は、弊社までご連絡ください。頂戴したご意見はマニュアルの向上に役立たせていただきます。ぜひとも、ご意見をお寄せください。

DocCommentXchange

DocCommentXchange を使用して、ヘルプ・トピックに関するご意見を直接お寄せいただくこともできます。DocCommentXchange (DCX) は、SQL Anywhere マニュアルにアクセスしたり、マニュアルについて議論するためのコミュニティです。DocCommentXchange は次の目的に使用できます (現在のところ、日本語はサポートされておりません)。

- マニュアルを表示する
- マニュアルの項目について明確化するために、ユーザによって追加された内容を確認する
- すべてのユーザのために、今後のリリースでマニュアルを改善するための提案や修正を行う

http://dcx.sybase.com を参照してください。

詳細情報の検索／テクニカル・サポートの依頼

詳しい情報やリソースについては、iAnywhere デベロッパー・コミュニティ (http://www.iAnywhere.jp/developers/index.html) 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。バージョンおよびビルド番号を調べるには、コマンド **dbeng11 -v** を実行します。

ニュースグループは、ニュース・サーバ *forums.sybase.com* にあります。

以下のニュースグループがあります。

- ianywhere.public.japanese.general

Web 開発に関する問題については、http://groups.google.com/group/sql-anywhere-web-development を参照してください。

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。

Mobile Link テクノロジの概要

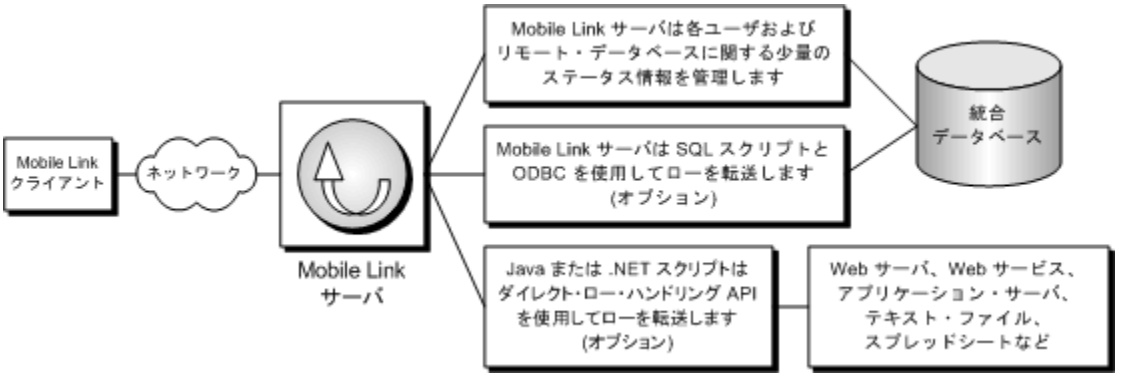
この項では、Mobile Link 同期テクノロジの概要と、Mobile Link 同期テクノロジを使用して 2 つ以上のデータベース間でデータをレプリケートする方法について説明します。

Mobile Link 同期の概要

目次

Mobile Link アプリケーションの各部分

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。1 つの 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」 『Mobile Link - クライアント管理』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。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・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通常、Mobile Link 同期に必要なシステム・テーブルとプロシージャ、および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。「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 サーバは通常、統合データベース内のシステム・テーブルの同期情報を管理します。情報の管理は、ODBC 接続を介して行われます。

  また、ステータス情報を別のデータベースに格納することもできます。「Mobile Link システム・データベース」 『Mobile Link - サーバ管理』を参照してください。
- **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。「サーバ側の同期論理の作成オプション」 15 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**　Mobile Link のダイレクト・ロー・ハンドリングを使用して、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。「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。
- **同期スクリプト**　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』
  ○「サーバ側の同期論理の作成オプション」 15 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複数のメカニズムがあります。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 認定のセキュリティ・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**　アップロード専用、ダウンロード専用、および双方向の同期を実行するよう選択できます。
- **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、データの更新をリモート・データベースにプッシュし、リモート・データベースによってデータが統合データベースにアップロードされるようにできることを意味しています。Mobile Link - サーバ起動同期を参照してください。
- **複数のネットワーク・プロトコル**　同期は TCP/IP、HTTP、HTTPS 経由で実行できます。Palm デバイスは HotSync を使用して同期できます。Windows Mobile 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・データベース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、各種の一般的なデータベース管理システムを使用できます。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、UNIX、Linux、Mac OS X 上で動作します。SQL Anywhere は、Windows、Windows Mobile、UNIX、Linux、Mac OS X 上で動作します。Ultra Light は、Palm または Windows Mobile 上で動作します。「サポートされるプラットフォーム」『SQL Anywhere 11 - 紹介』を参照してください。

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・データベースのいずれかです。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。このウィザードを使用すると、以下で説明している手順の大部分がウィザードにより処理されます。「Mobile Link のモデルの概要」 28 ページを参照してください。

ただし、Mobile Link モデルを使用する場合でも、Mobile Link 同期の処理とコンポーネントは理解しておく必要があります。

Mobile Link アプリケーションの概要

♦ **Mobile Link アプリケーションを作成するには、次の手順に従います。**

1. 統合データベースを設定します。
   - データベースに対して設定スクリプトを実行し、Mobile Link 同期に必要なシステム・オブジェクトを追加します。または、これらのオブジェクトを格納するためのシステム・データベースを別途作成します。

     「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。
2. リモート・データベースを設定します。
   - リモート・データベースとしては SQL Anywhere と Ultra Light のどちらも使用できます。また、両方を組み合わせて使用することもできます。
   - リモート・データベースで Mobile Link ユーザを作成します。「Mobile Link ユーザ」 『Mobile Link - クライアント管理』を参照してください。
   - SQL Anywhere のリモート・データベースでアップロードを特定するには、パブリケーションとサブスクリプションを作成します。「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。

     Ultra Light のリモート・データベースでアップロードを特定するには、パブリケーションを作成します。「Ultra Light のパブリケーション」 『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。
3. アップロードの適用形式を特定するには、サーバ同期論理を作成します。「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。

4. 前回のダウンロード以降に変更されたデータをダウンロードするには、タイムスタンプ・ベースの同期を設定します。「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。
5. Mobile Link サーバを起動します。「Mobile Link サーバ」『Mobile Link - サーバ管理』を参照してください。
6. クライアントの同期を開始します。
   - SQL Anywhere リモート・データベースの詳細については、「同期の開始」『Mobile Link - クライアント管理』を参照してください。
   - Ultra Light リモート・データベースの詳細については、「Ultra Light での同期の設計」『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。

入門情報

- 「Mobile Link 同期の概要」 3 ページ
- 「同期の方法」『Mobile Link - サーバ管理』

チュートリアル

- 「チュートリアル：Mobile Link の概要」 93 ページ
- 「Mobile Link CustDB サンプルの解説」 55 ページ
- 「Ultra Light CustDB サンプル」『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」 77 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」 117 ページ
- 「チュートリアル：Adaptive Server Enterprise 統合データベースと Mobile Link の使用」 139 ページ
- 「チュートリアル：Java 同期論理の使用」 161 ページ
- 「チュートリアル：.NET 同期論理の使用」 173 ページ
- 「チュートリアル：カスタム認証用の .NET と Java の使用」 187 ページ
- 「チュートリアル：ダイレクト・ロー・ハンドリングの概要」 199 ページ
- 「チュートリアル：Microsoft Excel との同期」 227 ページ
- 「チュートリアル：XML との同期」 249 ページ

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* の場所については、「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)。
- Mobile Link のコード・サンプルは、http://www.sybase.com/detail?id=1058600#319 にあります。

Mobile Link アプリケーションの設計

データベース・アプリケーションには、次の 2 つの基本的なアーキテクチャがあります。

- **オンライン・アプリケーション**　オンライン・アプリケーション・ユーザが統合データベースに直接接続してデータを更新します。接続できない場合、ユーザによる作業はできません。
- **随時接続スマート・クライアント・アプリケーション**　各ユーザがローカル・データベースを保有しています。各ユーザは接続状態に関わらず常に自分のデータベース・アプリケーションを使用できます。また、このデータベース・アプリケーションはシステム内の他のデータベースと同期されます。

Mobile Link では随時接続スマート・クライアント・アプリケーションを作成できます。スマート・クライアント・アプリケーションにより、アプリケーションの利便性、効率性、スケーラビリティが大幅に向上します。しかし、アプリケーションの開発に関する新しい問題も発生します。このセクションでは、スマート・クライアント・アプリケーションの開発に関する主な問題について説明します。また、Mobile Link 同期環境でソリューションを実装する方法についても説明します。

必要な同期のみを実行

大部分のアプリケーションでは、リモート・デバイスのデータを一部だけでも更新する場合に毎回統合データベース全体をダウンロードすると、大変なことになります。必要な時間と帯域が膨大なものとなり、システム全体の動作が停止してしまいます。各ユーザにとって必要なアップロードとダウンロードのみを行うようにするための方法は複数あります。

まず、各リモート・データベースには統合データベースのテーブルとカラムのサブセットのみが格納されるようにします。たとえば、地域 A の販売担当者が必要とするテーブルやカラムは地域 B の販売担当者や管理職とは異なる場合があります。

リモート・データベースで作成したテーブルやカラムのうち、同期が必要なものだけを同期対象として指定します。Mobile Link アプリケーションでは、データ型が一致していれば、名前が異なっていてもテーブルやカラムをマッピングすることができます。デフォルトではデータのアップロードとダウンロードの両方が可能ですが、Mobile Link では特定のカラムをアップロード専用またはダウンロード専用にすることもできます。

同期時には、各ユーザに関連するリモート・データベースにのみローをダウンロードするようにします。ダウンロードをリモート・データベース別、ユーザ別、またはその他の基準別に分割して行うこともできます。たとえば、地域 A の販売担当者が地域 A のデータのみを更新する必要がある場合を考えてみます。

更新する必要があるのは変更があるデータのみです。Mobile Link アプリケーションでは、アップロードはトランザクション・ログに基づいて行われるため、デフォルトではリモート・データベースで変更されているデータのみがアップロードされます。ダウンロードも同様に行うには、同期形式をタイムスタンプ・ベースに指定して、データが正常にダウンロードされた日時をシステムが記録するようにします。これにより、データのダウンロードはその日時以降に変更があった場合に限られます。

また、高優先度同期方式の実装が必要になる場合もあります。たとえば、緊急のデータは更新頻度が高くなるようスケジュールし、緊急でないデータは夜間やデバイスがクレードルにある間に

更新されるようスケジュールします。高優先度同期を実装するには、それぞれ異なる時刻に実行するようスケジュールされた複数のパブリケーションを作成します。

この他に、プッシュ同期により、必要に応じてデータを効果的にリモート・デバイスにプッシュダウンすることもできます。たとえば、トラック運送会社の配送係が交通の混乱を知らされた場合に、該当地域に向かっているトラック運転手の更新情報をダウンロードできます。Mobile Link では、これをサーバ起動同期と呼んでいます。

アップロードの競合の処理

倉庫を例にとって考えてみます。各従業員はハンドヘルド・デバイスを保有しており、箱の搬入出時に在庫情報を更新します。最初に 100 個の箱が搬入されています。そのため各従業員のリモート・データベースと統合データベースには 100 と登録されています。デービッドが 20 個の箱を搬出しました。彼は自分のデータベースを更新して同期します。これで彼のデータベースと統合データベースの両方に 80 と登録されます。次にスーザンが 10 個の箱を搬出します。ここでスーザンは自分のデータベースを更新して同期しようとしますが、スーザンのアプリケーションは統合データベースに登録されている箱の個数が 80 ではなく 100 であると想定しています。このため、アップロードの競合が発生します。

この倉庫アプリケーションの例では、この問題を解決するには、「デービッドによる更新値 - (最初の値 - スーザンによる更新値) = 正しい値」となるように、次のような競合解決論理を作成する必要があります。

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

この競合解決論理は倉庫などの在庫ベースのアプリケーションでは有効ですが、すべてのビジネス・アプリケーションで適しているわけではありません。Mobile Link では、次の競合解決論理を定義できます。

- **在庫モデル**　ローを更新してユニット数を修正します。
- **日付**　最新の更新が適用されます (値がデータベースで変更された日付が基準です。値が同期された日付ではありません)。
- **操作者**　たとえば、管理者を常に優先したり、レコードの所有者を常に優先したりします。
- **カスタム**　その他実装が必要なビジネス用論理です。

場合によっては、アップロードの競合が発生しないようにシステムを設計することができます。重複を避けるためにデータがリモートで分割されている場合は、競合を回避できる可能性があります。それでも競合が発生する場合は、競合の検出と解決を行うためのプログラムを作成する必要があります。

ユニークなプライマリ・キー

データをアップロードし、アップロードの競合を検出して、削除されたローを統合データベースで同期するには、データベース・システム内で同期されているすべてのテーブルにユニークなプライマリ・キーが必要です。各ローには、データベース内だけでなく、データベース・システム全体でユニークなプライマリ・キーが必要です。プライマリ・キーは更新できないようにする必要があります。

Mobile Link では、ユニークなプライマリ・キーを設定するための方法が複数あります。方法の 1 つとして、プライマリ・キーのデータ型を GUID に設定することがあげられます。GUID (グ

ローバル・ユニーク識別子) は 16 バイトの 16 進数です。Mobile Link の NEWID 関数を使用すると、新しいローに対して自動的に GUID を作成できます。

また、別の解決方法として、複合キーを使用することがあげられます。Mobile Link では各リモート・データベースにリモート ID と呼ばれるユニークな値が設定されています。リモート ID と通常のプライマリ・キー (順序数など) を組み合わせたものをプライマリ・キーとして使用することができます。

SQL Anywhere ではグローバル・オートインクリメント方式による解決方法もあります。あるカラムをグローバル・オートインクリメントとして宣言すると、ローの追加時に、これまでのプライマリ・キーの最後の値を増分することにより、プライマリ・キーが自動的に作成されます。この解決方法は、統合データベースが SQL Anywhere の場合に最適です。

最後に、リモート・データベースに配布されるプライマリ・キー値のプールを作成することもできます。

同期ソリューションの開発時における各種の決定と同様に、どのプライマリ・キー方式を選択するかは、統合データベースとリモート・データベースに対する制御のレベルによって異なります。多くの場合、リモート・データベースは管理なしで動作できる必要があります。また、統合データベースのスキーマを変更することが困難な場合もあります。さらに、統合データベースとして RDBMS を選択した場合、すべての RDBMS ですべての機能がサポートされているわけではないため、使用できるオプションが限られることがあります。

削除の処理

同期方式に関する別の問題として、統合データベースから削除されたローの処理があげられます。たとえば、統合データベースからあるローを削除したとします。次にデービッドが自分のリモート・データベースを同期すると、この削除データがダウンロードされ、デービッドのデータベースからローが削除されます。しかし統合データベースでこの後どうすればよいでしょうか。実際にはスーザンに対しても削除データのダウンロードが必要になるため、ローを削除できません。

ダウンロード削除を処理する方法は 2 通りあります。1 つ目の方法は、ローが削除されているかどうかを示すステータス・カラムを各テーブルに追加することです。この場合、ローは実際には削除されず、削除するようマークが付けられるだけです。削除するようマーク付けされたローは、後ですべてのリモート・データベースが更新されていることを確認できた時点で消去できます。また、各テーブルのシャドー・テーブルを作成することもできます。シャドー・テーブルには、削除されたローのプライマリ・キーの値が格納されています。ローを削除すると、トリガによりシャドー・テーブルに値が入力されます。シャドー・テーブルの値により、リモート・データベースで削除するローが決定されます。

トランザクション

同期データベース・システムでは、コミットされたデータベース・トランザクションのみが同期されます。さらに、同期するデータが関わっている、コミットされたすべてのトランザクションを同期する必要があります。この処理が正しく行われないと、エラーが発生します。これは Mobile Link でのデフォルトの動作です。

また、統合データベースへの接続の独立性レベルも考慮する必要があります。データの一貫性を確保できる範囲で最高のパフォーマンスを実現できる独立性レベルを使用する必要があります。通常、独立性レベル 0 (READ UNCOMMITTED) は同期には不適切で、データの不整合を引き起こす可能性があります。

デフォルトでは、Mobile Link はアップロードには独立性レベル SQL_TXN_READ_COMMITTED を使用し、可能な場合には、ダウンロードにはスナップショット・アイソレーションを使用します (不可能な場合には、SQL_TXN_READ_COMMITTED を使用します)。スナップショット・アイソレーションは、トランザクションが統合データベースで閉じられるまでダウンロードがブロックされる問題を解消します。ただし、すべての RDBMS がこの機能をサポートしているわけではありません。

夏時間

毎年、夏時間から通常時間への移行時にデータベースの同期に関する問題が発生する可能性があります。秋に時刻が 1 時間戻ると、午前 2 時が午前 1 時になります。午前 1 時と午前 2 時の間に同期を実行しようとすると、同期のタイムスタンプが、たとえば最初の午前 1 時 15 分なのか次の 1 時 15 分なのかわからなくなります。

この問題を解決するには、秋になり時間が移行するときに、1 時間の間シャットダウンするか、統合データベース・サーバを協定世界時 (UTC) にします。

詳細情報

- 「同期の方法」 『Mobile Link - サーバ管理』

Mobile Link アプリケーション開発のためのオプション

Mobile Link では、アプリケーション開発のためのさまざまな方法が用意されています。それぞれの方法を単独で使うことも、組み合わせて使うこともできます。

- **同期モデル作成ウィザード**　このウィザードを使用すると、アプリケーションの開発作業を順を追って簡単に実行できます。まずスキーマがある中央データベースを作成し、その後でリモート・データベースと同期に必要なスクリプトを作成できます。また、このウィザードではダウンロード削除などの処理を行うためのシャドー・テーブルを統合データベース上に作成できます。ウィザードが完了すると、同期モデルがモデル・モードで表示され、カスタマイズできるようになります。**同期モデル展開ウィザード**では、データベースとテーブルの作成、Mobile Link システム・テーブルの更新、Mobile Link ユーティリティを実行するためのスクリプトの作成ができます。

  Mobile Link モデルの展開後にカスタマイズを行う場合、モデル・モードで変更するか、または次のいずれかの方法によりモデル以外で変更します。モデル以外で変更してからモデルを再展開すると、変更は失われます。

  「Mobile Link のモデル」 27 ページを参照してください。

- **Sybase Central 管理モード**　Sybase Central の Mobile Link プラグインには管理モードと呼ばれるセクションがあり、Mobile Link アプリケーションのすべての要素を更新することができます。

  「モデル・モード」 35 ページを参照してください。

- **システム・プロシージャ**　中央データベースが統合データベースとして動作するよう設定すると、Mobile Link 同期で使用するシステム・オブジェクトが作成されます。この中には Mobile Link システム・テーブルも含まれます。ここにはサーバ側の Mobile Link アプリケーションの大部分が格納されます。また、この中には、Mobile Link スクリプトへの Mobile Link システム・テーブルの挿入やリモート・ユーザの登録などの作業を行うためのシステム・プロシージャやユーティリティも含まれます。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 システム・テーブルの直接操作**　上級ユーザの場合は、Mobile Link システム・テーブルのデータの追加、削除、更新を直接行うこともできます。この操作を行うには、Mobile Link の仕組みを詳細に理解している必要があります。

  「Mobile Link サーバのシステム・テーブル」 『Mobile Link - サーバ管理』を参照してください。

サーバ側の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と同期する場合は、通常は 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。

ダイレクト・ロー・ハンドリングでは、handle_UploadData イベントと handle_DownloadData イベントを使用して、データ・ソースと同期します。この操作により、アップロード・ローとダウンロード・ローが直接操作「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**　RDBMS でユーザ定義のストアド・プロシージャを使用できない場合は、Java や .NET でメソッドを作成できます。
- **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、接続または同期の間持続する変数を 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』

同期処理

「同期」とは、Mobile Link クライアントと中央データ・ソースの間で行われるデータ交換処理です。この処理の間、クライアントは Mobile Link 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。

クライアントは同期処理を正常に開始します。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、リモート・データベース上で前回の同期以後に更新、挿入、または削除されたすべてのロー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。

- **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 Mobile Link 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複数回変更されていても、その途中の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。

- **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、Mobile Link 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

いる場合など、通信が低速で遅延時間が長い場合は、プロトコルの冗長性の制限が重要になります。

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。

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。同期イベントが発生すると、Mobile Link はその同期イベントに対応する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。イベント用のスクリプトが定義され、Mobile Link システム・テーブルに格納されている場合は、そのスクリプトが呼び出されます。

Mobile Link スクリプト

イベントに関連するスクリプトが作成されている場合、イベントの発生時に 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 のスクリプトがすべて自動的に作成されます。ただし、デフォルトのスクリプトをカスタマイズしたり、新しいスクリプトを作成することもできます。

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。最初のイベント upload_insert は、upload_insert スクリプトの実行をトリガします。このスクリプトにより、emp_id カラムと emp_name カラムのすべての変更が emp テーブルに挿入されます。upload_delete スクリプトと upload_update スクリプトは、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。

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}

イベント	スクリプトの内容例
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}

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}
```

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』

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。

「サーバ側の同期論理の作成オプション」 15 ページを参照してください。

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、プロジェクトを展開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、ブロッキング・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。ブロッキング・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。

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。

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・タイムスタンプを繰り返し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刻の使用」 『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 回のトランザクションで処理します。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がすべて含まれます。
- **ダウンロード中の障害**　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。

  ブロッキング・ダウンロード確認を使用している場合、Mobile Link サーバで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されます。

  非ブロッキング・ダウンロード確認を使用している場合、ダウンロード・トランザクションはすでにコミットされていますが、nonblocking_download_ack スクリプトと publication_nonblocking_download_ack スクリプトは呼び出されません。

  ダウンロード確認を使用していない場合、ダウンロード中に障害が発生しても、サーバ側には影響はありません。

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、参照整合性に違反しない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 のローの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している場合、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もう一度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

Ultra Light J を除くすべての 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、次のような通知を行います。

- 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。参照整合性を維持するため、削除されるすべてのローで警告が発生します。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警告をトラップします。さらに、削除されたローの数を取得することもできます。

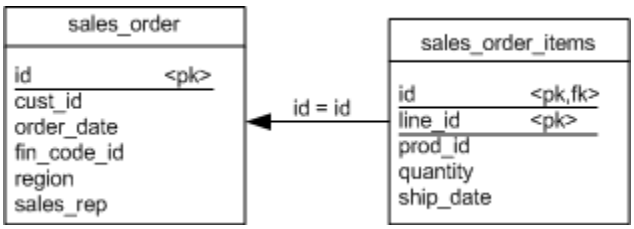
  警告が発生したときに同期を失敗させるには、同期 observer を実装し、observer に (グローバル変数などを使用して) エラー・コールバック関数から信号を送信する必要があります。この場合、同期は observer への次回の呼び出しで失敗します。

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、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 テーブル・スクリプトは必要ありません。
- 同期の効率が向上します。sales_order_items テーブルから削除するローをダウンロード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』

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、データベース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、使用しているデータベース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 データベースの保護」『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、選択した展開オプションを使用して、Mobile Link 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、そのままモデルまたはデータベースをカスタマイズし、再展開することができます。また、Mobile Link のマニュアル全体に記載されている技術を使用して、展開した同期システムを**モデル・**モード以外で変更することもできます。ただし、**モデル・**モード以外で同期システムを変更した場合は、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。

Mobile Link の制限事項

同期モデル作成ウィザードと**モデル・**モードを使用する場合の制限事項の一部を次に示します。

- **モデル以外で行われた変更の再展開はできない**　モデルを展開し、その後モデル以外で変更を行うと、この変更はモデルに保存されません。モデルを開始ポイントとして使用して展開し、すべての変更をモデル以外で行う場合はこれで問題ありません。しかし、モデルを再展開する場合は、**モデル・**モードで変更を行い、変更の保存と再展開ができるようにする必要があります。
- **DB2 メインフレーム**　Mobile Link モデルは、DB2 メインフレーム統合データベースでは動作しません。
- **Mobile Link システム・データベース**　Mobile Link システム・データベースを使用することはできません。
- **複数のアプリケーション**　複数のパブリケーションを作成することはできません。モデルの展開後、CREATE PUBLICATION 文などの非モデル・メソッドを使用して、パブリケーションをさらに追加することはできます。ただし、この方法で追加したパブリケーションをリバース・エンジニアリングでモデルに戻すことはできません。

- **ビュー**　ビュー テーブルのマッピングのために統合データベースのテーブルを選択する場合、ビューを選択することはできません。
- **生成済みリモート・データベース**　**同期モデル作成ウィザード**でリモート・スキーマを作成する場合、新しいリモート・データベースのカラムには統合データベースのカラムの外部キー、インデックス、デフォルト値は格納されません。Ultra Light データベースでは、NCHAR カラムと NVARCHAR カラムがサポートされません。そのため、これらのデータ型のカラムがある統合データベースのテーブルを使用して Ultra Light リモート・データベースの新しいリモート・スキーマを生成することはできません。展開の完了後、リモート・スキーマを作成または変更して、その後モデルのスキーマを変更することができます。
- **計算カラム**　計算カラムがある統合データベースのテーブルを同期する場合、テーブルにアップロードすることはできません。計算カラムがある同期モデルを展開すると、タイムスタンプベースのダウンロードに使用するトリガの作成に失敗する場合があります。カラムを同期対象から除外するか、テーブルをダウンロード専用に設定します (このとき、スナップショット・ダウンロードを使用するか、生成済み統合 SQL ファイルを編集して、計算カラムをトリガ定義から削除します)。

  計算カラムをコピーすると、新しいリモート・スキーマを展開して新しいリモート・データベースを作成する際に構文エラーが発生します。計算カラムを扱う際は、次のいずれかを行います。

  ○ 同期モデルを既存のリモート・データベースに展開する。

  ○ リモート・スキーマから計算カラムを除外する。計算カラムがある統合データベースのテーブルを同期する場合、テーブルにアップロードすることはできません。

  Microsoft SQL Server AdventureWorks サンプル・データベースには、計算カラムが含まれています。このデータベースを使用してモデルを作成する場合、カラムをダウンロード専用に設定するか、カラムを同期対象から除外します。
- **論理削除**　Mobile Link 同期モデルでの論理削除のサポートは、論理削除カラムが統合データベースのみにあり、リモート・データベースにはないことを前提としています。統合スキーマを新しいリモート・スキーマにコピーする場合、モデルの同期設定の論理削除カラムに対応するすべてのカラムを除外します。新しいモデルでは、デフォルトのカラム名が削除されます。

  論理削除カラム名をリモート・スキーマに追加するには、次の手順に従います。

  1. ウィザードで **[論理削除を使用する]** を選択します。
  2. 論理削除カラム名を変更し、統合データベースのどのカラム名とも一致しないようにします。
  3. ウィザードが完了したら、リモート・スキーマを更新し、デフォルトのテーブル選択を保持します。論理削除カラム名がスキーマ変更リストに表示され、リモート・スキーマに追加されます。

> **注意**
> リモートの論理削除カラムのカラム・マッピングを統合データベースの論理削除カラムに設定する必要があります。

配備に関する考慮事項

- **長いオブジェクト名**　配備時に作成されるデータベース・オブジェクトには、データベースでサポートされている長さより長い名前が割り当てられる場合があります (新しいオブジェクト名がベース・テーブル名にサフィックスを追加して作成されるため)。この場合、(データベースに直接ではなく) ファイルのみに配備し、生成された SQL ファイルを編集して、長すぎる名前をすべて置換します。

- **新しいリモート・スキーマ**　**同期モデル作成ウィザード**で新しいリモート・スキーマを作成する場合、新しいリモート・データベースのカラムには統合データベース内のカラムのインデックスは格納されません。外部キーとデフォルトのカラム値は新しいリモート・データベースにコピーされます。ただし、このサポートは ODBC ドライバによって返されるデータベースのメタ・データに依存しており、ドライバの問題が原因で構文エラーやその他のエラーが生じる可能性があります。たとえば、ドライバがレポートしたデフォルトのカラム値が、SQL Anywhere または Ultra Light リモート・データベースではデフォルト値の宣言に使用できないフォーマットだった場合、エラーが起こる可能性があります (展開時の構文エラーなど)。

  計算カラムをコピーすると、新しいリモート・スキーマを展開して新しいリモート・データベースを作成する際に構文エラーが発生します。計算カラムを扱う際は、次のいずれかを行います。

  ○ 同期モデルを既存のリモート・データベースに展開する。

  ○ リモート・スキーマから計算カラムを除外する。計算カラムがある統合データベースのテーブルを同期する場合、テーブルにアップロードすることはできません。

  Ultra Light データベースでは、NCHAR カラムと NVARCHAR カラムがサポートされません。そのため、これらのデータ型のカラムがある統合データベースのテーブルを使用して Ultra Light リモート・データベースの新しいリモート・スキーマを生成することはできません。

  展開の完了後、リモート・スキーマを作成または変更して、その後モデルのスキーマを変更することができます。

- **プロキシ・テーブル**　テーブルを他のデータベースにプロキシする統合データベース・テーブルと同期することはできますが、タイムスタンプベースのダウンロードのために TIMESTAMP カラムを使用するには、ベース・テーブルとプロキシ・テーブルの両方に TIMESTAMP カラムを追加する必要があります。**同期モデル展開ウィザード**ではカラムをプロキシ・テーブルやそのベース・テーブルに追加することはできません。そのため、ベース・テーブルとプロキシ・テーブルの両方に存在するカラムを使用するか、シャドー・テーブルまたはスナップショット・ダウンロードを使用する必要があります。

- **マテリアライズド・ビュー**　タイムスタンプベースのダウンロードを使用していて、タイムスタンプ・カラムを統合テーブルに追加するように選択した場合は、テーブルに依存するマテリアライズド・ビューを無効にしてから展開を開始します。こうしないと、テーブルの変更時にエラーが発生する場合があります。SQL Anywhere 統合データベースでは、sa_dependent_views システム・プロシージャを使用して、テーブルに依存するマテリアライズド・ビューがあるかどうかを判別します。sa_dependent_views システム・プロシージャを参照してください。

その他の考慮事項

- **Oracle 統合データベースに基づくリモート・データベースの作成**　Oracle 統合データベースに基づいて SQL Anywhere または Ultra Light リモート・データベースを作成する場合、統合デー

タベースの DATE カラムを TIMESTAMP に変更することが必要になる場合があります。この処理を行わないと、秒未満の情報が更新時に失われます。

- **プロキシ・テーブル**　トリガによって管理されているシャドウ・テーブルを使用している場合、このトリガとシャドウ・テーブルはベース・テーブルにある必要があります。ベース・テーブルのトリガは、ベース・テーブルに対してプロキシ・テーブルを定義しているデータベース内のシャドウ・テーブルを変更できません。

モデルの作成

♦ **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設定します。**

1. **[プログラム] - [SQL Anywhere 11] - [Sybase Central]** を選択します。
2. **[ツール] - [Mobile Link 11] - [Mobile Link の同期の設定]** を選択します。
3. **[ようこそ]** ページで、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。**[次へ]** をクリックします。
4. **[プライマリ・キー要件]** ページで、3 つのチェックボックスを選択します。プライマリ・キーの詳細については、「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。**[次へ]** をクリックします。
5. **[統合データベース・スキーマ]** ページで、**[統合データベースの選択]** をクリックし、データベースを選択します。**[次へ]** をクリックします。

   Mobile Link アプリケーションの統合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**同期モデル展開ウィザード**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)。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」 33 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。別の所有者に設定するには、設定する所有者により所有される既存のリモート・データベースを使用してください。
7. **同期モデル作成ウィザード**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、モデル以外での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」 35 ページと「モデルの配備」 48 ページを参照してください。

説明

- 1 つのモデルに設定できるパブリケーションは 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、設定する所有者により所有される既存の 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、モデル以外で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を変更することはできません。「Ultra Light で使用するデータベース作成パラメータの選択」『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」 32 ページ
- 「モデルの配備」 48 ページ

統合データベースの変更

データベースを Mobile Link 統合データベースとして使用するには、まずテーブル、カラム、トリガなどの同期に必要なオブジェクトを追加する必要があります。これは、データベースに対して設定スクリプトを実行することによって行われます。サポートされている各 RDBMS 用に個別の設定スクリプトがあります。これらのスクリプトは、すべて *install-dir¥MobiLink¥setup* にあります。スクリプトによる処理の内容を詳細に確認するには、テキスト・エディタでスクリプトを開きます。

Mobil Link 同期モデルを Sybase Central で作成する場合、ソフトウェアによるスクリプトの自動実行を選択することができます。また、手動で実行することもできます。**同期モデル作成ウィザード**を最初に起動したときに、Mobile Link の設定をインストールするよう要求されます。ここでインストールを行わない場合、モデルの展開時にもう一度インストールを要求されます。

参照

- 「統合データベースの設定」『Mobile Link - サーバ管理』
- 「モデルの配備」 48 ページ

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以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。

Sybase Central の外部での Mobile Link アプリケーションの変更

展開したモデルは Sybase Central の外部で変更することもできます。

たとえば、システム・プロシージャを使用して Mobile Link スクリプトの追加や変更を行うことができます。「Mobile Link システム・プロシージャ」 『Mobile Link - サーバ管理』を参照してください。

Sybase Central の外部で変更を加えるときも、**管理**モードと同じルールが適用されます。つまり、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。

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]**
   - **[リモートにのみダウンロード]**
   - **[同期しない]**

   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、**[同期しない]** 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・データベースを作成した場合は、**[新しいリモート・テーブル]** ウィンドウを使用します。「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」 37 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、**[新しいリモート・テーブル]** ウィンドウを使用します。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 **[新しいリモート・テーブル]** ウィンドウを開くには、**[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、**[新しいリモート・テーブル]** ウィンドウ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」 46 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、モデル以外でリモート・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」 46 ページ
- 「テーブル・マッピングとカラム・マッピングの変更」 35 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、このリモート・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。

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。

> **注意**
> プライマリ・キーは削除できません。

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]** を選択した場合は、テーブルを右クリックして **[イベントに移動]** を選択することで、**[イベント]** タブを開きます。
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]** カラム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。(Mobile Link ユーザ名用に使用するカラムは、ユーザ名として使用する値を格納できるタイプである必要があります。)サブセットの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リモート ID を設定できます。(リモート ID 用に使用するカラムは、リモート ID として使用する値を格納できるタイプである必要があります。)サブセットの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]** ドロップダウン・リストで、**[なし]**、**[ユーザ]**、**[リモート]**、**[カスタム]** のいずれかのダウンロード・サブセットを選択します。
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 download_cursor スクリプトの作成に使用する情報を追加できます。download_cursor をすべて自分で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・データベース間でのローの分割」 『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刻の使用」 『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。**[ダウンロード・タイプ]** カラムで [タイムスタンプ]、**[サブセットのダウンロード]** カラムで [なし] をそれぞれ選択します。テーブルを右クリックし、**[イベントに移動]** を選択します。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の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] タブで、Customer テーブルの **[ダウンロード・タイプ]** で **[タイムスタンプ]**、**[サブセットのダウンロード]** で **[カスタム]** をそれぞれ選択します。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、同じ結果でパフォーマンスがよくなります。

- **[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、ここで完了です。
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. **[カスタム]** 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る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が表示されます。また、このスクリプトの生成方法が**同期モデル作成ウィザード**による生成、ユーザ定義、生成されたスクリプトの上書きのうちどれであるかも表示されます。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、スクリプトは完全にそのユーザの制御下に置かれるため、**モデル・**モードで関連する設定に変更が加えられても、スクリプトは自動的に変更されません。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、変更した生成スクリプトをリストアしたり、無視されるように設定した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したりすることができます。リストアまたは削除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出しです。

   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」 48 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」 50 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決定します。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 イベント」 『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も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必要な処理については、「サーバ起動同期のクイック・スタート・ガイド」 『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、[スキーマの更新] 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。

[新しいリモート・テーブル] ウィンドウはリモート・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」 37 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。
2. 次のオプションのうちの 1 つを選択してください。
   - **[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. **スキーマ更新ウィザード**の指示に従います。

   [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すべて切断されます。モデルを展開するまで、モデル以外での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。

4. **[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」 35 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。次の複数の要素を展開できます。

- 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 (生成されるバッチファイルの先頭には変数宣言があり、バッチ ファイルの実行前にこの変数宣言を編集できます)
- 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

同期モデル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展開できます。

- 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。

♦ **SQL ファイルから統合データベースに展開するには、次の手順に従います。**

- **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;uid=myuserid;pwd=mypassword"
  ```

  一部のドライバでは、DSN にユーザ ID とパスワードがあるので、これらの指定は不要です。

注意
展開時にシャドー・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・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・データベースに展開することもできます。

♦ **SQL ファイルから統合データベースに展開するには、次の手順に従います。**

- **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
  ```

  既存のリモート・データベースを使用している場合は、パスワードの入力を要求されます。

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。

- 指定するオプションで 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、指定するオプションで 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、指定するオプションで 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。
2. **同期モデル展開ウィザード**の指示に従います。
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」 50 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**・モード以外で展開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、モデル以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。展開することで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」 46 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**の最初のページ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* という名前で保存されたモデル・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

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¥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ための 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・ソースの作成」『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
   ```

   既存のリモート・データベースを使用している場合は、パスワードの入力を要求されます。

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。

     ```
     dbeng11 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』

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。

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」 58 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・データベース間でのローの分割」『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 11 - 紹介』を参照してください。

詳細情報

- 「Ultra Light CustDB サンプル」『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

MobileLink でサポートされている任意の統合データベースを CustDB 統合データベースとして使用できます。

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 11 CustDB という DSN はインストール環境に含まれています。

このデータベースは、*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*
SQL Server	*custmss.sql*
Oracle	*custora.sql*
DB2 LUW	*custdb2.sql*
MySQL	*custmys.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、DB2 LUW、MySQL、Oracle、SQL Server の場合)。

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 11 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。
   - MySQL 統合データベースの場合は、*syncmys.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、*samples-dir¥MobiLink¥CustDB* 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。
   - MySQL 統合データベースの場合は、*custmys.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・コンピュータ上で、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。
   a. **[スタート] - [プログラム] - [SQL Anywhere 11] - [ODBC アドミニストレータ]** を選択します。
   b. **[追加]** をクリックします。
   c. 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。
   d. この ODBC データ・ソースに CustDB という名前を付けます。
   e. **[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 (DB2 LUW の場合)。

1. DB2 LUW サーバ上に統合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。

a. 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。

b. 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 LUW のマニュアルを参照してください。

3. 以下のように、*MobiLink¥setup¥syncdb2.sql* ファイルを使用して、Mobile Link システム・テーブルを DB2 LUW 統合データベースに追加します。

a. *syncdb2.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 ~
```

b. サーバまたはクライアント・コンピュータで、DB2 LUW 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 *syncdb2.sql* を実行します。

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

4. *samples-dir¥MobiLink¥CustDB* にある *custdb2.class* を DB2 LUW サーバの *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。

5. 以下のように、データ・テーブルを CustDB に追加します。

a. 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
```

b. 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。

c. 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
```

d. 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
```

6. DB2 LUW クライアント上で、DB2 LUW 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。

a. 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 11] - [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。

b. **[ユーザー DSN]** タブで、**[追加]** をクリックします。

c. **[データ ソースの新規作成]** ウィンドウで、DB2 LUW 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。

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 LUW のマニュアル
- Mobile Link の推奨 ODBC ドライバ

7. 以下のようにして、DB2 LUW クライアント上で *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、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、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 LUW 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 LUW データベースの 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 LUW 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 LUW 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。

   ```
   mlsrv11 -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。
   a. 使用中の PC で、Palm Desktop を起動します。
   b. [Palm Desktop] ツールバーの **[インストール]** をクリックします。
   c. **[追加]** をクリックします。SQL Anywhere 11 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。
   d. **[開く]** をクリックします。
   e. 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。
   a. 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。
   b. 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。
   c. プロンプトで、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」 67 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ウィンドウが表示されます。

d. 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。

e. 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。この手順は、データベースを変更したときのみ行ってください。

a. 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。

b. Palm デバイスをクレードルに置きます。

c. 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 11 - 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 LUW 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 11.0',
'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**

```
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。
   a. Palm デバイス用に、PC で Palm Desktop を起動します。
   b. [Palm Desktop] ツールバーの **[インストール]** をクリックします。
   c. **[追加]** をクリックします。SQL Anywhere 11 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。
   d. **[終了]** をクリックします。
   e. 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。
   a. Palm デバイスで [ULDBUtil] アイコンを選択します。
   b. CustDB を選択し、**[データを削除します]** を選択します。
   c. 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。

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 テーブルのローのうち 1 つのみについて、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・データベース間でのローの分割」『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、一部のデータ、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。
   a. コマンド・プロンプトで、*samples-dir¥MobiLink¥Contact* に移動します。
   b. 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。
   a. コマンド・プロンプトで、*samples-dir¥MobiLink¥Contact* に移動します。
   b. 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。
   a. コマンド・プロンプトで、*samples-dir¥MobiLink¥Contact* に移動します。
   b. 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) に割り当てられます。

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。**sync_user** は REMOTE DBA 権限を持っているので、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。そのため、**sync_user** の ID およびパスワードを使用しても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。**mlmaint** 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・データベース間でのローの分割」『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。このテーブルでは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 = ?
```

このテーブルでは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、それぞれの注文数が 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。

Mobile Link チュートリアル

この項には、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。

チュートリアル：Mobile Link の概要

目次

Mobile Link チュートリアルの概要

このチュートリアルでは、統合データベースの設定、ODBC データ・ソースおよびリモート・データベースの作成、アプリケーションを調整するためのデータベース・オブジェクトの設定など、Mobile Link アプリケーションの開発方法を示します。

必要なソフトウェア

- SQL Anywhere 11

前提知識と経験

基本的なコンピュータ・スキル。

目的

次の内容を理解します。

- Mobile Link の基本概念
- Mobile Link アプリケーションの開発方法

関連項目

Mobile Link の概要については、次を参照してください。

- 「Mobile Link 同期の概要」 3 ページ
- 「Mobile Link 統合データベース」 『Mobile Link - サーバ管理』
- 「Mobile Link のモデル」 27 ページ
- 「同期の方法」 『Mobile Link - サーバ管理』

レッスン 1 : ユニークなプライマリ・キーの確保

このチュートリアルでは、Mobile Link 同期の設定が行われていないサンプル・データベース Demo を使用します。すべてのテーブルにはプライマリ・キーがありますが、Demo データベース内でしかユニークではありません。Demo を同期システム内で統合データベースにするためには、プライマリ・キーがすべてのリモート・データベースを含むシステム全体でユニークである必要があります。

レッスン 2：同期モデル作成ウィザードの実行

♦ **同期モデル作成ウィザードを実行するには、次の手順に従います。**

1. Sybase Central の左ウィンドウ枠で、**[Mobile Link]** を右クリックし、**[新規] - [同期モデル]** を選択します。
2. 次のフィールドを完成させます。
   a. **[新しい同期モデルの名前を指定してください。]** フィールドに **MLDemo** と入力します。
   b. **[新しい同期モデル・ファイルを保存するフォルダを指定してください。]** フィールドに *C:¥temp* と入力します。
   c. **[次へ]** をクリックします。
3. 3 つのチェックボックスをすべて選択します。**[次へ]** をクリックします。
4. **[統合データベースの選択]** をクリックします。
5. 統合データベースの接続パラメータを指定します。
   a. **[ODBC データ・ソース名]** をクリックします。
   b. **[ODBC データ・ソース名]** リストで、**[SQL Anywhere 11 Demo]** を選択します。
   c. **[OK]** をクリックします。
   d. システム設定をインストールするかどうかを確認するプロンプトが表示されたら、**[はい]** をクリックします。
6. **[次へ]** をクリックします。
7. **[いいえ、新しいリモート・データベース・スキーマを作成します]** をクリックします。**[次へ]** をクリックします。
8. **[すべて選択]** をクリックします。**[次へ]** をクリックします。
9. **[ダウンロード・タイプ]**、**[タイムスタンプ・ダウンロードのオプション]**、**[削除のダウンロード]**、**[サブセットのダウンロード]** の各ページで **[次へ]** をクリックして、デフォルトの設定をそのまま使用します。
10. **[ローベースの競合検出]** をクリックします。**[次へ]** をクリックします。
11. **[統合データベースの勝ち]** をクリックします。**[次へ]** をクリックします。
12. **[次へ]** をクリックします。
13. **[完了]** をクリックします。

チュートリアル：スクリプトの作成と同期のモニタリング

目次

概要

目的

このチュートリアルの目的は、次のタスクに関する知識と経験を得ることです。

- 統合データベースのスキーマをリモート・データベースに移行する。
- Sybase Central または Interactive SQL を使用して同期に必要な基本スクリプトを作成し、統合データベースに保存する。
- 競合の検出と解決のためのスクリプトを作成する。
- ログ・ファイルと Mobile Link モニタを使用して同期をモニタリングする。

関連項目

Mobile Link のアーキテクチャの詳細については、「Mobile Link 同期の概要」 3 ページを参照してください。

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。

Interactive SQL の詳細については、「Interactive SQL 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。

Sybase Central の詳細については、「Sybase Central 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。

同期の概要については、「チュートリアル：Mobile Link の概要」 93 ページを参照してください。

Mobile Link での競合解決の詳細については、「競合の解決」 『Mobile Link - サーバ管理』を参照してください。

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

レッスン 1：SQL Anywhere 統合データベースの設定

このレッスンでは、SQL Anywhere 統合データベースを設定する手順を説明します。

1. 統合データベースとスキーマを作成します。
2. Mobile Link 設定スクリプトを実行します。
3. 統合データベース用の ODBC データ・ソースを定義します。

統合データベースの作成

次の手順では、Sybase Central のデータベース作成ウィザードを使用して統合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、次の手順に従います。**

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Sybase Central]** を選択します。
2. **[ツール] - [SQL Anywhere 11] - [データベースの作成]** を選択します。
3. **[ようこそ]** ページで、**[次へ]** をクリックします。
4. **[次へ]** をクリックします。
5. **[メイン・データベース・ファイルを保存]** フィールドに *C:¥MLmon¥cons.db* と入力します。**[次へ]** をクリックします。
6. **データベース作成ウィザード**の指示に従い、デフォルト値をそのまま使用します。
7. **[完了]** をクリックします。

統合データベース・スキーマの生成

統合データベース・スキーマに、ハードウェア製品の名前と数量を格納する Product テーブルを追加します。

♦ **統合データベース・スキーマに Product テーブルを追加する場合は、次の手順に従います。**

1. Sybase Central の左ウィンドウ枠で、**cons - DBA** を右クリックし、**[ファイル] - [Interactive SQL を開く]** を選択します。
2. Product 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。

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

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

insert into Product(name, quantity)
```

```
  values ( 'Drywall Screws 10lb', 30);

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

go
```

3. 競合解決に使用するテンポラリ・テーブルを作成します。

   「レッスン 6：競合検出と競合解決のためのスクリプトの作成」 109 ページでは、競合が発生したときにこれらのテーブルに値を挿入するスクリプトを作成します。

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

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

4. 各テーブルの作成が成功したことを検証します。

   - たとえば、Product テーブルの内容を検証するには、Interactive SQL で次のコマンドを実行します。

```
select * from Product
```

5. Mobile Link の設定スクリプトを実行して Mobile Link に必要なシステム・オブジェクトを追加します。

   - **[SQL 文]** ウィンドウ枠で次のように入力します。

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

統合データベース用の ODBC データ・ソースを定義します。

SQL Anywhere 11 ドライバを使用して、cons 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. **[スタート]** - **[プログラム]** - **[SQL Anywhere 11]** - **[ODBC アドミニストレータ]** を選択します。
2. **[ユーザー DSN]** タブをクリックし、**[追加]** をクリックします。
3. **[名前]** リストで **[SQL Anywhere 11]** をクリックします。**[完了]** をクリックします。
4. **[SQL Anywhere 11 の ODBC 設定]** ウィンドウで、次の操作を行います。
   a. **[ODBC]** タブをクリックします。
   b. **[データ・ソース名]** フィールドに **sa_cons** と入力します。

c. **[ログイン]** タブをクリックします。
d. **[ユーザ ID]** フィールドに **DBA** と入力します。
e. **[パスワード]** フィールドに **sql** と入力します。
f. **[データベース]** タブをクリックします。
g. **[サーバ名]** フィールドに **cons** と入力します。
h. **[データベース・ファイル]** フィールドに *C:¥MLmon¥cons.db* と入力します。
i. **[OK]** をクリックします。

5. **[OK]** をクリックします。

詳細情報

dbinit コマンド・ライン・ユーティリティを使用した統合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』または「レッスン 3：同期スクリプトの作成」 105 ページを参照してください。

統合データベースの詳細については、SQL Anywhere 以外の RDBMS が統合データベースの場合も含め、「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。

Interactive SQL の詳細については、「Interactive SQL 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。

レッスン 2：SQL Anywhere リモート・データベースの設定

Mobile Link は、統合データベース・サーバと多数のモバイル・データベースとの間で同期を実行できるように設計されています。この項では、リモート・データベースを 2 つ作成します。データベースごとに次の作業を行う必要があります。

- 統合スキーマ内の特定部分の移行
- 同期パブリケーション、同期ユーザ、同期サブスクリプションの作成

SQL Anywhere データベースの作成

レッスン 1 では、Sybase Central を使用してデータベースを作成しました。ここでは、コマンド・ライン・ユーティリティを使用します。この 2 つのツールのどちらを使用しても結果は同じです。

♦ **SQL Anywhere リモート・データベースを作成して起動するには、次の手順に従います。**

1. コマンド・プロンプトで、リモート・データベースを作成するディレクトリに移動します。
2. 次のコマンドを入力して、データベースを作成します。

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

   次のように入力して、remote2 も起動します。

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

   ここでは、-p オプションを使用してページ・サイズを 4K に設定しています。このサイズに設定すると、多くの環境でパフォーマンスが向上します。

   dbinit のオプションの詳細については、「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。
3. 次のように入力してデータベースを起動します。

   ```
   dbeng11 remote1.db
   ```

   次のように入力して、remote2 も起動します。

   ```
   dbeng11 remote2.db
   ```

統合データベース・スキーマのサブセットの移行

統合データベース・スキーマのサブセットを移行するには、次の作業を行う必要があります。

- リモート・データベースへの接続
- リモート・サーバと外部ログインの作成
- Sybase Central のデータベース移行ウィザードの使用

♦ **統合データベース・スキーマのサブセットを remote1 に移行するには、次の手順に従います。**

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Sybase Central]** をクリックします
2. 次の手順でリモート・データベースに接続します。
   a. 左ウィンドウ枠で、**[SQL Anywhere 11]** をクリックします。
   b. **[ファイル] - [接続]** をクリックします。
   c. **[ID]** タブをクリックします。
   d. **[ユーザ ID]** フィールドに **DBA** と入力します。
   e. **[パスワード]** フィールドに **sql** と入力します。
   f. **[データベース]** タブをクリックします。
   g. **[サーバ名]** フィールドに **remote1** と入力します。
   h. **[OK]** をクリックします。
3. 次の手順でリモート・サーバを作成します。
   a. 左ウィンドウ枠で **[リモート・サーバ]** を右クリックし、**[新規] - [リモート・サーバ]** を選択します。
   b. **[新しいリモート・サーバの名前を指定してください。]** フィールドに **my_sa** と入力します。**[次へ]** をクリックします。
   c. **[SQL Anywhere]** をクリックします。**[次へ]** をクリックします。
   d. **[接続情報を指定してください。]** フィールドに **sa_cons** と入力します。**[次へ]** をクリックします。
   e. **[次へ]** をクリックします。
   f. **[現在のユーザの外部ログインを作成する]** をクリックします。
   g. **[ログイン名]** フィールドに **DBA** と入力します。
   h. **[パスワード]** フィールドに **sql** と入力します。
   i. **[パスワードの確認]** フィールドに **sql** と入力します。
   j. **[完了]** をクリックします。
4. 次の手順で統合データベース・スキーマを移行します。
   a. **[ツール] - [SQL Anywhere 11] - [データベースの移行]** をクリックします。
   b. **[次へ]** をクリックします。
   c. **[移行先のデータベースを指定してください。]** リストで **remote1** を選択します。**[次へ]** をクリックします。
   d. **[移行元のリモート・サーバを指定してください。]** リストで **my_sa** を選択します。**[次へ]** をクリックします。
   e. [使用可能なテーブル] リストで **Product** を選択し、**[追加]** をクリックします。**[次へ]** をクリックします。
   f. **[DBA]** をクリックします。**[次へ]** をクリックします。

g. **[データの移行]** をオフにします。**[完了]** をクリックします。

h. **[閉じる]** をクリックします。

5. **remote2** データベースを使用して手順 2 ～ 4 を繰り返します。

同期サブスクリプションとパブリケーション

パブリケーションは、リモート・データベース上の同期対象となるテーブルとカラムを識別します。これらのテーブルとカラムを**アーティクル**と呼びます。同期サブスクリプションは、パブリケーションに対する Mobile Link ユーザのサブスクリプションです。

同期サブスクリプションとパブリケーションはリモート・データベースに格納されます。

♦ リモートの同期パブリケーション、同期ユーザ、同期サブスクリプションを作成するには、次の手順に従います。

1. Sybase Central の左ウィンドウ枠で、**remote1 - DBA** を右クリックし、**[ファイル]** - **[Interactive SQL を開く]** を選択します。
2. 次の手順で remote1 の同期情報を入力します。
   - Interactive SQL で次のコードを実行します。

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

3. Interactive SQL を起動し、remote2 に接続します。
4. 次の手順で remote2 の同期情報を入力します。
   - Interactive SQL で次のコードを実行します。

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

これで、リモート・データベースと統合データベースの準備が完了しました。次のレッスンでは、同期スクリプトを作成します。レッスン 4 では同期を実行します。

詳細情報

パブリケーションとサブスクリプションの定義方法の詳細については、「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。

レッスン 3：同期スクリプトの作成

Sybase Central を使用して同期スクリプトを表示、作成、修正できます。この項では、次の同期スクリプトを作成します。

- **upload_insert**　リモート・データベースに挿入されたデータを統合データベースにどのように適用するかを定義します。
- **download_cursor**　統合データベースからダウンロードする必要があるデータを定義します。

スクリプトは、それぞれ指定のスクリプト・バージョンに属しています。スクリプトは、統合データベースにスクリプト・バージョンを追加した後に追加してください。

♦ スクリプト・バージョンを追加するには、次の手順に従います。

1. Sybase Central の Mobile Link プラグインを使用して cons データベースに接続します。
   a. Sybase Central の左ウィンドウ枠で **[Mobile Link 11]** を選択します。
   b. **[モード]** - **[管理]** を選択します。
   c. **[ファイル]** - **[接続]** を選択します。
   d. **[ID]** タブをクリックします。
   e. **[ODBC データ・ソース名]** をクリックし、**sa_cons** と入力します。**[OK]** をクリックします。
2. スクリプト・バージョン ver1 を追加します。
   a. 左ウィンドウ枠で **[バージョン]** を右クリックし、**[新規]** - **[バージョン]** を選択します。
   b. **[新しいスクリプト・バージョンの名前を指定してください。]** フィールドに **ver1** と入力します。
   c. **[完了]** をクリックします。

♦ 統合データベースに同期テーブルを追加するには、次の手順に従います。

1. **[Mobile Link 11]** リストの左ウィンドウ枠で、**[同期テーブル]** を右クリックし、**[新規]** - **[同期テーブル]** を選択します。
2. **[リモート・テーブルと同じ名前のテーブルを統合データベースで選択する]** をクリックします。
3. **[同期するテーブルの所有者を指定してください。]** リストで **[DBA]** をクリックします。
4. **[同期するテーブルを指定してください。]** リストで **[Product]** をクリックします。
5. **[完了]** をクリックします。
6. アップロード用とダウンロード用のテーブル・スクリプトを統合データベースに新しく追加します。

♦ **Product テーブルのテーブル・スクリプトを追加するには、次の手順に従います。**

1. **[Mobile Link 11]** リストの左ウィンドウ枠で、**[同期テーブル]** を展開します。
2. **[Product]** テーブルを右クリックし、**[新規] - [テーブル・スクリプト]** を選択します。
3. **[テーブル・スクリプトを作成するバージョンを指定してください。]** リストで **ver1** をクリックします。
4. **[テーブル・スクリプトを実行するイベントを指定してください。]** リストで **upload_insert** をクリックします。**[次へ]** をクリックします。
5. **[完了]** をクリックします。
6. Sybase Central の右ウィンドウ枠に、次の SQL 文を入力します。

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

upload_insert イベントは、リモート・データベースに挿入されたデータが統合データベースにどのように適用されるかを決定します。upload_insert の詳細については、「upload_insert テーブル・イベント」『Mobile Link - サーバ管理』を参照してください。

7. **[ファイル] - [保存]** を選択します。
8. 次の SQL 文を使用して、**download_cursor** イベントに対して手順 1 ～ 5 を繰り返します。

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

download_cursor スクリプトは、ダウンロードしてリモート・データベースに (挿入または更新によって) 取り込む必要があるローを、統合データベースから選択するためのカーソルを定義します。download_cursor の詳細については、「download_cursor テーブル・イベント」『Mobile Link - サーバ管理』を参照してください。

詳細情報

ここで作成したスクリプトの詳細については、「upload_insert テーブル・イベント」『Mobile Link - サーバ管理』と「download_cursor テーブル・イベント」『Mobile Link - サーバ管理』を参照してください。

スクリプト・バージョンの詳細については、「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。

スクリプトの追加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。

テーブル・スクリプトの作成方法の詳細については、「テーブル・スクリプト」『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』を参照してください。

同期をカスタマイズするために設定できるイベントの完全なリストについては、「同期イベント」『Mobile Link - サーバ管理』を参照してください。

レッスン 4：Mobile Link 同期の実行

♦ **remote1 と remote2 の同期を実行するには、次の手順に従います。**

1. Mobile Link サーバ・ユーティリティ (mlsrv11) を実行して Mobile Link サーバを起動します。
   - コマンド・プロンプトで次のコマンドを入力します。

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

   mlsrv11 のオプションについては、「Mobile Link サーバ・オプション」『Mobile Link - サーバ管理』を参照してください。

   Mobile Link サーバで要求を処理する準備ができたことを示すメッセージが **Mobile Link サーバ・ウィンドウ**に表示されます。
2. Mobile Link 同期クライアント・ユーティリティ (dbmlsync) を実行して同期を開始します。
   - remote1 を同期するには、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+
     ```

   - remote2 を同期するには、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=remote2;uid=DBA;pwd=sql" -o rem2.txt -v+
     ```

   dbmlsync のオプションの完全なリストについては、「Mobile Link SQL Anywhere クライアント・ユーティリティ (dbmlsync)」『Mobile Link - クライアント管理』を参照してください。

   Mobile Link 同期クライアントが起動すると、Mobile Link の同期が成功したことを示すメッセージが DBMLSync ウィンドウに表示されます。

詳細情報

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

mlsrv11 のオプションの完全なリストについては、「Mobile Link サーバ・オプション」『Mobile Link - サーバ管理』を参照してください。

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。

dbmlsync のオプションの完全なリストについては、「Mobile Link SQL Anywhere クライアント・ユーティリティ (dbmlsync)」『Mobile Link - クライアント管理』を参照してください。

レッスン 5：ログ・ファイルを使用した Mobile Link 同期のモニタリング

テーブルの同期が完了したら、コマンド・ラインで指定して生成したメッセージ・ログ・ファイル、つまり *mlserver.mls*、*rem1.txt*、*rem2.txt* を使用して、同期の経過を表示できます。これらのファイルのデフォルト・ロケーションは、コマンドが実行されたディレクトリです。

♦ **Mobile Link 同期ログ・ファイル内でエラーを探すには、次の手順に従います。**

1. ログ・ファイルをテキスト・エディタで開きます。このチュートリアルでは、ログ・ファイルは *mlserver.mls* です。
2. このファイル内で文字列「MobiLink Server started.」を検索します。
3. ファイルの左側を下へスキャンします。「I.」 で始まる行は情報メッセージ、「E.」 で始まる行はエラー・メッセージです。
4. この例では、「E.」 の横に次のテキストがあります。

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

   このメッセージは、アップロードとダウンロードの前のエラーを示します。同期サブスクリプションまたはパブリケーションの定義にエラーが含まれる可能性があります。
5. 次の形で始まる句を検索します。

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

   この句は、同期要求が確立されたことを示します。
6. 「Working on a request」で始まる句を検索します。これは、クライアントとサーバが通信していることを示します。このメッセージは、高レベルの冗長性を指定している場合に表示されることがあります。

♦ **Mobile Link 同期クライアントのログ・ファイル内でエラーを検出するには、次の手順に従います。**

1. クライアント・ログ・ファイル *rem1.txt* をテキスト・エディタで開きます。
2. このファイル内で文字列「COMMIT」を検索します。この文字列がある場合、同期は成功しています。
3. このファイル内で文字列「ROLLBACK」を検索します。トランザクションがロールバックされている場合、エラーのためトランザクションが完了していません。
4. ファイルの左側を下へスキャンします。「E.」 と表示されている場合、エラーが発生しています。エラーがない場合、同期は正常に完了しています。

詳細情報

Mobile Link サーバのログ・ファイルの詳細については、「Mobile Link サーバの動作のロギング」 『Mobile Link - サーバ管理』を参照してください。

レッスン 6：競合検出と競合解決のためのスクリプトの作成

競合は、統合データベースにローをアップロードしているときに発生します。異なるリモート・データベースで 2 人のユーザが同じローを修正した場合、Mobile Link サーバに 2 つめのローが到着したときに競合が検出されます。同期スクリプトを使用すると、競合を検出して解決できます。

Mobile Link での競合解決の詳細については、「競合の解決」 『Mobile Link - サーバ管理』を参照してください。

例 (在庫管理)

現場に販売担当者が 2 人いるとします。最初の在庫数は 10 個で、販売担当者 1 はそのうちの 3 個を販売しました。この担当者は、リモート・データベース remote1 に記録されている在庫数を 7 に更新します。販売担当者 2 は 4 個を販売し、remote2 に記録されている在庫数を 6 に更新します。

Mobile Link 同期クライアント・ユーティリティを使用して remote1 の同期が実行されると、統合データベース内の在庫数は 7 に更新されます。remote2 の同期が実行されると、競合が検出されます。これは、統合データベース内の在庫数が変更されているためです。

この競合をプログラムで解決するには、次のような 3 つのロー値が必要となります。

1. 統合データベースにある現在の値。

   remote1 の同期が実行された後の統合データベース内の値は 7 です。

2. Remote2 がアップロードした新しいローの値。
3. remote2 が直前の同期の間に取得した古いローの値。

この場合、次のビジネス論理を使用することで、新しい在庫数を計算して競合を解決できます。

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

この式で、リモート・データベースの古い値からリモート・データベースの新しい値を引いて求めている値は、在庫数ではなく、販売担当者 2 が販売した個数です。

競合の検出と解決のための同期スクリプト

競合を検出して解決するには、次のスクリプトを追加します。

- **upload_update**　upload_update イベントは、リモート・データベースに挿入されたデータが統合データベースにどのように適用されるかを決定します。更新の競合の検出には、upload_update の拡張プロトタイプも使用できます。

  upload_update を使用して競合を検出する方法の詳細については、「競合の検出」 『Mobile Link - サーバ管理』を参照してください。

  upload_update の詳細については、「upload_update テーブル・イベント」 『Mobile Link - サーバ管理』を参照してください。

- **upload_old_row_insert**　このスクリプトは、リモート・データベースがその直前の同期で取得した古いロー値を処理するために使用できます。

  upload_old_row_insert の詳細については、「upload_old_row_insert テーブル・イベント」『Mobile Link - サーバ管理』を参照してください。

- **upload_new_row_insert**　このスクリプトは、新しいロー値 (リモート・データベースで更新された値) を処理するために使用できます。

  upload_new_row_insert の詳細については、「upload_new_row_insert テーブル・イベント」『Mobile Link - サーバ管理』を参照してください。

- **resolve_conflict**　競合解決スクリプトは、競合の解決のためにビジネス論理を適用します。

  resolve_conflict の詳細については、「resolve_conflict テーブル・イベント」『Mobile Link - サーバ管理』を参照してください。

♦ 競合の検出と解決のためのスクリプトをインストールするには、次の手順に従います。

1. Interactive SQL を起動します。
   a. コマンド・プロンプトで **dbisql** と入力します。
   b. **[ID]** タブをクリックします。
   c. **[ユーザ ID]** フィールドに **DBA** と入力します。
   d. **[パスワード]** フィールドに **sql** と入力します。
   e. **[データベース]** タブをクリックします。
   f. **[サーバ名]** フィールドに **cons** と入力します。
   g. **[OK]** をクリックします。
2. 競合の検出と解決のためのスクリプトを実装します。

   Interactive SQL で次のコードを実行します。

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

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

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



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

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

  // obtain the old remote value
  SELECT quantity INTO @old_rem_val
    FROM Product_old;

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

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

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

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

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

SQL Anywhere 統合データベースの設定はこれで完了です。

詳細情報

Mobile Link での競合の検出と解決の詳細については、「競合の解決」 『Mobile Link - サーバ管理』を参照してください。

Mobile Link 統合データベースの詳細については、「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 7：Mobile Link モニタによる更新競合の検出

Mobile Link モニタを使用すると、同期が行われたときにその統計情報を収集できます。Mobile Link モニタのグラフィック・チャートでは、水平軸に時間の経過が示され、垂直軸にはタスクが示されます。

Mobile Link モニタを使用することによって、エラーを引き起こしたり、特定の条件を満たす同期を迅速に突き止めることができます。Mobile Link モニタによってパフォーマンスが大幅に低下することはないので、開発時、運用時ともこのモニタを使用することをおすすめします。

この項では、次のことを学習します。

- Mobile Link モニタを起動し、更新の競合が発生している同期をはっきりと識別できるように設定する。
- remote1 と remote2 上の同じローを更新して競合を発生させる。
- Mobile Link モニタを使用して競合を検出する。

♦ **更新の競合を検出できるように Mobile Link モニタを設定するには、次の手順に従います。**

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Mobile Link モニタ]** をクリックします。
2. Mobile Link サーバに接続します。
   a. **[ID]** タブをクリックします。
   b. **[ユーザ]** フィールドに **monitor_user** と入力します。-zu+ オプションを指定して Mobile Link サーバを起動したので、このユーザは自動的に追加されます。
   c. **[パスワード]** フィールドにパスワードを入力するか、フィールドを空白のままにします。
   d. **[データベース]** タブをクリックします。
   e. **[サーバ名]** フィールドに **cons** と入力します。
   f. **[OK]** をクリックします。
3. **[ツール] - [ウォッチ・マネージャ]** をクリックして、Mobile Link モニタのウォッチ・マネージャを起動します。
4. 次の手順で、更新の競合をモニタリングする新しいウォッチを追加します。
   a. **[新規]** をクリックします。
   b. **[名前]** フィールドに **conflict_detected** と入力します。
   c. **[プロパティ]** リストで **conflicted_updates** をクリックします。

      統計のプロパティ conflicted_updates は、アップロードされた更新のうち、競合が検出された更新の数を示します。

      Mobile Link モニタの統計のプロパティの詳細については、「Mobile Link の統計のプロパティ」 『Mobile Link - サーバ管理』を参照してください。
   d. **[演算子]** リストで **[より大]** をクリックします。
   e. **[値]** フィールドに **0** と入力します。

f. **[追加]** をクリックします。

g. **[チャート・パターン]** リストで、**[チャート]** ウィンドウ枠のパターンを設定します。[チャート] ウィンドウ枠は、Mobile Link モニタの中央のウィンドウ枠です。

h. **[概要の色]** リストで、**[概要]** ウィンドウ枠の色を指定します。[概要] ウィンドウ枠は、Mobile Link モニタの下部のウィンドウ枠です。

5. **[OK]** をクリックします。

6. **[OK]** をクリックします。

♦ 更新の競合を生成するには、次の手順に従います。

1. remote1 の在庫数を更新します。

   Screwmaster Drill の最初の在庫数は 10 個で、販売担当者 1 がそのうちの 3 個を販売しました。この担当者は、リモート・データベース remote1 に記録されている在庫数を 7 に更新します。この更新は次の手順で行います。

   a. Interactive SQL を起動します。remote1 にまだ接続していない場合は接続します。

   b. Interactive SQL で次のように実行して、Screwmaster Drill の在庫数を 7 個に更新します。

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

2. remote1 の同期を実行します。

   コマンド・プロンプトで次のコマンドを入力して、Mobile Link 同期クライアントを起動します。

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

   同期が完了すると、統合データベース内の Screwmaster Drill の在庫数は 7 個に更新されます。

3. remote2 の在庫数を更新します。

   販売担当者 2 は 4 個を販売し、remote2 に記録されている在庫数を 6 に更新します。remote2 の同期が実行されると、競合が検出されます。これは、統合データベース内の在庫数が変更されているためです。この更新は次の手順で行います。

   a. Interactive SQL を起動し、remote2 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
      ```

   b. **[ID]** タブをクリックします。

   c. **[ユーザ ID]** フィールドに **DBA** と入力します。

   d. **[パスワード]** フィールドに **sql** と入力します。

   e. **[データベース]** タブをクリックします。

   f. **[サーバ名]** フィールドに **remote2** と入力します。

   g. **[OK]** をクリックします。

h. Interactive SQL で次のように実行して、Screwmaster Drill の在庫数を 6 個に更新します。

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

4. remote2 の同期を実行します。

● Mobile Link 同期クライアントを起動します。

コマンド・プロンプトで次のコマンドを入力します。

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

以上の操作が終わると、Mobile Link モニタに切り替えて同期の結果を確認できます。

♦ Mobile Link モニタを使用して更新の競合を検出するには、次の手順に従います。

1. チャート・スクロールを一時的に停止します。

[ファイル] - **[モニタ]** - **[チャート・スクロールの一時停止]** をクリックします。

2. Mobile Link モニタの **[概要]** ウィンドウ枠、**[チャート]** ウィンドウ枠、**[詳細]** テーブルを使用して、同期についての統計情報を確認します。

a. モニタの **[概要]** ウィンドウ枠 (Mobile Link モニタの下部のウィンドウ枠) で同期を見つけます。更新の競合を発生させた remote2 の同期は、赤で表示されます。

b. **[チャート]** ウィンドウ枠で remote2 の同期を表示するには、**[概要]** ウィンドウ枠内の同期オブジェクトをクリックしてドラッグします。

conflict_detected ウォッチ用に指定したパターンが適用された状態で、同期オブジェクトが表示されます。

c. 同期の詳細を確認するには、ズーム・ツールを使用します。

[表示] - **[ズーム・イン]** を選択します。

d. 同期プロパティを表示するには、同期オブジェクトか、詳細テーブル内の対応するローをダブルクリックします。競合が起きている更新の数を確認するには、**[アップロード]** タブを選択します。

詳細情報

Mobile Link での競合解決の詳細については、「競合の解決」『Mobile Link - サーバ管理』を参照してください。

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

Mobile Link モニタの統計のプロパティの詳細については、「Mobile Link の統計のプロパティ」『Mobile Link - サーバ管理』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。

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

1. 以下のアプリケーションのインスタンスをすべて閉じます。
   - Mobile Link モニタ
   - Sybase Central
   - Interactive SQL
2. タスクバー上で、SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを右クリックし、**[閉じる]** を選択して閉じます。
3. 次の手順で、チュートリアルに関連するすべてのデータ・ソースを削除します。
   a. ODBC アドミニストレータを起動します。
   b. **[スタート] - [プログラム] - [SQL Anywhere 11] - [ODBC アドミニストレータ]** を選択します。
   c. **[ユーザー データ ソース]** リストから **sa_cons** を選択し、**[削除]** をクリックします。
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。
   a. 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。
   b. *cons.db*、*cons.log*、*remote1.db*、*remote1.log*、*remote2.db*、*remote2.log* を削除します。

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。

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

Mobile Link での競合の検出と解決の詳細については、「競合の解決」『Mobile Link - サーバ管理』を参照してください。

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、Mobile Link を使用して Oracle 10g データベースを活用する方法について説明します。ここでは、Oracle 10g 統合データベースと SQL Anywhere リモート・データベースの間の同期を設定します。また、Ultra Light リモート・データベースを設定することもできます。

このチュートリアルは、販売チームに関するデータを活用することを目的としています。このシナリオでは、各販売担当者はリモート同期クライアントです。各販売担当者にはローカルの SQL Anywhere データベースが用意されています。このデータベースは Mobile Link を使用して本社にある社内 Oracle データベースと同期されます。各販売担当者はラップトップまたはハンドヘルド・デバイスを使用して社内データにアクセスし、リモート・データベースからデータを操作します。

必要なソフトウェア

このチュートリアルでは、チュートリアルを実行するローカル・コンピュータに SQL Anywhere (Mobile Link を含む) が完全にインストールされていることを前提としています。このチュートリアルは Oracle Database 10g リリース 2 で作成されています。他のバージョンの Oracle でも機能する場合がありますが、動作は保証されていません。また、このチュートリアルでは Oracle Database 10g がローカル・コンピュータにインストールされていることを前提としていますが、リモートでアクセスすることもできます。

このチュートリアルでは、Oracle Database 10g の基本インストールが行われていることを前提としています。このインストールでは、orcl という名前のスターター・データベースが作成されます。orcl データベースには OE (注文エントリ) と HR (人材) というサンプル スキーマがあります。また、サンプル・スキーマを手動でインストールしたり、Oracle Database Configuration Assistant を使用してインストールすることもできます。これらのサンプル・スキーマのインストールの詳細については、http://www.oracle.com/technology/obe/obe1013jdev/common/OBEConnection.htm を参照してください。

このチュートリアルでは、SYSDBA 権限がある SYS ユーザとして Oracle に接続できることを前提としています。これはレッスン 7 で Oracle システム・ビュー V_$TRANSACTION に対するパーミッションを付与するための必要条件です。SYS ユーザのパスワードは Oracle データベースのインストール時に設定されます。

概要

このチュートリアルでは、次の作業の方法について説明します。

- リモート・スキーマの設計時に、リモート・テーブルの同期方向などの重要な考慮事項を決定する。
- 統合データベースとリモート・データベースにユニークなプライマリ・キーを追加する。
- Mobile Link を Oracle 10g データベースに接続する ODBC データ・ソースを作成する。
- **同期モデル作成ウィザード**を使用して、統合データベースとリモート・データベースの間の同期を設定する。
- モデル・モードを使用して同期モデルをカスタマイズする。

- **同期モデル展開ウィザード**を使用して統合データベースとリモート・データベースを展開する。
- リモート・クライアントを統合データベースと同期する。

関連項目

Mobile Link の同期の概要については、「Mobile Link 同期の概要」 3 ページを参照してください。

Mobile Link モデルの概要については、「Mobile Link のモデル」 27 ページを参照してください。

レッスン 1：スキーマの設計

このチュートリアルでは、OE (注文エントリ) と HR (人材) というサンプル・スキーマがインストールされていることを前提としています。OE スキーマは統合データベースとして使用します。このスキーマには従業員、注文、顧客、製品に関する情報がまとめられています。このチュートリアルでは、主に OE スキーマを使用します。ただし、各販売担当者に関する情報を取得する場合は、HR スキーマの EMPLOYEES テーブルを参照する必要があります。OE スキーマのテーブルのうち、このチュートリアルに関連するテーブルの概要を次に示します。

テーブル	説明
CUSTOMERS	レコードに情報が記録されている顧客。
INVENTORIES	各倉庫に保管されている各製品の数量。
ORDER_ITEMS	各注文の対象となっている製品のリスト。
ORDERS	特定の日付に販売担当者と顧客の間で成立した販売の記録。
PRODUCT_DESCRIPTIONS	各製品に関する、各種言語での説明。
PRODUCT_INFORMATION	システム内の各製品の記録。

リモート・スキーマの設計

まず最初に、リモート・スキーマを設計します。販売担当者ごとに統合データベース全体をコピーしておくことは不要であり、非効率的です。リモート・スキーマは、1 人の特定の販売担当者に関連する情報のみを格納するように設計します。そのため、リモート・スキーマは次のように設計します。

統合テーブル	リモート・テーブル
CUSTOMERS	すべての行を抽出。
INVENTORIES	リモートでは使用しない。
ORDER_ITEMS	sales_rep_id を基準にフィルタ処理。
ORDERS	すべての行を抽出。
PRODUCT_DESCRIPTIONS	リモートでは使用しない。
PRODUCT_INFORMATION	すべての行を抽出。

各販売担当者はすべての顧客と製品の記録を保持し、すべての製品をどの顧客にも販売できるようにする必要があります。このチュートリアルでは、販売担当者は常に顧客と同じ言語を使用していることを前提としているため、PRODUCT_DESCRIPTIONS テーブルは必要ありません。各販売担当者には注文に関する情報が必要ですが、他の販売担当者に関する注文の情報は不要です。そのため、ローは販売担当者の識別子に基づいてフィルタされます。

次に、各テーブルの同期の方向を選択します。リモート・データベースで読み込む情報と、リモート・データベースで作成、変更、または削除する情報について考慮する必要があります。この例では、各販売担当者は製品と顧客のリストを必要としていますが、新しい製品の情報をシステムに入力することはありません。製品と顧客の情報の入力は必ず本社の統合データベースから行うという制限が設けられています。一方で、販売担当者は新しい販売情報を常時記録できる必要があります。このような理由から、各テーブルの同期については次のように決められています。

テーブル	同期
CUSTOMERS	リモートへのダウンロードのみ。
ORDER_ITEMS	ダウンロードとアップロード。
ORDER	ダウンロードとアップロード。
PRODUCT_INFORMATION	リモートへのダウンロードのみ。

レッスン 2：統合データベースの準備

このチュートリアルでは、OE (注文エントリ) サンプル・データベースがインストールされていることを前提としています。このサンプル・スキーマのインストールに関する情報は、Oracle のマニュアルか、またはオンラインで http://www.oracle.com/technology/obe/obe1013jdev/common/OBEConnection.htm を参照してください。

OE データベースは Mobile Link で使用できるよう変更する必要があります。ユーザ定義型として作成されたカラムは削除されます。これらのユーザ定義型を SQL Anywhere が認識できる型に変換することもできますが、このチュートリアルではこの操作は行いません。次に、Mobile Link では OE のクレデンシャルを使用していくつかのトリガを作成する必要があるため、トリガを作成するためのパーミッションを OE ユーザに付与する必要があります。

♦ **統合データベースを準備するには、次の手順に従います。**

1. SYSDBA 権限を持つ SYS ユーザとして、Oracle SQL Plus アプリケーションを使用して接続します。コマンド・プロンプトで次のコマンドを実行します。

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

2. ユーザ定義型として作成されたカラムを削除するには、次のコマンドを実行します。

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

3. OE ユーザのロックを解除し、パスワードを sql に設定します。次のコマンドを実行します。

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

4. OE ユーザがトリガを作成できるようにするには、次のコマンドを実行します。

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

5. orders_customer 外部キーを削除して、CUSTOMERS テーブルの customer_id を参照する新しい外部キーを作成するには、次のコマンドを実行します。

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

ユニークなプライマリ・キーの追加

同期システムでは、テーブルのプライマリ・キーは、異なるデータベース内の同じローを識別する唯一の方法であり、競合を検出する唯一の方法です。使用する各テーブルには、プライマリ・キーが必要です。プライマリ・キーが更新されることはありません。また、1 つのデータベースに挿入されたプライマリ・キーの値が別のデータベースに挿入されないようにする必要があります。

ユニークなプライマリ・キーを生成する方法は複数あります。このチュートリアルでは、簡単に操作を行うため、複合プライマリ・キー方式を使用します。この方式では、統合データベースとリモート・データベースにまたがってユニークな複数のカラムを使用してプライマリ・キーを作成します。

♦ **ユニークなプライマリ・キーを統合データベースに追加するには、次の手順に従います。**

1. SYSDBA 権限を持つ SYS ユーザとして、Oracle SQL Plus アプリケーションを使用して接続します。コマンド・プロンプトで次のコマンドを実行します。

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

2. SALES_REP_ID に追加する値は HR.EMPLOYEES テーブルに存在する必要があります。ORDERS_SALES_REP_FK 外部キーによりこのルールが強制的に適用されます。操作を簡単にするために、次のコマンドを実行してこの外部キーを削除します。

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

3. SALES_REP_ID カラムには NULL 値が含まれているため、このカラムをプライマリ・キーとして追加することはできません。このチュートリアルでは、NULL 値は 1 に置き換えます。次のコマンドを実行します。

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

4. ORDER_ID カラムは ORDERS テーブルの現在のプライマリ・キーです。現在のプライマリ・キーを削除するには、次のコマンドを実行します。

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

5. 複合プライマリ・キーは SALES_REP_ID カラムと ORDER_ID カラムにより構成されます。複合プライマリ・キーを追加するには、次のコマンドを実行します。

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

これらのコマンドを追加すると、Mobile Link サーバでは問題なく統合データベースに接続し、任意の数のリモート・データベースと同期できるよう統合データベースを設定できるようになります。

外部キーの追加

すべてのデータベースに対してユニークなプライマリ・キー

レッスン 4 では、統合スキーマからリモート・スキーマを作成します。このため、リモート・スキーマのプライマリ・キーは統合スキーマと同じものになります。

プライマリ・キーがすべてのデータベースに対してユニークになるようにカラムが選択されています。ORDERS テーブルでは、プライマリ・キーは SALES_REP_ID カラムと ORDER_ID カラムで構成されています。リモート・データベースの sales テーブルに挿入されるすべての値には、ユニークな注文番号が必要です (SALES_REP_ID の値は常に同じです)。これにより、リモートの各 ORDERS テーブルで一意性が確保されます。統合データベースの ORDERS テーブルのプライマリ・キーは、複数の販売担当者がデータがアップロードした場合の競合を防止する役割があります。販売担当者ごとに SALES_REP_ID の値が異なるため、1 人の販売担当者が行うアップロードはいずれも他の販売担当者に対してユニークです。

詳細情報

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

Oracle を統合データベースとして設定する方法の詳細については、「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。

ユニークなプライマリ・キーを生成する各種の方法については、「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。

レッスン 3：Mobile Link との接続

Mobile Link との接続 このレッスンでは、Mobile Link を統合データベースに接続する ODBC データ・ソースを作成します。

♦ **Mobile Link を統合データベースに接続するには、次の手順に従います。**

1. ODBC データ・ソースを作成します。

   SQL Anywhere 11 に付属の Oracle 用 iAnywhere ドライバを使用します。次の設定を使用してください。

ODBC タブのフィールド	値の範囲
[データ・ソース名]	oracle_cons
[ユーザ ID]	OE
[パスワード]	sql
[SID]	orcl
[プロシージャは結果を返す]	オフ
[配列サイズ]	60000

   このチュートリアルでは、Oracle Database 10g の基本インストールが行われていることを前提としています。このインストールでは、orcl という名前のスターター・データベースが作成されます。OE (注文エントリ) スキーマは自動的に orcl にインストールされます。OE スキーマを別のデータベースにインストールした場合、データベース名を SID の値として使用します。

2. ODBC 接続をテストするには、**[接続テスト]** をクリックします。

ODBC データ・ソースを設定すると、Mobile Link プラグインを使用して Oracle データベースに接続し、同期モデルを作成することができるようになります。

詳細情報

Mobile Link で推奨される ODBC ドライバの詳細については、「Mobile Link の推奨 ODBC ドライバ」を参照してください。

iAnywhere Solutions Oracle ドライバの設定で使用するオプションの詳細については、「iAnywhere Solutions Oracle ドライバ」『Mobile Link - サーバ管理』を参照してください。

レッスン 4：同期モデルの作成

同期モデル作成ウィザードを使用すると、統合データベースとリモート・データベースの間の同期を順を追って設定できます。

♦ **同期モデルを作成するには、次の手順に従います。**

1. Sybase Central を起動します。
2. **[ツール] - [Mobile Link 11] - [Mobile Link の同期の設定]** をクリックします。
3. 新しい同期モデルに **sync_oracle** という名前を付け、このモデルのロケーションを入力します。
4. **[プライマリ・キー要件]** ページで、3 つのチェックボックスをすべて選択します。
5. **[統合データベース・スキーマ]** ページで、統合データベースに接続します。
   a. **[統合データベースの選択]** をクリックします。
   b. **[ODBC データ・ソース名]** を選択してから、**oracle_cons** を選択します。**[OK]** をクリックします。
   c. Mobile Link が今回初めて統合データベースを使用する場合、Mobile Link システム設定をインストールするかどうか確認するメッセージが表示されます。Mobile Link システム設定をインストールすると、Mobile Link システム・テーブルと Mobile Link システム・プロシージャが追加されます。**[はい]** をクリックします。

      エラー・メッセージが表示された場合は、データ・ソースが正しく設定されているか確認します。
   d. Mobile Link システム・テーブルと Mobile Link システム・プロシージャが統合データベースに追加されると、名前、ユーザ、製品、バージョンがページ上に表示されます。**[次へ]** をクリックします。

      統合データベース・スキーマからテーブルがロードされます。
6. 次の手順でリモート・スキーマを作成します。
   a. **[リモート・データベース・スキーマ]** ページで **[いいえ、新しいリモート・データベース・スキーマを作成します]** を選択し、**[次へ]** をクリックします。
   b. **[新しいリモート・データベース・スキーマ]** ページで、リモート・スキーマに追加する次のテーブルのチェックボックスをオンにして、**[次へ]** をクリックします。
      - CUSTOMERS
      - ORDERS
      - ORDER_ITEMS
      - PRODUCT_INFORMATION
7. ダウンロード・タイプを次の手順で選択し、設定します。
   a. **[ダウンロード・タイプ]** ページで、**[タイムスタンプベースのダウンロード]** 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すると、前回のダウンロード以降に更新されたデータのみが転送されるため、データ量を最小限に抑えることができます。

b. **[タイムスタンプ・ダウンロードのオプション]** ページで、**[シャドー・テーブルを使用してタイムスタンプ・カラムを保持する]** を選択します。

シャドー・テーブルを使用すると既存のテーブルを変更する必要がないため、推奨されることが多くあります。

8. **[削除のダウンロード]** ページで、リモート・デバイスにレコードの削除を伝達する方法を次のように指定します。

a. リモート・データベースが削除をダウンロードするようにするには、**[はい]** を選択します。

b. **[シャドー・テーブルを使用して削除を記録する]** を選択します。

シャドー・テーブルが統合データベースに作成され、削除が実装されます。

9. **[サブセットのダウンロード]** ページで、リモート・データベースが統合データベースのデータのサブセットのみをダウンロードするように指定します。

a. **[はい、各リモート・データベースに同じデータをダウンロードします]** をクリックします。(モデル・モードのレッスンの手順 2 で、カスタム論理を使用して特定のデータをリモート・データベースにダウンロードする方法を指定します。)

10. **[アップロード競合の検出]** ページで、**[競合検出を実行しない]** を選択します。

多くのアプリケーションでは競合の検出が必要ですが、このチュートリアルでは実行しません。

11. **[パブリケーション、スクリプト・バージョン、説明]** ページで、パブリケーション名として **sync_oracle_publication** と入力し、スクリプト・バージョンとして **sync_oracle_scriptversion** と入力します。

パブリケーションは、同期するデータを指定するリモート・データベース上のオブジェクトです。Mobile Link サーバのスクリプトにより、リモート・データベースから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と、スクリプト・バージョンによりスクリプトをグループ化する方法が定義されます。アプリケーションごとに異なるスクリプト・バージョンを使用できるため、1 つの Mobile Link サーバを管理するだけで複数のアプリケーションを同期できます。

12. **[完了]** をクリックします。

モデルがモデル・モードで表示されます。

モデル・モード

1. データの同期方向を指定するには、**[マッピング方向]** カラムで次のように方向を設定します。

- ORDERS と ORDER_ITEMS の各テーブルは双方向 (アップロードとダウンロード)
- それ以外のテーブルはダウンロードのみ

2. リモート・データベースにダウンロードされたローを、リモート ID を基準として次のようにフィルタします。

a. ORDERS テーブルで、**[サブセットのダウンロード]** を **[カスタム]** に変更します。

b. 画面下部の **[サブセットのダウンロード]** タブを開きます。

c. リモート ID はリモート・データベースをユニークに識別します。リモート ID を基準にローをフィルタするには、download_cursor スクリプトの WHERE 句に制限を追加します。この処理を行うには、**[ダウンロード・カーソルの WHERE 句で使用する SQL 式]** テキスト・ボックスで次のように探索条件を入力します。

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

ダウンロード・カーソル・スクリプトは、各テーブルからどのカラムとローをリモート・データベースにダウンロードするかを指定します。探索条件の指定により、1 人の販売担当者 (データベースのリモート ID が一致する担当者) に関する情報のみをダウンロードするようにすることができます。

3. 同期モデルを保存します。

同期モデルが完成して、展開の準備が完了します。

詳細情報

- 「統合データベースの設定」 『Mobile Link - サーバ管理』
- 「Mobile Link サーバのシステム・テーブル」 『Mobile Link - サーバ管理』
- 「Mobile Link システム・プロシージャ」 『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』
- 「競合の解決」 『Mobile Link - サーバ管理』
- 「競合の解決」 『Mobile Link - サーバ管理』
- 「データのパブリッシュ」 『Mobile Link - クライアント管理』
- 「モデル・モード」 35 ページ
- 「ダウンロード・タイプの変更」 38 ページ
- 「競合の検出と解決の変更」 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」 35 ページ

レッスン 5：同期モデルの展開

この手順では、**同期モデル展開ウィザード**を使用して統合データベースとリモート・データベースを展開します。統合データベースとリモート・データベースの展開は 1 つずつ行うこともできますが、両方一度に行うこともできます。

♦ **同期モデルを展開するには、次の手順に従います。**

1. Sybase Central の左ウィンドウ枠で、同期モデルを右クリックし、**[展開]** を選択します。
2. **[ようこそ]** ページで、**[次の 1 つまたは複数の項目の展開の詳細を指定する]** をクリックし、**[統合データベース]**、**[リモート・データベースと同期クライアント]**、**[Mobile Link サーバ]** を選択します。**[次へ]** をクリックします。
3. **[統合データベースの展開先]** ページで、次の操作を行います。
   a. **[次の SQL ファイルに変更を保存する]** を選択します。このチュートリアルでは、デフォルトのロケーションを使用します。
   b. **[統合データベースに接続して変更を直接適用する]** をクリックして、統合データベースに変更をただちに適用します。
   c. **[統合データベースの選択]** をクリックします。
   d. **[ODBC データ・ソース名]** をクリックし、**oracle_cons** を選択します。
   e. **[OK]** をクリックします。
   f. **[次へ]** をクリックします。

      consolidated ディレクトリを作成するかどうかを確認するメッセージが表示されます。**[はい]** をクリックします。
4. **[リモート・データベースの展開]** ページで、**[新しい SQL Anywhere データベース]** を選択します。**[次へ]** をクリックします。
5. **[新しい SQL Anywhere リモート・データベース]** ページで、次の操作を行います。
   a. **[データベースの作成コマンドが含まれたコマンド・ファイルと SQL ファイルを作成する]** をクリックします。
   b. **[SQL ファイル]** フィールドで、SQL ファイルのデフォルト・ロケーションをそのまま使用します。
   c. **[リモートの SQL Anywhere データベースを作成する]** をクリックします。
   d. **[次へ]** をクリックします。

      remote ディレクトリを作成するかどうかを確認するメッセージが表示されます。**[はい]** をクリックします。
6. **[Mobile Link ユーザ]** ページで、次の操作を行います。
   a. **[Mobile Link サーバに接続するのに使用するユーザ名を指定してください。]** フィールドに **oracle_remote** と入力します。
   b. **[使用するパスワードを指定してください。]** フィールドに **oracle_pass** と入力します。

c. **[次へ]** をクリックします。

7. **[同期ストリーム・パラメータ]** ページで、**[TCP/IP]** をクリックし、**[ポート]** フィールドに **2439** と入力します。**[次へ]** をクリックします。
8. **[クライアント・ストリーム・パラメータ]** ページで、**[ホスト]** フィールドに **localhost** と入力します。**[次へ]** をクリックします。
9. デフォルトの設定をそのまま使用するには、**[Mobile Link サーバ・ストリーム・パラメータ]** ページと **[Mobile Link サーバの冗長性]** ページで、**[次へ]** をクリックします。
10. **[Mobile Link サーバのオプション]** ページで、**[Mobile Link サーバの名前を指定してください。]** フィールドに **oracle_mlsrv** と入力します。

    mlsrv ディレクトリを作成するかどうかを確認するメッセージが表示されます。**[はい]** をクリックします。
11. **[SQL Anywhere リモート同期クライアントの冗長性]** ページで、リモート同期クライアントの冗長性を選択し、リモート・データベース・ログ・ファイルのデフォルトのファイル名を使用します。**[完了]** をクリックします。
12. **[閉じる]** をクリックします。

統合データベースを複数のリモート・クライアントと同期するための設定がすべて完了し、1 つのリモート・クライアントが正常に展開されました。他のリモート・クライアントを展開する場合は、もう一度このウィザードを実行し、新しい Mobile Link ユーザを作成して統合データベースと Mobile Link サーバの展開を終了します。これらのものについては展開がすでに終了しているため、あとは他のリモート同期クライアントを展開するだけです。

詳細情報

- 「モデルの配備」 48 ページ
- 「リモート・データベースの作成」 『Mobile Link - クライアント管理』
- 「Mobile Link ユーザの概要」 『Mobile Link - クライアント管理』

レッスン 6：サーバとクライアントの起動

このレッスンでは、Mobile Link サーバを起動します。

ここまでに、ダウンロード・カーソル・スクリプトを変更して、1 人の販売担当者に関する情報をダウンロードしています。このレッスンでは、リモート ID を販売担当者識別子に設定して、販売担当者を指定します。

Mobile Link サーバを起動します。

デフォルトでは、Mobile Link は、アップロードとダウンロードに対してスナップショット・アイソレーション／READ COMMITTED 独立性レベルを使用します。Mobile Link サーバがスナップショット・アイソレーションを最大限有効に利用できるようにするには、Mobile Link サーバが使用する Oracle アカウントは、Oracle システム・ビュー V_$TRANSACTION にアクセスできる必要があります。アクセスできない場合には、警告が表示され、ローはダウンロードで失われることがあります。

♦ OE ユーザにアクセスを許可するには

1. SYSDBA 権限を持つ SYS ユーザとして、Oracle SQL Plus アプリケーションを使用して接続します。コマンド・プロンプトで次のコマンドを実行します。

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

2. Oracle システム・ビュー V_$TRANSACTION へのアクセスを許可するには、次のコマンドを実行します。

   ```
   GRANT SELECT ON SYS.V_$TRANSACTION TO OE;
   ```

3. Oracle システム・ビュー V_$SESSION へのアクセスを許可するには、次のコマンドを実行します。

   ```
   GRANT SELECT ON SYS.V_$SESSION TO OE;
   ```

♦ Mobile Link サーバを起動するには、次の手順に従います。

1. コマンド・プロンプトで、同期モデルを作成したディレクトリに移動します。(このフォルダは、**同期モデル作成ウィザード**の最初の手順で選択したルート・ディレクトリです。)

   デフォルトのディレクトリ名を使用している場合、ルート・ディレクトリには *sync_oracle¥mlsrv* ディレクトリがあります。

2. mlsrv ディレクトリから次のコマンドを実行します。

   ```
   sync_oracle_mlsrv.bat "DSN=oracle_cons;UID=OE;PWD=sql;"
   ```

   - **sync_oracle_mlsrv.bat**　Mobile Link サーバを起動するために作成されたコマンド・ファイル。
   - **DSN**　ODBC データソース名。
   - **UID**　統合データベースへの接続に使用するユーザ名。
   - **PWD**　統合データベースへの接続に使用するパスワード。

このコマンドが正常に実行されると、**Mobile Link サーバ・メッセージ**・ウィンドウに MobiLink Server Started というメッセージが表示されます。

Mobile Link サーバが起動しなかった場合は、統合データベースの接続情報を確認します。

♦ **リモート・データベースを起動するには、次の手順に従います。**

1. コマンド・プロンプトで、**同期モデル展開ウィザード**によりリモート・データベースを作成したディレクトリに移動します。

   デフォルトのディレクトリ名を使用している場合、ルート・ディレクトリには *sync_oracle¥remote* ディレクトリがあります。

2. 次のコマンドを実行して、SQL Anywhere データベースを起動します。

   ```
   dbeng11 -n remote_eng sync_oracle_remote.db -n remote_db
   ```

   - **dbeng11**　SQL Anywhere データベースの起動に使用するデータベース・サーバ。
   - **remote_eng**　データベース・サーバ名。
   - **sync_oracle_remote.db**　remote_eng で起動するデータベース・ファイル。
   - **remote_db**　remote_eng にあるデータベースの名前。

このコマンドが正常に実行されると、remote_eng という名前の SQL Anywhere データベース・サーバが起動し、remote_db という名前のデータベースがロードされます。

リモート ID の設定

リモート・スキーマでは、各リモート・データベースは 1 人の販売担当者を表しています。作成した同期スクリプトに含まれている論理により、Mobile Link サーバはリモート・データベースのリモート ID に基づいてデータのサブセットをダウンロードします。データベースのリモート ID は有効な販売担当者識別子の値に設定する必要があります。

リモート・デバイスが最初に同期する際に、選択された販売担当者に関するすべての情報がダウンロードされるため、この手順は最初の同期の前に完了している必要があります。

♦ **リモート ID を有効な販売担当者識別子の値に設定するには、次の手順に従います。**

1. 有効な販売担当者識別子を選択します。
   a. SYSDBA 権限を持つ SYS ユーザとして、Oracle SQL Plus アプリケーションを使用して接続します。コマンド・プロンプトで次のコマンドを実行します。

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

   b. ORDERS テーブルで有効な販売担当者識別子のリストを表示するには、次の文を実行します。

      ```
      SELECT COUNT( SALES_REP_ID ), SALES_REP_ID
      FROM OE.ORDERS GROUP BY SALES_REP_ID;
      ```

      この例では、リモート・データベースは SALES_REP_ID が 154 である販売担当者を表しています。

c. Oracle を終了するには、次のコマンドを実行します。

```
exit
```

2. データベースのリモート ID の値を 154 に設定するには、次のコマンドを実行します。

```
dbisql
-c "ENG=remote_eng;DBN=remote_db;UID=DBA;PWD=sql"
"SET OPTION PUBLIC.ml_remote_id='154'"
```

- **dbisql**　SQL Anywhere データベースに対して SQL コマンドを実行するためのアプリケーション。
- **ENG**　データベース・サーバ名として remote_eng を指定します。
- **DBN**　データベース名として remote_db を指定します。
- **UID**　リモート・データベースへの接続に使用するユーザ名。
- **PWD**　リモート・データベースへの接続に使用するパスワード。
- **SET OPTION PUBLIC.ml_remote_id='154'**　リモート ID を 154 に設定するための SQL コマンド。

詳細情報

- 「SQL Anywhere データベース・サーバ」『SQL Anywhere サーバ - データベース管理』
- 「展開したモデルの同期」 50 ページ
- 「Mobile Link サーバの実行」『Mobile Link - サーバ管理』
- 「リモート ID」『Mobile Link - クライアント管理』

レッスン 7：同期

次に、リモート・クライアントの最初の同期を実行します。この処理は、Mobile Link クライアント・プログラム dbmlsync により行います。dbmlsync はリモート・データベースに接続して Mobile Link サーバにより認証されると、リモート・データベースのパブリケーションに基づいてリモート・データベースと統合データベースの同期に必要なすべてのアップロードとダウンロードを実行します。

♦ **リモート・クライアントを同期するには、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実行します。

```
dbmlsync
-c "ENG=remote_eng;DBN=remote_db;UID=DBA;PWD=sql;"
-n sync_oracle_publication
-u oracle_remote -mp oracle_pass
```

- **dbmlsync**　同期アプリケーション。
- **ENG**　リモート・サーバ名を指定します。
- **DBN**　リモート・データベースの名前が表示されます。
- **UID**　リモート・データベースへの接続に使用するユーザ名。
- **PWD**　リモート・データベースへの接続に使用するパスワード。
- **sync_oracle_publication**　同期の実行時に使用するリモート・デバイスのパブリケーション。(このパブリケーションは**同期モデル作成ウィザード**で作成されています。)
- **oracle_remote**　Mobile Link サーバによる認証に使用するユーザ名。
- **oracle_pass**　Mobile Link サーバによる認証に使用するパスワード。

SQL Anywhere Mobile Link クライアントのメッセージ・ウィンドウに同期の進行状況が表示されます。このコマンドが正常に実行されると、dbmlsync アプリケーションによりリモート・データベースに統合データベースの情報のサブセットが格納されます。

同期が失敗した場合は、dbmlsync アプリケーションに渡す接続情報、および Mobile Link ユーザ名とパスワードを確認します。それでも失敗する場合は、使用したパブリケーション名を確認し、統合データベースと Mobile Link サーバが実行中であることを確認します。また、同期ログ (サーバ、クライアントとも) の内容を確認することもできます。

> **注意**
> 別のコンピュータにある dbmlsync アプリケーションを Mobile Link サーバから実行している場合、Mobile Link サーバのロケーションを指定する引数を渡す必要があります。

データの表示

Mobile Link サーバを使用してリモート・クライアントを正常に統合データベースに同期すると、リモート・データベースには 1 人の販売担当者に関する情報が格納されます。SQL Anywhere 11 プラグインを使用すると、Sybase Central でこの状態を確認することができます。

♦ **リモート・データベースのデータを表示する場合は、次の手順に従います。**

1. Sybase Central を起動します。
2. 次の手順でリモート・データベースに接続します。
   a. 左ウィンドウ枠で **[SQL Anywhere 11]** を右クリックして **[接続]** を選択します。
   b. **[ユーザ ID]** に **DBA** と入力し、**[パスワード]** に **sql** と入力します。
   c. **[データベース]** タブで、**[サーバ名]** に **remote_eng** と入力し、**[データベース・ファイル]** に **remote_db** と入力します。
   d. **[OK]** をクリックします。
3. ORDERS テーブルを選択して、右ウィンドウ枠の **[データ]** タブをクリックします。

   ORDERS テーブルでは、すべてのレコードが識別子 154 の販売担当者に関するものです。この販売担当者は、他の販売担当者の販売情報とは関係ありません。このため、リモート ID を基準にしてローをフィルタ処理で除外するよう同期スクリプトを設定し、このデータベースのリモート ID を特定の販売担当者識別子の値に設定します。この販売担当者のデータベースの容量は小さくなり、同期に必要な時間も短くなります。リモート・データベースのサイズは最小限に抑えられているため、新しい販売記録の入力や過去の販売に対する払い戻し処理などの頻繁に行われる処理が迅速かつ効率的に実行されます。

詳細情報

- 「同期処理」 17 ページ
- 「dbmlsync 構文」『Mobile Link - クライアント管理』

クリーンアップ

注文エントリのデータベースを再生成して、チュートリアルのすべての教材をコンピュータから削除します。

♦ 注文エントリのデータベースを再生成するには

- *oe_main.sql* を実行して現在の OE スキーマを削除し、新しい OE スキーマをインストールします。*oe_main.sql* ファイルは *$ORACLE_HOME/demo/schema/order_entry* にあります。

また、Oracle Database Configuration Assistant を使用して新しいデータベースを作成し、サンプル・スキーマを新しくインストールすることもできます。

♦ 同期モデルを削除するには、次の手順に従います。

1. Sybase Central を起動します。
2. 右ウィンドウ枠で、**[Mobile Link 11]** をダブルクリックします。

   sync_oracle モデルが右ウィンドウ枠に表示されます。
3. **sync_oracle** を右クリックして、**[削除]** をクリックします。
4. **[削除の確認]** ウィンドウで、**[削除]** をクリックします。

♦ リモート・データベースを消去するには、次の手順に従います。

- リモート・データベースを消去するには、dberase ユーティリティを使用します。次のコマンドを実行します。

  ```
  dberase sync_oracle¥remote¥sync_oracle_remote.db
  ```


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。

チュートリアル：Adaptive Server Enterprise 統合データベースと Mobile Link の使用

目次

Adaptive Server Enterprise のチュートリアルの概要

このチュートリアルでは、Mobile Link を使用して Adaptive Server Enterprise データベースを活用する方法について説明します。ここでは、Adaptive Server Enterprise 統合データベースと SQL Anywhere リモート・データベースの間の同期を設定します。また、Ultra Light クライアントも使用できます。

このチュートリアルでは、書店チェーンのデータを活用することを目的とします。このシナリオに登場する各書店は、リモート同期環境です。各書店にはローカル SQL Anywhere データベースがあり、本社の Adaptive Server Enterprise データベースと同期しています。各書店には、リモート・データベースのデータにアクセスして操作できるコンピュータを複数設置することもできます。

必要なソフトウェア

このチュートリアルでは、チュートリアルを実行するローカル・コンピュータに SQL Anywhere (Mobile Link を含む) が完全にインストールされていることを前提としています。このチュートリアルは Adaptive Server Enterprise 15.0 を使用して作成されています。他のバージョンの Adaptive Server Enterprise でも機能する場合がありますが、動作は保証されていません。また、このチュートリアルでは、Adaptive Server Enterprise 15.0 がローカル・コンピュータにインストールされていることを前提としています。Adaptive Server Enterprise 15.0 には Sybase Open Client を使用してリモートでアクセスすることも可能です。

このチュートリアルでは、pubs2 サンプル・スキーマが Adaptive Server Enterprise サーバにインストールされていることを前提としています。pubs2 サンプル・スキーマは Adaptive Server Enterprise 15.0 に付属しており、オプションとしてインストールされます。このチュートリアルでは、統合データベースとして使用します。このサンプルに関する情報は Adaptive Server Enterprise のマニュアルに記載されています。また、オンライン (http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sqlug/html/sqlug/sqlug894.htm) で参照することもできます。

このチュートリアルでは、デフォルトの **sa** アカウントを使用します。Adaptive Server Enterprise のインストール時点では、**sa** アカウントのパスワードは NULL です。このチュートリアルでは、NULL のパスワードが有効なパスワードに変更されていることを前提としています。Adaptive Server Enterprise で NULL のパスワードを変更する方法については、http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sag1/html/sag1/sag1615.htm を参照してください。

概要

このチュートリアルでは、次の作業の方法について説明します。

- リモート・スキーマの設計時に、リモート・テーブルの同期方向などの重要な考慮事項を決定する。
- 統合データベースとリモート・データベースにユニークなプライマリ・キーを追加する。
- Mobile Link を Adaptive Server Enterprise データベースに接続する ODBC データ・ソースを作成する。
- **同期モデル作成ウィザード**を使用して、統合データベースとリモート・データベースの間の同期を設定する。

- モデル・モードを使用して同期モデルをカスタマイズする。
- **同期モデル展開ウィザード**を使用して統合データベースとリモート・データベースを展開する。
- リモート・クライアントを統合データベースと同期する。

関連項目

Mobile Link の同期の概要については、「Mobile Link 同期の概要」 3 ページを参照してください。

レッスン 1：スキーマの設計

このチュートリアルでは、pubs2 サンプル・スキーマが Adaptive Server Enterprise サーバにインストールされていることを前提としています。Adaptive Server Enterprise サーバは、ローカルのコンピュータにインストールされているものを使用するか、Sybase Open Client を使用してリモートでアクセスします。

pubs2 サンプル・スキーマは統合データベース・スキーマとして使用します。このスキーマには書店、タイトル、著者、出版社、販売に関する情報が格納されています。次の表は、Adaptive Server Enterprise データベースの各テーブルの説明です。

テーブル	Description
au_pix	著者の写真。
authors	システムに登録されている各タイトルの著者。
discounts	特定の書店で行われている各種の割引の記録。
sales	各販売レコードは、特定の書店での 1 件の販売を表します。
salesdetail	1 件の販売で対象となった各タイトルに関する情報。
stores	各店舗レコードは、システムに登録されている 1 軒の書店または支店を表しています。
titleauthor	どのタイトルがどの著者によって執筆されたかを表します。
titles	システムに登録されているすべての本の記録。
blurbs、publishers、roysched	ここに保存されている情報は、このデモでは使用しません。

リモート・スキーマの設計

書店ごとに統合データベース全体をコピーしておくことは不要であり、非効率的です。リモート・スキーマでは同じテーブル名を使用しますが、各書店に関する情報だけが格納されます。これを実現するため、リモート・スキーマは統合データベースのサブセットとして次のように設計されています。

統合テーブル	リモート・テーブル
au_pix	すべての行を抽出。
authors	すべての行を抽出。
discounts	stor_id を基準にフィルタ処理。

統合テーブル	リモート・テーブル
sales	stor_id を基準にフィルタ処理。
salesdetail	stor_id を基準にフィルタ処理。
stores	stor_id を基準にフィルタ処理。
titleauthor	すべての行を抽出。
titles	すべての行を抽出。
blurbs	リモートでは使用しない。
publishers	リモートでは使用しない。
roysched	リモートでは使用しない。

各書店では、すべてのタイトルと著者の記録を保持し、顧客が在庫を検索できるようにする必要があります。一方で、出版社や印税に関する情報は書店では必要ないため、これらの情報は各書店に対しては同期されません。各書店では販売と割引に関する情報が必要ですが、他の書店の販売や割引に関する情報は必要ありません。そのため、ローは書店の識別子に基づいてフィルタされます。

注意
リモート・データベースで不要になるカラムがある場合は、テーブルからカラムのサブセットを取得することもできます。

次に、各テーブルの同期の方向を選択します。リモート・データベースで読み込む情報と、リモート・データベースで作成、変更、または削除する情報について考慮する必要があります。この例では、書店は著者とタイトルのリストにアクセスする必要がありますが、新しい著者名をシステムに入力することはありません。このため、著者とタイトルは必ず本社の統合データベースから入力するという制限が適用されています。一方で、書店では新しい販売情報を常時記録できるようにする必要があります。これらの要因から、各テーブルの同期の方向は次のようになります。

テーブル	同期
titleauthor	リモートへのダウンロードのみ。
authors	リモートへのダウンロードのみ。
au_pix	リモートへのダウンロードのみ。
titles	リモートへのダウンロードのみ。
stores	リモートへのダウンロードのみ。

テーブル	同期
discounts	リモートへのダウンロードのみ。
sales	ダウンロードとアップロード。
salesdetail	ダウンロードとアップロード。

レッスン 2：統合データベースの準備

このチュートリアルでは、pubs2 データベースへの接続にデフォルトの **sa** アカウントを使用します。Adaptive Server Enterprise のインストール時点では、**sa** アカウントのパスワードは NULL です。このチュートリアルでは、NULL のパスワードが有効なパスワードに変更されていることを前提としています。Adaptive Server Enterprise で NULL のパスワードを変更する方法については、http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sag1/html/sag1/sag1615.htm を参照してください。

このレッスンでは、Mobile Link 同期用の統合データベースのサイズを増やし、ユニークなプライマリ・キーを作成します。

統合データベースのサイズの増加

Mobile Link では、同期のためにシステム・テーブルなどのオブジェクトを pubs2 データベースに追加する必要があります。この処理を行うには、pubs2 データベースのサイズを増やす必要があります。

♦ 統合データベースのサイズを増やすには、次の手順に従います。

1. Adaptive Server Enterprise の isql ユーティリティを使用して、pubs2 データベースに **sa** として接続します。コマンド・プロンプトで、次のコマンドをすべて 1 行に入力して実行します。

   ```
   isql
   -U sa
   -P Your password for sa account
   -D pubs2
   ```

   Adaptive Server Enterprise にリモートでアクセスしている場合は、-S パラメータでサーバ名を指定します。

2. データベースのサイズを増やすための所定のパーミッションを得るには、master データベースにアクセスする必要があります。isql で次のコマンドを実行します。

   ```
   use master
   ```

3. Adaptive Server Enterprise では、データベースはディスクまたはディスクの一部に保存されます。pubs2 データベースのサイズを増やすには、次のコマンドを実行します (pubs2 が格納されているディスクを指定する必要があります)。

   ```
   ALTER DATABASE pubs2 ON disk name = 33
   ```

ユニークなプライマリ・キーの追加

同期システムでは、テーブルのプライマリ・キーは、異なるデータベース内の同じローを識別する唯一の方法であり、競合を検出する唯一の方法です。使用する各テーブルには、プライマリ・キーが必要です。プライマリ・キーが更新されることはありません。また、1 つのデータベースに挿入されたプライマリ・キーの値が別のデータベースに挿入されないようにする必要があります。

ユニークなプライマリ・キーを生成する方法は複数あります。このチュートリアルでは、簡単に操作を行うため、複合プライマリ・キー方式を使用します。この方式では、統合データベースとリモート・データベースにまたがってユニークな複数のカラムを使用してプライマリ・キーを作成します。

♦ **ユニークなプライマリ・キーを統合データベースに追加するには、次の手順に従います。**

1. Adaptive Server Enterprise の isql ユーティリティを使用して、pubs2 データベースに **sa** として接続します。コマンド・プロンプトで、次のコマンドをすべて 1 行に入力して実行します。

   ```
   isql
   -U sa
   -P Your password for sa account
   -D pubs2
   ```

   Adaptive Server Enterprise にリモートでアクセスしている場合は、-S パラメータでサーバ名を指定します。

2. 次のローは、手順 5 で salesdetail テーブルに対して作成した複合プライマリ・キーを基準とするとユニークではありません。操作を簡単にするために、次のコマンドを実行してこのローを削除します。

   ```
   DELETE FROM salesdetail
   WHERE stor_id = '5023'
   AND ord_num = 'NF-123-ADS-642-9G3'
   AND title_id = 'PC8888'

   DELETE FROM salesdetail
   WHERE stor_id = '5023'
   AND ord_num = 'ZS-645-CAT-415-1B2'
   AND title_id = 'BU2075'
   ```

3. 次のインデックスは、手順 5 でのプライマリ・キーの作成に干渉しています。このインデックスを削除するには、次のコマンドを実行します。

   ```
   DROP INDEX authors.auidind
   DROP INDEX titleauthor.taind
   DROP INDEX titles.titleidind
   DROP INDEX sales.salesind
   ```

4. ユニークなプライマリ・キーを追加します。

   ```
   ALTER TABLE au_pix ADD PRIMARY KEY (au_id)
   ALTER TABLE authors ADD PRIMARY KEY (au_id)
   ALTER TABLE titleauthor ADD PRIMARY KEY (au_id, title_id)
   ALTER TABLE titles ADD PRIMARY KEY (title_id)
   ALTER TABLE discounts ADD PRIMARY KEY (discounttype)
   ALTER TABLE stores ADD PRIMARY KEY (stor_id)
   ALTER TABLE sales ADD PRIMARY KEY (stor_id, ord_num)
   ALTER TABLE salesdetail ADD PRIMARY KEY (stor_id, ord_num, title_id)
   ```

これらのコマンドを追加すると、Mobile Link サーバでは問題なく統合データベースに接続し、任意の数のリモート・データベースと同期できるよう統合データベースを設定できるようになります。

注意
プライマリ・キーがない統合データベースとデータを同期することも可能です。ただし、他のテーブルでローを一意に識別するよう設計されたシャドー・テーブルで機能する同期イベントを自分で作成する必要があります。

すべてのデータベースに対してユニークなプライマリ・キー

レッスン 4 では、統合スキーマからリモート・スキーマを作成します。このため、リモート・スキーマのプライマリ・キーは統合スキーマと同じものになります。

プライマリ・キーがすべてのデータベースに対してユニークになるようにカラムが選択されています。sales テーブルでは、プライマリ・キーは stor_id と ord_num カラムで構成されています。リモート・データベースの sales テーブルに挿入されるすべての値には、ユニークな注文番号が必要です (stor_id の値は常に同じです)。これにより、リモートの各 sales テーブルで一意性が確保されます。統合データベースの sales テーブルのプライマリ・キーは、複数の書店でデータがアップロードされた場合の競合を防止する役割があります。書店ごとに stor_id の値が異なるため、1 軒の書店からのアップロードはいずれも他の書店に対してユニークです。

salesdetail テーブルでは、プライマリ・キーは stor_id、ord_num、title_id の各カラムで構成されています。1 件の注文に複数の本のタイトルが存在する場合もあります。リモート・データベースの sales テーブルでは、stor_id と ord_num については複数のローで同じ値になってもかまいませんが、title_id の値はローごとに異なる必要があります。これにより、各リモート・データベースの salesdetail テーブルで一意性が確保されます。sales テーブルと同様に、書店ごとに stor_id の値が異なるため、1 軒の書店から統合データベースへのアップロードはいずれも他の書店に対してユニークです。

詳細情報

Adaptive Server Enterprise に関する問題の詳細については、「Adaptive Server Enterprise 統合データベース」『Mobile Link - サーバ管理』を参照してください。

ユニークなプライマリ・キーを生成する各種の方法については、「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。

レッスン 3：Mobile Link との接続

このレッスンでは、Mobile Link を統合データベースに接続する ODBC データ・ソースを作成します。

♦ **Mobile Link を統合データベースに接続するには、次の手順に従います。**

1. ODBC データ・ソースを作成します。

   Adaptive Server Enterprise に付属の ODBC ドライバを使用する必要があります。このチュートリアルでは、次の設定を使用します。

[General] タブのフィールド	値
Data Source Name	ase_cons
Description	
Server Name (ASE Host Name)	localhost
Server Port	5000
Database Name	pubs2
Logon ID	sa
Use Cursors	オフ

[Transaction] タブのフィールド	値
Server Initiated Transactions	オフ

2. ODBC 接続をテストします。
   a. **[General]** タブで **[Test Connection]** をクリックします。

      Adaptive Server Enterprise のログオン画面が表示されます。
   b. **sa** アカウントのパスワードを入力します。

      [Logon Succeeded] というメッセージが表示されます。

ODBC データ・ソースを設定すると、Mobile Link プラグインを使用して統合データベースに接続し、同期モデルを作成することができるようになります。

詳細情報

Mobile Link で推奨される ODBC ドライバの詳細については、「Mobile Link の推奨 ODBC ドライバ」を参照してください。

レッスン 4：同期モデルの作成

同期モデル作成ウィザードを使用すると、統合データベースとリモート・データベースの間の同期を順を追って設定できます。

♦ **同期モデルを作成するには、次の手順に従います。**

1. Sybase Central を起動します。
2. **[ツール]** - **[Mobile Link 11]** - **[Mobile Link の同期の設定]** を選択します。
3. **[ようこそ]** ページの **[新しい同期モデルの名前を指定してください。]** フィールドに **sync_ase** と入力し、新しいモデルのロケーションを指定します。**[次へ]** をクリックします。
4. **[プライマリ・キー要件]** ページで、3 つのチェック・ボックスをすべて選択します (レッスン 2 でユニークなプライマリ・キーが設定されています)。**[次へ]** をクリックします。
5. **[統合データベース・スキーマ]** ページで、次の操作を行います。
   a. **[統合データベースの選択]** をクリックします。
   b. **[ODBC データ・ソース名]** をクリックし、**ase_cons** を選択します。
   c. **[ユーザ ID]** フィールドに **sa** と入力します。
   d. **[パスワード]** フィールドに **sa** と入力します。
   e. **[OK]** をクリックします。

      Mobile Link が今回初めて統合データベースを使用する場合、Mobile Link システム設定をインストールするかどうか確認するメッセージが表示されます。**[はい]** をクリックします。
   f. **[次へ]** をクリックします。
6. **[リモート・データベース・スキーマ]** ページで、**[いいえ、新しいリモート・データベース・スキーマを作成します]** をクリックします。**[次へ]** をクリックします。
7. **[新しいリモート・データベース・スキーマ]** ページ、**[リモート・データベースに含める統合データベース・テーブルを指定してください。]** リストで、次のテーブルを選択します。
   - au_pix
   - authors
   - discounts
   - sales
   - salesdetail
   - stores
   - titleauthor
   - titles
8. **[次へ]** をクリックします。
9. **[ダウンロード・タイプ]** ページで、**[タイムスタンプベースのダウンロード]** をクリックします。**[次へ]** をクリックします。

タイムスタンプベースのダウンロードを選択すると、前回のダウンロード以降に更新されたデータのみが転送されるため、データ量を最小限に抑えることができます。

10. **[タイムスタンプ・ダウンロードのオプション]** ページで、**[シャドー・テーブルを使用してタイムスタンプ・カラムを保持する]** をクリックします。**[次へ]** をクリックします。

    シャドー・テーブルを使用すると既存のテーブルを変更する必要がないため、推奨されることが多くあります。

11. **[削除のダウンロード]** ページで、次の操作を行います。

    a. **[統合データベース上で削除されたデータを、リモート・データベース上で削除しますか ?]** フィールドで、**[はい]** をクリックします。

    b. **[シャドー・テーブルを使用して削除を記録する]** をクリックします。

       シャドー・テーブルが統合データベースに作成され、削除が実装されます。

    c. **[次へ]** をクリックします。

12. **[サブセットのダウンロード]** ページで、**[はい、各リモート・データベースに同じデータをダウンロードします]** をクリックします。**[次へ]** をクリックします。

    モデル・モードのレッスンの手順 2 で、カスタム論理を使用して特定のデータをリモート・データベースにダウンロードする方法を指定します。

13. **[アップロード競合の検出]** ページで、**[競合検出を実行しない]** をクリックします。**[次へ]** をクリックします。

    このチュートリアルでは競合検出を実行しないよう指定していますが、多くのアプリケーションでは競合検出が必要になります。

14. **[パブリケーション、スクリプト・バージョン、説明]** ページで、次の操作を行います。

    a. **[パブリケーションの名前を指定してください。]** フィールドに **sync_ase_publication** と入力します。

    b. **[スクリプト・バージョンの名前を指定してください。]** フィールドに **sync_ase_scriptversion** と入力します。

       パブリケーションは、同期するデータを指定するリモート・データベース上のオブジェクトです。Mobile Link サーバのスクリプトにより、リモート・データベースから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と、スクリプト・バージョンによりスクリプトをグループ化する方法が定義されます。アプリケーションごとに異なるスクリプト・バージョンを使用できるため、1 つの Mobile Link サーバを管理するだけで複数のアプリケーションを同期できます。

    c. **[完了]** をクリックします。

       モデルがモデル・モードで表示されます。

モデル・モード

1. データの同期方向を指定するには、**[マッピング方向]** カラムで次のように方向を設定します。

   - sales と salesdetail の各テーブルは双方向
   - それ以外のテーブルはダウンロードのみ

2. リモート・データベースにダウンロードされたローを、リモート ID を基準として次のようにフィルタします。

   a. stores テーブルで、**[サブセットのダウンロード]** を **[カスタム]** に変更します。

   b. 画面下部の **[サブセットのダウンロード]** タブを開きます。

   c. リモート ID はリモート・データベースをユニークに識別します。リモート ID を基準にローをフィルタするには、download_cursor スクリプトの WHERE 句に制限を追加します。この処理を行うには、**[ダウンロード・カーソルの WHERE 句で使用する SQL 式]** テキスト・ボックスで次のように探索条件を入力します。

      ```
      "dbo"."stores"."stor_id" = {ml s.remote_id}
      ```

      ダウンロード・カーソル・スクリプトは、各テーブルからどのカラムとローをリモート・データベースにダウンロードするかを指定します。探索条件の指定により、1 つの書店 (データベースのリモート ID が一致する書店) に関する情報のみをダウンロードするようにすることができます。

   d. sales、salesdetail、discounts の各テーブルに対して、次の手順を実行します。式でテーブル名を適切に変更してください。

3. 同期モデルを保存します。

   同期モデルが完成して、展開の準備が完了します。

詳細情報

- 「統合データベースの設定」『Mobile Link - サーバ管理』
- 「Mobile Link サーバのシステム・テーブル」『Mobile Link - サーバ管理』
- 「Mobile Link システム・プロシージャ」『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』
- 「競合の解決」『Mobile Link - サーバ管理』
- 「競合の解決」『Mobile Link - サーバ管理』
- 「データのパブリッシュ」『Mobile Link - クライアント管理』
- 「モデル・モード」 35 ページ
- 「ダウンロード・タイプの変更」 38 ページ
- 「競合の検出と解決の変更」 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」 35 ページ

レッスン 5：同期モデルの展開

同期モデル展開ウィザードを使用すると、統合データベースとリモート・データベースを展開できます。これらのデータベースの展開は 1 つずつ行うこともできますが、両方一度に行うこともできます。**同期モデル展開ウィザード**では、展開のオプションを順を追って設定できます。

♦ **同期モデルを展開するには、次の手順に従います。**

1. Sybase Central の左ウィンドウ枠で、同期モデルを右クリックし、**[展開]** を選択します。
2. **[次の 1 つまたは複数の項目の展開の詳細を指定する]** をクリックし、**[統合データベース]**、**[リモート・データベースと同期クライアント]**、**[Mobile Link サーバ]** を選択します。**[次へ]** をクリックします。
3. **[統合データベースの展開先]** ページで、次のフィールドを完成させます。
   a. **[次の SQL ファイルに変更を保存する]** をクリックし、ファイルのデフォルト・ロケーションをそのまま使用します。

      Mobile Link により *sql* ファイルが生成され、同期設定が可能になるように統合データベースが変更されます。*.sql* ファイルを後で確認し、独自に変更を行うこともできます。この場合、*.sql* ファイルを手動で実行する必要があります。
   b. 統合データベースにすぐに変更を適用する場合、**[統合データベースに接続して変更を直接適用する]** をクリックします。
   c. **[統合データベースの選択]** をクリックします。
   d. **[ODBC データ・ソース名]** を選択してから、**ase_cons** を選択します。
   e. **[ユーザ ID]** フィールドに **sa** と入力します。
   f. **[パスワード]** フィールドに **sa** と入力します。
   g. **[OK]** をクリックします。
   h. **[次へ]** をクリックします。

      consolidated ディレクトリを作成するかどうかを確認するプロンプトが表示されます。**[はい]** をクリックします。
4. **[リモート・データベースの展開]** ページで、**[新しい SQL Anywhere データベース]** をクリックします。**[次へ]** をクリックします。
5. **[新しい SQL Anywhere リモート・データベース]** ページで、次のフィールドを完成させます。
   a. **[データベースの作成コマンドが含まれたコマンド・ファイルと SQL ファイルを作成する]** をクリックします。

      別の *.sql* ファイルが生成されます。このファイルに含まれるコマンドにより、リモート・データベースにすべてのスキーマと同期情報が設定されます。
   b. **[SQL ファイル]** フィールドで、SQL ファイルのデフォルト・ロケーションをそのまま使用します。
   c. **[リモートの SQL Anywhere データベースを作成する]** をクリックします。

この操作を行わない場合は、新しいリモート・データベースを生成して、このデータベースに対して *.sql* ファイルを実行する必要があります。これにより、*.sql* ファイルを後で確認し、独自に変更を行うことができます。

d. **[SQL Anywhere データベース・ファイル]** フィールドで、リモートの SQL Anywhere データベースのデフォルト・ロケーションをそのまま使用します。

e. **[次へ]** をクリックします。

remote ディレクトリを作成するかどうかを確認するプロンプトが表示されます。**[はい]** をクリックします。

6. **[Mobile Link ユーザ]** ページで、次のフィールドを完成させます。

a. **[Mobile Link サーバに接続するのに使用するユーザ名を指定してください。]** フィールドに **ase_remote** と入力します。

b. **[使用するパスワードを指定してください。]** フィールドに **ase_pass** と入力します。

c. **[次へ]** をクリックします。

7. **[同期ストリーム・パラメータ]** ページで、**[TCP/IP]** をクリックし、**[ポート]** フィールドに **2439** と入力します。**[次へ]** をクリックします。

8. **[クライアント・ストリーム・パラメータ]** ページで、**[ホスト]** フィールドに **localhost** と入力します。**[次へ]** をクリックします。

オプションでコンピュータの名前または IP アドレス、使用する別のネットワーク・サーバの名前または IP アドレス、その他のクライアント・ストリーム・オプションを指定することもできます。

9. デフォルトの設定をそのまま使用するには、**[Mobile Link サーバ・ストリーム・パラメータ]** ページと **[Mobile Link サーバの冗長性]** ページで、**[次へ]** をクリックします。

10. **[Mobile Link サーバのオプション]** ページで、**[Mobile Link サーバの名前を指定してください。]** フィールドに **ase_mlsrv** と入力します。**[次へ]** をクリックします。

mlsrv ディレクトリを作成するかどうかを確認するメッセージが表示されます。**[はい]** をクリックします。

11. リモート同期クライアントの冗長性を選択し、リモート・データベース・ログ・ファイルのデフォルトのファイル名を使用します。**[次へ]** をクリックします。

12. **[完了]** をクリックします。

13. **[閉じる]** をクリックします。

統合データベースを複数のリモート・クライアントと同期するための設定がすべて完了し、1 つのリモート・クライアントが正常に展開されました。他のリモート・クライアントを展開する場合は、もう一度このウィザードを実行し、新しい Mobile Link ユーザを作成して統合データベースと Mobile Link サーバの展開を終了します。これらのものについては展開がすでに終了しているため、あとは他のリモート同期クライアントを展開するだけです。

詳細情報

- 「モデルの配備」 48 ページ
- 「リモート・データベースの作成」 『Mobile Link - クライアント管理』
- 「Mobile Link ユーザの概要」 『Mobile Link - クライアント管理』

レッスン 6：サーバとクライアントの起動

このレッスンでは、Mobile Link サーバとリモート・データベースを起動します。

ここまでに、ダウンロード・カーソル・スクリプトを変更して、1 軒の書店に関する情報をダウンロードしています。このレッスンでは、リモート ID を書店識別子に設定して、書店を指定します。

♦ Mobile Link サーバを起動するには、次の手順に従います。

1. コマンド・プロンプトで、同期モデルを作成したフォルダに移動します。(このフォルダは、**同期モデル・ウィザード**の最初の手順で選択したルート・ディレクトリです。)

   所定のディレクトリ名を使用している場合は、*sync_ase¥mlsrv* ディレクトリに移動します。

2. Mobile Link サーバを起動するには、次のコマンドを実行します。

   ```
   sync_ase_mlsrv.bat "dsn=ase_cons;uid=sa;pwd=your password for sa account;"
   ```

   - **sync_ase_mlsrv.bat**　Mobile Link サーバを起動するためのコマンド・ファイル。
   - **dsn**　ODBC データソース名。
   - **uid**　統合データベースへの接続に使用するユーザ名 (Adaptive Server Enterprise の場合、デフォルトは **sa**)。
   - **pwd**　**sa** として接続するためのパスワード。

   このコマンドが正常に実行されると、**Mobile Link サーバ・メッセージ**・ウィンドウに [Mobile Link サーバが起動しました。] というメッセージが表示されます。

   Mobile Link サーバが起動しなかった場合は、統合データベースの接続情報を確認します。

♦ リモート・データベースを起動するには、次の手順に従います。

1. コマンド・プロンプトで、**同期モデル展開ウィザード**によりリモート・データベースを作成したディレクトリに移動します。

   所定のディレクトリ名を使用している場合は、*sync_ase¥remote* ディレクトリに移動します。

2. SQL Anywhere リモート・データベースを起動するには、次のコマンドを入力します。

   ```
   dbeng11 -n remote_eng sync_ase_remote.db -n remote_db
   ```

   - **dbeng11**　SQL Anywhere データベースの起動に使用するデータベース・サーバ。
   - **remote_eng**　データベース・サーバ名。
   - **sync_ase_remote.db**　remote_eng で起動するデータベース・ファイル。
   - **remote_db**　remote_eng にあるデータベースの名前。

   このコマンドが正常に実行されると、remote_eng という名前の SQL Anywhere データベース・サーバが起動し、remote_db という名前のデータベースがロードされます。

リモート ID の設定

リモート・スキーマでは、各リモート・データベースは 1 軒の書店を表しています。作成した同期スクリプトに含まれている論理により、Mobile Link サーバはリモート・データベースのリモート ID に基づいてデータのサブセットをダウンロードします。データベースのリモート ID は有効な書店識別子の値に設定する必要があります。

リモート・デバイスが最初に同期する際に、書店 (ここでは Thoreau Reading Discount Chain) に関するすべての情報がダウンロードされるため、この手順は最初の同期の前に完了している必要があります。

♦ **リモート ID を有効な書店識別子の値に設定するには、次の手順に従います。**

1. 有効な書店識別子を選択します。
   a. Adaptive Server Enterprise の isql ユーティリティを使用して、pubs2 データベースに **sa** として接続します。コマンド・プロンプトで、次のコマンドをすべて 1 行に入力して実行します。

      ```
      isql
      -U sa
      -P Your password for sa account
      -D pubs2
      ```

      Adaptive Server Enterprise にリモートでアクセスしている場合は、-S パラメータでサーバ名を指定します。
   b. stores テーブルで有効な書店識別子のリストを表示するには、次の文を実行します。

      ```
      SELECT * FROM stores
      ```

      このチュートリアルでは、リモート・データベースは Thoreau Reading Discount Chain という名前の書店を表しています。書店識別子は 5023 です。
   c. Adaptive Server Enterprise を終了するには、次のコマンドを実行します。

      ```
      exit
      ```

2. データベースのリモート ID を 5023 に設定するには、次のコマンドをすべて 1 行で入力して実行します。

   ```
   dbisql
     -c "eng=remote_eng;dbn=remote_db;uid=DBA;pwd=sql"
     "SET OPTION PUBLIC.ml_remote_id='5023'"
   ```

   - **dbisql**　SQL Anywhere データベースに対して SQL コマンドを実行するためのアプリケーション。
   - **eng**　データベース・サーバ名として remote_eng を指定します。
   - **dbn**　データベース名として remote_db を指定します。
   - **uid**　リモート・データベースへの接続に使用するユーザ名。
   - **pwd**　リモート・データベースへの接続に使用するパスワード。
   - **SET OPTION PUBLIC.ml_remote_id='5023'**　リモート ID を 5023 に設定するための SQL コマンド。

詳細情報

- 「SQL Anywhere データベース・サーバ」 『SQL Anywhere サーバ - データベース管理』
- 「展開したモデルの同期」 50 ページ
- 「Mobile Link サーバの実行」 『Mobile Link - サーバ管理』
- 「リモート ID」 『Mobile Link - クライアント管理』

レッスン 7：同期

次に、リモート・クライアントの最初の同期を実行します。この処理は、Mobile Link クライアント・プログラム dbmlsync により行います。dbmlsync はリモート・データベースに接続して Mobile Link サーバにより認証されると、リモート・データベースのパブリケーションに基づいてリモート・データベースと統合データベースの同期に必要なすべてのアップロードとダウンロードを実行します。

♦ **リモート・クライアントを同期するには、次の手順に従います。**

● コマンド・プロンプトで、次のコマンドをすべて 1 行に入力して実行します。

```
dbmlsync
  -c "eng=remote_eng;dbn=remote_db;uid=DBA;pwd=sql;"
  -n sync_ase_publication
  -u ase_remote -mp ase_pass
```

- **dbmlsync**　同期アプリケーション。
- **eng=remote_eng**　リモート・データベース・サーバ名を指定します。
- **dbn=remote_db**　リモート・データベース名を指定します。
- **uid**　リモート・データベースへの接続に使用するユーザ名。
- **pwd**　リモート・データベースへの接続に使用するパスワード。
- **sync_ase_publication**　同期の実行に使用するリモート・デバイスのパブリケーション名。(このパブリケーションは**同期モデル作成ウィザード**で作成されています。)
- **ase_remote**　Mobile Link サーバによる認証に使用するユーザ名。
- **ase_pass**　Mobile Link サーバによる認証に使用するパスワード。

SQL Anywhere Mobile Link クライアントのメッセージ・ウィンドウに同期の進行状況が表示されます。このコマンドが正常に実行されると、dbmlsync アプリケーションによりリモート・データベースに統合データベースの情報のサブセットが格納されます。

同期が失敗した場合は、dbmlsync アプリケーションに渡す接続情報、および Mobile Link ユーザ名とパスワードを確認します。それでも失敗する場合は、使用したパブリケーション名を確認し、統合データベースと Mobile Link サーバが実行中であることを確認します。また、同期ログ (サーバ、クライアントとも) の内容を確認することもできます。

> **注意**
> 別のコンピュータにある dbmlsync アプリケーションを Mobile Link サーバから実行している場合、Mobile Link サーバのロケーションを指定する引数も渡す必要があります。

データの表示

Mobile Link サーバを使用してリモート・クライアントを正常に統合データベースに同期すると、リモート・データベースには 1 軒の書店に関する情報が格納されます。SQL Anywhere 11 プラグインを使用すると、Sybase Central でこの状態を確認することができます。

♦ **リモート・データベースのデータを表示する場合は、次の手順に従います。**

1. Sybase Central を起動します。
2. 次の手順でリモート・データベースに接続します。
   a. 左ウィンドウ枠で **[SQL Anywhere 11]** を右クリックして **[接続]** を選択します。
   b. **[ユーザ ID]** に **DBA** と入力し、**[パスワード]** に **sql** と入力します。
   c. **[データベース]** タブで、**[サーバ名]** に **remote_eng** と入力し、**[データベース名]** に **remote_db** と入力します。
   d. **[OK]** をクリックします。
3. 統合データベースから作成されたテーブルが表示されていない場合は、次の手順を実行します。
   a. *remote_db* を右クリックして **[所有者フィルタの設定]** をクリックします。
   b. **[dbo]** を選択して **[OK]** をクリックします。

      統合データベースから作成されたテーブルが左ウィンドウ枠に表示されます。dbo がこれらのテーブルに対して持っている所有権はリモート・データベースで保持されます。
4. 任意のリモート・テーブルを選択して、右ウィンドウ枠の **[データ]** タブをクリックします。

   sales、salesdetail、store の各テーブルでは、すべてのレコードが識別子 5023 の書店に関するものです。この書店は、他の書店の販売情報とは関係ありません。このため、リモート ID を基準にしてローをフィルタ処理で除外するよう同期スクリプトを設定し、このデータベースのリモート ID を特定の書店識別子の値に設定します。この書店のデータベースは容量が少なく、同期に必要な時間も短くなります。リモート・データベースのサイズは最小限に抑えられているため、新しい販売記録の入力や過去の販売に対する払い戻し処理などの頻繁に行われる処理が迅速かつ効率的に実行されます。

詳細情報

- 「同期処理」 17 ページ
- 「dbmlsync 構文」 『Mobile Link - クライアント管理』

クリーンアップ

pubs2 データベースを再生成して、チュートリアルのすべての教材をコンピュータから削除します。

♦ **pub2 データベースを再生成するには、次の手順に従います。**

- pubs2 データベースをインストールするスクリプトを実行するには、次のコマンドを実行します。

```
isql
-U sa
-P Your password for sa account
-i %SYBASE%¥%SYBASE_ASE%¥scripts¥instpbs2
```

Adaptive Server Enterprise にリモートでアクセスしている場合は、-S パラメータでサーバ名を指定します。また、instpbs2 ファイルをローカルのコンピュータにコピーする必要があります。-i パラメータを更新して、instpbs2 ファイルの新しいロケーションを指定する必要があります。

♦ **同期モデルを削除するには、次の手順に従います。**

1. Sybase Central を起動します。
2. 右ウィンドウ枠で **[Mobile Link 11]** をダブルクリックします。

   sync_ase モデルが表示されます。
3. **sync_ase** を右クリックして、**[削除]** を選択します。

♦ **リモート・データベースを消去するには、次の手順に従います。**

- リモート・データベースを消去するには、dberase ユーティリティを使用します。次のコマンドを実行します。

```
dberase sync_ase¥remote¥sync_ase_remote.db
```


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 (mlsrv11) を実行します。

必要なソフトウェア

- SQL Anywhere 11
- 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 (mlsrv11) 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 11 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**
> 独自のカスタム・スクリプトを作成する場合は、クラスと関連メソッドを **public** として定義します。

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含まれています。

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. 次のコマンドを実行します。

```
javac custdbscripts.java -classpath "%sqlany11%¥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 サーバの使用方法については、「CustDB 統合データベースの設定」 58 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。
   a. Sybase Central を起動します。
   b. **[表示] - [フォルダ]** をクリックします。
   c. **[接続] - [Mobile Link 11 に接続]** をクリックします。
   d. **[ID]** タブをクリックします。
   e. **[ODBC データ・ソース名]** をクリックし、**SQL Anywhere 11 CustDB** と入力します。
   f. **[OK]** をクリックします。

   Sybase Central は Mobile Link 11 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。
   a. 左ウィンドウ枠で、**[同期テーブル]** フォルダをダブルクリックし、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。
   b. custdb 11.0 の upload_insert イベントに関連するテーブル・スクリプトを選択します。**[編集] - [削除]** をクリックします。
   c. **[削除の確認]** ウィンドウで、**[はい]** をクリックします。ULCustomer テーブルから custdb 11.0 の upload_insert イベントが削除されます。
3. ULCustomer テーブル用に新規の upload_insert イベントを作成します。
   a. **[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]** をクリックします。

b. スクリプト・バージョンとして custdb 11.0 を選択します。

c. 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。

d. **[新しいスクリプト定義を作成する]** を選択して **[Java]** を選択します。

e. **[完了]** をクリックします。

4. upload_insert event に対して **CustdbScripts.UploadInsert** メソッドを実行するよう Mobile Link サーバに指示します。

a. custDB 11.0 の upload_insert スクリプトを選択します。

b. 右ウィンドウ枠で、次のコードを入力します。

```
CustdbScripts.UploadInsert
```

c. **[ファイル] - [保存]** をクリックします。

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、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 システム・プロシージャ」『Mobile Link - サーバ管理』と「ml_add_java_table_script システム・プロシージャ」『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。

a. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Interactive SQL]** をクリックするか、次のコマンドを実行します。

```
dbisql
```

b. **[ID]** タブをクリックします。

c. **[ODBC データ・ソース名]** をクリックし、**SQL Anywhere 11 CustDB** と入力します。

d. **[OK]** をクリックします。

2. Interactive SQL で次のコマンドを実行します。

```
CALL ml_add_java_table_script(
'custdb 11.0',
'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 11.0	スクリプト・バージョン
ULCustomer	同期テーブル
download_cursor	イベント名
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 システム・プロシージャ」『Mobile Link - サーバ管理』と「ml_add_java_table_script システム・プロシージャ」『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 (mlsrv11) 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実行します。

```
mlsrv11 -c "dsn=SQL Anywhere 11 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. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Ultra Light] - [Windows サンプル・アプリケーション]** をクリックします。
2. 従業員 ID を入力します。

   50 の値を入力し、**[OK]** をクリックします。

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]** をクリックします。
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. **[OK]** をクリックし、新規注文を追加します。

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]** をクリックします。

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メッセージが表示されます。

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」 55 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。
   a. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Interactive SQL]** をクリックするか、次のコマンドを実行します。

      ```
      dbisql
      ```

   b. **[ID]** タブをクリックします。
   c. **[ODBC データ・ソース名]** をクリックし、**SQL Anywhere 11 CustDB** と入力します。
   d. **[OK]** をクリックします。
   e. Interactive SQL で次のコマンドを実行します。

      ```
      CALL ml_add_table_script( 'custdb 11.0',
        'ULCustomer',
        'upload_insert',
        'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

      CALL ml_add_table_script( 'custdb 11.0',
        'ULCustomer',
        'download_cursor',
        'SELECT "cust_id", "cust_name"
         FROM "ULCustomer" WHERE "last_modified" >= ?' );
      COMMIT;
      ```

2. SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを閉じるには、各タスクバーを右クリックし、**[閉じる]** を選択します。
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。

4. Windows サンプル・アプリケーションのデータベースをリセットします。

   samples-dir¥UltraLite¥CustDB ディレクトリから次のコマンドを実行します。

   ```
   newdb.bat
   ```

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。

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 (mlsrv11) を実行します。

必要なソフトウェア

- SQL Anywhere 11
- 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 (mlsrv11) 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 11 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 のレポジトリが含まれています。

.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 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 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。
- テキスト・ファイルに 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 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 の **[プロジェクト] - [既存項目の追加]** を選択します。
   - *install-dir¥Assembly¥v2* にある *iAnywhere.MobiLink.Script.dll* を選択します。Visual Studio の場合は、**[開く] - [リンク ファイル]** を選択します。Visual Studio 2005 の場合は、**[追加] - [リンクの追加]** 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 **[プロパティ] - [全般]** を選択します。Visual Basic .NET の場合、すべてのテキストについて、**[ルート名前空間]** テキスト・フィールド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。

   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 *CustdbScripts.cs* (Visual Basic .NET の場合は *CustdbScripts.vb*) として *c:¥mldnet* に保存します。

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:"%sqlany11%¥Assembly
¥v2¥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:"%sqlany11%¥Assembly
¥v2¥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」 58 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。
   a. Sybase Central を起動します。
   b. **[表示]** 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。
   c. **[接続]** - **[Mobile Link 11 に接続]** を選択します。
   d. **[ID]** タブで、**[ODBC データ・ソース名]** に [SQL Anywhere 11 CustDB] を選択します。
   e. **[OK]** をクリックして接続します。
   f. これで、Sybase Central は Mobile Link 11 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。
   a. 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。
   b. 右ウィンドウ枠で、custdb 11.0 の upload_insert テーブル・スクリプトを開きます。**[ファイル]** - **[言語]** を選択し、言語を .NET に変更します。
3. custdb 11.0 の upload_insert テーブル・スクリプトの .NET メソッド名を入力します。
   a. upload_insert テーブル・スクリプトの内容を削除し、右ウィンドウ枠をブランクにします。
   b. 右ウィンドウ枠に次のように入力します。

      ```
      MLExample.CustdbScripts.UploadInsert
      ```

c. **[ファイル] - [保存]** を選択し、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ています。

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 システム・プロシージャ」『Mobile Link - サーバ管理』と「ml_add_dnet_table_script システム・プロシージャ」『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。
   a. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Interactive SQL]** を選択するか、次のコマンドを実行します。

      ```
      dbisql
      ```

   b. **[ID]** タブをクリックします。
   c. **[ODBC データ・ソース名]** をクリックし、SQL Anywhere 11 CustDB と入力します。
   d. **[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 11.0',
   '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 11.0	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 システム・プロシージャ」『Mobile Link - サーバ管理』と「ml_add_dnet_table_script システム・プロシージャ」『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 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 (mlsrv11) 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 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。

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -dl -o c:¥mldnet¥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。

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -dl -o c:¥mldnet¥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. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Ultra Light] - [Windows サンプル・アプリケーション]** を選択します。
2. 従業員 ID を入力します。

   50 の値を入力し、[OK] 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。
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. [OK] 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メッセージが表示されます。

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」 55 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。
   a. Interactive SQL を開きます。

      [スタート] - [プログラム] - [SQL Anywhere 11] - [Interactive SQL] を選択するか、次のコマンドを実行します。

      ```
      dbisql
      ```

   b. **[ID]** タブをクリックします。
   c. **[ODBC データ・ソース名]** をクリックし、**SQL Anywhere 11 CustDB** と入力します。
   d. **[OK]** をクリックして接続します。
   e. Interactive SQL で次のコマンドを実行します。

      ```
      CALL ml_add_table_script( 'custdb 11.0',
        'ULCustomer',
        'upload_insert',
        'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

      CALL ml_add_table_script( 'custdb 11.0',
        'ULCustomer',
        'download_cursor',
        'SELECT "cust_id", "cust_name"
         FROM "ULCustomer"
         WHERE "last_modified" >= ?' );
      ```

2. タスクバー上で、SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを右クリックし、**[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。

4. Windows サンプル・アプリケーションのデータベースをリセットします。

   samples-dir¥UltraLite¥CustDB ディレクトリから次のコマンドを実行します。

   ```
   newdb.bat
   ```

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。

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 11
- 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 (mlsrv11) 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」 192 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(0,2).equals("ML")) {
      // success: an auth status code of 1000
      authentication_status.setValue(1000);
   } else {
      // fail: an authentication_status code of 4000
      authentication_status.setValue(4000);
   }
}
```

   .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。

   - Java の場合は、ファイルに *MobiLinkAuth.java* という名前を付けて *c:¥mlauth* に保存します。*c:¥mlauth* に移動します。ファイルをコンパイルするには、次のコマンドを実行します。

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


MobiLinkAuth.class ファイルが生成されます。

- .NET の場合は、ファイルに *MobiLinkAuth.cs* という名前を付けて *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。ファイルをコンパイルするには、次のコマンドを実行します。

```
csc /out:c:¥mlauth¥MobiLinkAuth.dll /target:library /reference:"%SQLANY11%¥Assembly
¥v2¥iAnywhere.MobiLink.Script.dll" MobiLinkAuth.cs
```

MobiLinkAuth.dll 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 11 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。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=SQL Anywhere 11 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 11.0',
   '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 11.0',
   '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 システム・プロシージャ」『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 システム・プロシージャ」『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 (mlsrv11) を起動するには、次の手順に従います。**

● Mobile Link サンプル・データベースに接続して、 mlsrv11 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、次のコマンドを実行します。

```
mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -sl java(-cp c:¥mlauth)
```

.NET の場合は、次のコマンドを実行します。

```
mlsrv11 -c "dsn=SQL Anywhere 11 CustDB" -sl dnet(-MLAutoLoadPath=c:¥mlauth)
```

authenticate_user 同期イベントが発生すると、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 11] - [Ultra Light] - [Windows サンプル・アプリケーション] 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 の値として **50** を入力し、[OK] を押します。

   ユーザ ID またはパスワードが無効であることを示す SQL コード -103 の同期エラーが **[Ultra Light CustDB デモ]** ウィンドウに表示されます。

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」 55 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。
   a. 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。
      次のコマンドを実行します。

      ```
      dbisql -c "dsn=SQL Anywhere 11 CustDB"
      ```

   b. 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 11.0',
      '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 11.0',
      '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。
5. Windows サンプル・アプリケーションのデータベースをリセットします。
   samples-dir¥UltraLite¥CustDB ディレクトリから次のコマンドを実行します。

   ```
   newdb.bat
   ```


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。

チュートリアル：ダイレクト・ロー・ハンドリングの概要

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

- 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 同期の概要」 3 ページ
- 「同期の方法」 『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』

Mobile Link のコード・サンプルは、http://www.sybase.com/detail?id=1058600#319 にあります。

質問がある場合は Mobile Link ニュースグループ sybase.public.sqlanywhere.mobilink に投稿できます。

レッスン 1：Mobile Link 統合データベースの設定

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. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Sybase Central]** を選択します。
2. Sybase Central で、**[ツール] - [SQL Anywhere 11] - [データベースの作成]** を選択します。
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]** をデフォルト設定のままにします。**[次へ]** をクリックします。
5. **[メイン・データベース・ファイルを保存]** フィールドに、データベースのファイル名およびパスを入力します。たとえば、*c:¥MLdirect¥MLconsolidated.db* と入力します。**[次へ]** をクリックします。
6. **データベース作成ウィザード**の残りの指示に従い、デフォルト値をそのまま使用します。**[データベースへの接続]** ページで、**[最終切断後にデータベースを停止]** オプションをオフにします。
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・ソースを定義します。

SQL Anywhere 11 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. Sybase Central で、**[ツール] - [SQL Anywhere 11] - [ODBC アドミニストレータを開く]** を選択します。
2. **[ユーザー DSN]** タブをクリックし、**[追加]** をクリックします。
3. **[名前]** リストで **[SQL Anywhere 11]** をクリックします。**[完了]** をクリックします。
4. **[SQL Anywhere 11 の ODBC 設定]** ウィンドウで、次の操作を行います。
   a. **[ODBC]** タブをクリックします。
   b. **[データ・ソース名]** フィールドに **mobilink_db** と入力します。
   c. **[ログイン]** タブをクリックします。
   d. **[ユーザ ID]** フィールドに **DBA** と入力します。
   e. **[パスワード]** フィールドに **sql** と入力します。
   f. **[データベース]** タブをクリックします。
   g. **[サーバ名]** フィールドに **MLconsolidated** と入力します。
   h. **[データベース・ファイル]** フィールドに *c:¥MLdirect¥MLconsolidated.db* と入力します。
   i. **[OK]** をクリックします。
5. **[OK]** をクリックします。

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。

   コマンド・プロンプトで次のコマンドを実行します。

   ```
   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 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 11 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。

   次のコマンドを実行します。

   ```
   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 11¥* は、SQL Anywhere 11 の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。

   ```
   read "c:¥Program Files¥SQL Anywhere 11¥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。

   次のコマンドを実行します。

   ```
   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」 207 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。

   次のコマンドを実行します。

   ```
   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・ダウンロードの処理」『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・データベース間でのローの分割」『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)」 213 ページまたは「MobiLinkOrders の全リスト (.NET)」 215 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」 220 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 the remote.
     // in this example you just use the known column order to determine the column
     // indexes

     // only process this insert if the order_comment is not null
     if( !new_comment_reader.IsDBNull( 2 ) ) {
  int comment_id = new_comment_reader.GetInt32( 0 );
  int order_id   = new_comment_reader.GetInt32( 1 );
  string comments= new_comment_reader.GetString( 2 );
  WriteOrderComment( comment_id, order_id, comments );
     }
  }
  // always close the reader when you are done with it!
  new_comment_reader.Close();
    }
```

7. 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」 220 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。

```
// 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 reading in responses
if (my_reader==null)
 my_reader = new BufferedReader(new FileReader( "c:¥¥MLdirect¥¥orderResponses.txt"));

// send updated comments down to clients
String commentLine;

// read the first line from orderResponses.txt
commentLine = my_reader.readLine();

  while(commentLine != null)
{
  // get the next line from orderResponses.txt
  String[] response_details = commentLine.split("¥t");

  if (response_details.length != 3)
      {
        System.err.println("Error reading from orderResponses.txt");
        System.err.println("Error setting direct row handling download");
        return;
      }

  int comment_id = Integer.parseInt(response_details[0]);
  int order_id = Integer.parseInt(response_details[0]);
  String updated_comment = response_details[2];

  // set an order comment response in the MobiLink download
update_ps.setInt(1, comment_id);
  update_ps.setInt(2, order_id);
  update_ps.setString(3, updated_comment);
   update_ps.executeUpdate();

  // get next line from orderResponses.txt
       commentLine  = my_reader.readLine();
    }
```

.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、*install-dir¥java* にある *mlscript.jar* を参照する必要があります。次のコマンドを実行して Java クラスをコンパイルします。

  ```
  javac -classpath "%SQLANY11%¥java¥mlscript.jar" MobiLinkOrders.java
  ```

  .NET の場合は、次のコマンドを実行します。

  ```
  csc /out:MobiLinkServerCode.dll /target:library /reference:"%SQLANY11%¥Assembly
  ¥v2¥iAnywhere.MobiLink.Script.dll" MobiLinkOrders.cs
  ```

注意
ここに示した例では、プライマリ・キーの値がユニークとはかぎりません。「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。

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」 207 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 reading in responses
        if (my_reader == null)
            my_reader = new BufferedReader(new FileReader( "c:¥¥MLdirect¥¥orderResponses.txt"));

        // get the next line from orderResponses
        String commentLine;
        commentLine = my_reader.readLine();

        // send comment responses down to clients
        while (commentLine != null)
        {
            // get the next line from orderResponses.txt
            String[] response_details = commentLine.split("¥t");

            if (response_details.length != 3)
            {
                System.err.println("Error reading from orderResponses.txt");
                System.err.println("Error setting direct row handling download");
                return;
            }
            int comment_id = Integer.parseInt(response_details[0]);
            int order_id = Integer.parseInt(response_details[1]);
            String updated_comment = response_details[2];

            // set an order comment response in the MobiLink download
            update_ps.setInt(1, comment_id);
            update_ps.setInt(2, order_id);
            update_ps.setString(3, updated_comment);
            update_ps.executeUpdate();

            // get next line
            commentLine = my_reader.readLine();
        }
        update_ps.close();
    }

    public void EndDownload()
        throws IOException
    {
        // close i/o resources
        if (my_reader != null) my_reader.close();
        if (my_writer != null) my_writer.close();
    }
}
```


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」 207 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:¥¥MLdirect¥¥orderComments.txt");
      }
      my_writer.WriteLine("{0}¥t{1}¥t{2}",
         comment_id, order_id, comments);
      my_writer.Flush();
   }

   // method for the handle_UploadData synchronization event.
   public void GetUpload(UploadData ut)
   {
      // get UploadedTableData for remote table called OrderComments
      UploadedTableData order_comments_table_data =
         ut.GetUploadedTableByName("OrderComments");

      // get inserts uploaded by the MobiLink client
      IDataReader new_comment_reader =
         order_comments_table_data.GetInserts();

      while (new_comment_reader.Read())
      {
         // Columns are
         // 0 - "order_comment"
         // 1 - "comment_id"
         // 2 - "order_id"
         // You can look up these values using the DataTable returned by:
         // order_comments_table_data.GetSchemaTable() if the send
         // column names option is turned on at the remote.
         // In this example, you just use the known column order to
         // determine the column indexes
```

```
        // Only process this insert if the order_comment is not null
        if (!new_comment_reader.IsDBNull(2))
        {
            int comment_id = new_comment_reader.GetInt32(0);
            int order_id = new_comment_reader.GetInt32(1);
            string comments = new_comment_reader.GetString(2);
            WriteOrderComment(comment_id, order_id, comments);
        }
    }
    // Always close the reader when you are done with it!
    new_comment_reader.Close();
}

private const string read_file_path =
    "c:¥¥MLdirect¥¥orderResponses.txt";

// method for the handle_DownloadData synchronization event
public void SetDownload()
{
    if ((my_reader == null) && !File.Exists(read_file_path))
    {
        System.Console.Out.Write("There is no file to read.");
        return;
    }
    DownloadTableData comments_for_download =
        _cc.GetDownloadData().GetDownloadTableByName("OrderComments");

    // Add upserts to the set of operation that are going to be
    // applied at the remote database
    IDbCommand comments_upsert =
        comments_for_download.GetUpsertCommand();

    if (my_reader == null)
    {
        my_reader = new StreamReader(read_file_path);
    }
    string comment_line;
    while ((comment_line = my_reader.ReadLine()) != null)
    {
        // three values are on each line separated by '¥t'
        string[] response_details = comment_line.Split('¥t');
        if (response_details.Length != 3)
        {
            throw (new SynchronizationException(
                "Error reading from orderResponses.txt"));
        }
        int comment_id = System.Int32.Parse(response_details[0]);
        int order_id = System.Int32.Parse(response_details[1]);
        string comments = response_details[2];

        // Parameters of the correct number and type have
        // already been added so you just need to set the
        // values of the IDataParameters
        ((IDataParameter)(comments_upsert.Parameters[0])).Value =
            comment_id;
        ((IDataParameter)(comments_upsert.Parameters[1])).Value =
            order_id;
        ((IDataParameter)(comments_upsert.Parameters[2])).Value =
            comments;
        // Add the upsert operation
        comments_upsert.ExecuteNonQuery();
    }
}
```



```
    public void EndDownload()
    {
        if (my_writer != null)
        {
            my_writer.Close();
            my_writer = null;
        }
        if (my_reader != null)
        {
            my_reader.Close();
            my_reader = null;
        }
    }
}
```

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 オプションを使って Mobile Link サーバ (mlsrv11) を起動し、統合データベースに接続します。-sl java オプションまたは -sl dnet オプションを使って、Java クラスまたは .NET 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。

● 統合データベースに接続し、mlsrv11 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、次のコマンドを実行します。*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。

```
mlsrv11 -c "dsn=mobilink_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip -sl java (-cp c:¥MLdirect)
```

.NET の場合：

```
mlsrv11 -c "dsn=mobilink_db;uid=DBA;pwd=sql" -o serverOut.txt -v+ -dl -zu+ -x tcpip -sl dnet (-
MLAutoLoadPath=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・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 サーバはすべて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。

   a. SQL Anywhere データベースを作成するには、次のコマンドを実行します。

      ```
      dbinit -i -k remote1
      ```

      -i オプションと -k オプションは、それぞれ 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。

   b. データベース・サーバを起動するには、次のコマンドを実行します。

      ```
      dbeng11 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。

   次のコマンドを実行します。

   ```
   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 クライアントの詳細については、「Mobile Link 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。

   次のコマンドを実行します。

   ```
   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;
   ```

5. *orderResponses.txt* という空白のテキスト・ファイルを統合データベースと同じディレクトリに作成します。

♦ 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 行で実行します。

```
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. SQL Anywhere Mobile Link クライアント ウィンドウをすべて閉じます。
2. 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
   ```
3. 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。

   次のコマンドを実行します。

   ```
   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、mlsrv11 -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。
   a. 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
      ```

   b. 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*、*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。

チュートリアル：Microsoft Excel との同期

目次

Excel との同期のチュートリアルの概要

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、Microsoft Excel のスプレッドシートにあるデータを Mobile Link クライアントと同期する基本的な手順について説明します。このチュートリアルでは、例として Java 実装を使用します。

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。

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。

このチュートリアルでは、Microsoft Excel のスプレッドシートとリモート・クライアントの間でデータを同期する方法を示します。

必要なソフトウェア

- SQL Anywhere 11
- Java ソフトウェア開発キット
- Microsoft Excel 2000 以降

前提知識と経験

次の知識と経験が必要です。

- Java の知識
- Microsoft Excel 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 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
- Java を使用した Microsoft Excel ワークシートにあるデータへのアクセス

関連項目

- 「Mobile Link 同期の概要」 3 ページ
- 「同期の方法」 『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』

Mobile Link のコード・サンプルは、http://www.sybase.com/detail?id=1058600#319 にあります。

質問がある場合は Mobile Link ニュースグループ sybase.public.sqlanywhere.mobilink に投稿できます。

レッスン 1：Excel ワークシートの設定

このレッスンでは、Excel ワークシートを作成し、Microsoft Excel ドライバを使用して ODBC データ・ソースを定義します。Excel ワークシートには製品情報を格納します。

♦ **Excel データ・ソースを設定するには、次の手順に従います。**

1. 次の内容の Excel ワークシートを作成します。

order_id	comment_id	order_comment
1	2	出荷する宣伝用資料
1	3	必要な資料の詳細

2. デフォルトのワークシート名 **Sheet1** を **order_sheet** に変更します。
   a. **[Sheet1]** タブをダブルクリックします。
   b. **order_sheet** と入力します。
3. Excel ワークブックを保存します。

   このチュートリアルでは、*c:¥MLobjexcel* をサーバサイド・コンポーネントの作業ディレクトリとします。ワークブックを *order_central.xls* という名前でこの作業ディレクトリに保存します。
4. Microsoft Excel ドライバを使用して ODBC データ・ソースを作成します。
   a. **[スタート] - [プログラム] - [SQL Anywhere 11] - [ODBC アドミニストレータ]** をクリックします。
   b. **[ユーザー DSN]** タブをクリックします。
   c. **[追加]** をクリックします。
   d. **[Microsoft Excel Driver (*.xls)]** をクリックします。
   e. **[完了]** をクリックします。
   f. **[データ ソース名]** フィールドに **excel_datasource** と入力します。
   g. **[Select Workbook]** をクリックし、*c:¥MLobjexcel¥order_central.xls* (ワークシートが含まれるファイル) を選択します。
   h. **[OK]** をクリックします。
   i. **[OK]** をクリックします。

レッスン 2：Mobile Link 統合データベースの設定

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. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Sybase Central]** をクリックします
2. **[ツール] - [SQL Anywhere 11] - [データベースの作成]** をクリックします。
3. **[次へ]** をクリックします。
4. **[次へ]** をクリックします。
5. **[メイン・データベース・ファイルを保存]** フィールドに、データベースのファイル名およびパスを入力します。たとえば、*c:¥MLobjexcel¥MLconsolidated.db* と入力します。
6. **データベース作成ウィザード**の残りの指示に従い、デフォルト値をそのまま使用します。**[データベースへの接続]** ページで、**[最終切断後にデータベースを停止]** オプションをオフにします。
7. **[完了]** をクリックします。

   MLconsolidated 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・ソースを定義します。

SQL Anywhere 11 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. Sybase Central を起動します。

2. **[ツール] - [SQL Anywhere 11] - [ODBC アドミニストレータを開く]** をクリックします。
3. **[ユーザー DSN]** タブをクリックし、**[追加]** をクリックします。
4. **[名前]** リストで **[SQL Anywhere 11]** をクリックします。**[完了]** をクリックします。
5. **[SQL Anywhere 11 の ODBC 設定]** ウィンドウで、次の操作を行います。
   a. **[ODBC]** タブをクリックします。
   b. **[データ・ソース名]** フィールドに **mlexcel_db** と入力します。
   c. **[ログイン]** タブをクリックします。
   d. **[ユーザ ID]** フィールドに **DBA** と入力します。
   e. **[パスワード]** フィールドに **sql** と入力します。
   f. **[データベース]** タブをクリックします。
   g. **[サーバ名]** フィールドに **MLconsolidated** と入力します。
   h. **[データベース・ファイル]** フィールドに *c:¥MLobjexcel¥MLconsolidated.db* と入力します。
   i. **[OK]** をクリックします。
6. **[OK]** をクリックします。

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。

   コマンド・プロンプトで次のコマンドを実行します。

   ```
   dbisql -c "dsn=mlexcel_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 11 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。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=mlexcel_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 11¥* は、SQL Anywhere 11 の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。

   ```
   read "c:¥Program Files¥SQL Anywhere 11¥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 3：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 のロー・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。このチュートリアルでは、SQL ベースのスクリプト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。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=mlexcel_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 クラスを 「レッスン 4：Mobile Link のダイレクト・ロー・ハンドリングを使用する Java のクラスの作成」 237 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。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. handle_UploadData と handle_DownloadData の各同期イベント用の Java メソッドを登録します。

   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
   ```

   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・ダウンロードの処理」『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・データベース間でのローの分割」『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 4：Mobile Link のダイレクト・ロー・ハンドリングを使用する Java のクラスの作成

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 *order_central.xls* という Excel ワークシートに書き込みます。
- **SetDownload**　このメソッドは handle_DownloadData イベントに使用します。SetDownload は、Excel ワークシート *order_central.xls* に格納されたデータを取り出し、リモート・クライアントに送信します。

次の手順では、処理用メソッドを含む Java 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 コードの全リスト (Java)」 240 ページを参照してください。

♦ ダウンロード専用のダイレクト・ロー・ハンドリング用の Java のクラスを作成するには、次の手順に従います。

1. MobiLinkOrders というクラスを作成します。

   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 {
       // ...
   }
   ```

2. クラスレベルの DBConnectionContext インスタンスを宣言します。

   ```
   DBConnectionContext _cc;
   ```

   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。

   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。

   ```
   public MobiLinkOrders( DBConnectionContext cc ) {
       _cc = cc;
   }
   ```

4. 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 6：Mobile Link クライアントの設定」 243 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。

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。

```
// method for the handle_UploadData synchronization event
public void GetUpload( UploadData ut )
  throws SQLException, IOException {
  // get an UploadedTableData for OrderComments
  UploadedTableData orderCommentsTbl = ut.getUploadedTableByName("OrderComments");

  // get inserts uploaded by the MobiLink client
  ResultSet insertResultSet = orderCommentsTbl.getInserts();
  try {
    // connect to the excel worksheet through ODBC
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con = DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

    while( insertResultSet.next() ) {

      // get order comments
      int _commentID = insertResultSet.getInt("comment_id");
      int _orderID = insertResultSet.getInt("order_id");
      String _specialComments = insertResultSet.getString("order_comment");

      // execute an insert statement to add the order comment to the worksheet
      Statement st = con.createStatement();
      st.executeQuery( "insert into [order_sheet$]"
        + "(order_id, comment_id, order_comment) VALUES ("
        + Integer.toString(_orderID) + ", "
        + Integer.toString(_commentID) + ", '"
        + _specialComments + "')");

      st.close();
    }
    con.close();
  } catch(Exception ex) {
    System.err.print("Exception: ");
    System.err.println(ex.getMessage());
  }
  insertResultSet.close();
}
```

5. 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。

      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。

```
DownloadData download_d = _cc.getDownloadData();
DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );
```


> **注意**
> このテーブルは、「レッスン 6：Mobile Link クライアントの設定」 243 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。

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。

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。

```
PreparedStatement download_upserts = download_td.getUpsertPreparedStatement();
```

c. 各ローのダウンロード・データを設定します。

次の例では、*order_central.xls* ワークシートを参照して、Mobile Link ダウンロードにデータを追加しています。

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。

```
try {
  // connect to the excel worksheet through ODBC
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con = DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

  // retrieve all the rows in the worksheet
  Statement st = con.createStatement();
  ResultSet Excel_rs = st.executeQuery( "select * from [order_sheet$]" );

  while (Excel_rs.next()) {
    // retrieve the row data
    int Excel_comment_id = Excel_rs.getInt(1);
    int Excel_order_id = Excel_rs.getInt(2);
    String Excel_comment = Excel_rs.getString(3);

    // add the Excel data to the MobiLink download.
    download_upserts.setInt( 1, Excel_comment_id );
    download_upserts.setInt( 2, Excel_order_id );
    download_upserts.setString( 3, Excel_comment );
    download_upserts.executeUpdate();
  }
  // close the excel result set, statement, and connection.
  Excel_rs.close();
  st.close();
  con.close();
} catch(Exception ex) {
  System.err.print("Exception: ");
  System.err.println(ex.getMessage());
}
```

d. 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。

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。

```
download_upserts.close();
```

6. Java コードを *MobiLinkOrders.java* という名前で作業ディレクトリ *c:¥MLobjexcel* に保存します。
7. クラス・ファイルをコンパイルします

a. Java 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。

b. Java 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。

install-dir¥Java にある *mlscript.jar* を参照する必要があります。次のコマンドを実行して Java クラスをコンパイルします。*c:¥Program Files¥SQL Anywhere 11¥* は SQL Anywhere 11 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 11¥java¥mlscript.jar" MobiLinkOrders.java
```

詳細情報

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。

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 コードの全リスト (Java)

このチュートリアルで使用している Java の MobiLinkOrders クラスの全コードを次に示します。

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class MobiLinkOrders {

  // class level DBConnectionContext
  DBConnectionContext _cc;

  public MobiLinkOrders( DBConnectionContext cc )
    throws IOException, FileNotFoundException {
    // declare a class-level DBConnectionContext
    _cc = cc;
  }


  //  method for the handle_UploadData synchronization event
  public void GetUpload( UploadData ut )
    throws SQLException, IOException {
    //  get an UploadedTableData for OrderComments
    UploadedTableData orderCommentsTbl = ut.getUploadedTableByName("OrderComments");

    // get inserts uploaded by the MobiLink client
    ResultSet insertResultSet = orderCommentsTbl.getInserts();
    try {
      // connect to the excel worksheet through ODBC
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

      while( insertResultSet.next() ) {
        // get order comments
        int _commentID = insertResultSet.getInt("comment_id");
        int _orderID = insertResultSet.getInt("order_id");
        String _specialComments = insertResultSet.getString("order_comment");

        // execute an insert statement to add the order comment to the worksheet
```



```
        Statement st = con.createStatement();
        st.executeQuery( "insert into [order_sheet$]"
          + "(order_id, comment_id, order_comment) VALUES ("
          + Integer.toString(_orderID) + ", "
          + Integer.toString(_commentID) + ", '"
          + _specialComments + "')");
        st.close();
      }
      con.close();
    } catch(Exception ex) {
      System.err.print("Exception: ");
      System.err.println(ex.getMessage());
    }
    insertResultSet.close();
  }


  public void SetDownload() throws SQLException, IOException {

    DownloadData download_d = _cc.getDownloadData();

    DownloadTableData download_td = download_d.getDownloadTableByName( "OrderComments" );

    // Prepared statement to compile upserts (inserts or updates).
    PreparedStatement download_upserts = download_td.getUpsertPreparedStatement();
    try {
      // connect to the excel worksheet through ODBC
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con = DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

      // retrieve all the rows in the worksheet
      Statement st = con.createStatement();
      ResultSet Excel_rs = st.executeQuery( "select * from [order_sheet$]" );

      while (Excel_rs.next()) {
        // retrieve the row data
        int Excel_comment_id = Excel_rs.getInt(1);
        int Excel_order_id = Excel_rs.getInt(2);
        String Excel_comment = Excel_rs.getString(3);

        // add the Excel data to the MobiLink download.
        download_upserts.setInt( 1, Excel_comment_id );
        download_upserts.setInt( 2, Excel_order_id );
        download_upserts.setString( 3, Excel_comment );
        download_upserts.executeUpdate();
      }
      // close the excel result set, statement, and connection.
      Excel_rs.close();
      st.close();
      con.close();
    } catch (Exception ex) {
      System.err.print("Exception: ");
      System.err.println(ex.getMessage());
    }

    download_upserts.close();
  }

}
```

レッスン 5：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 (mlsrv11) を起動して統合データベースに接続し、-sl java オプションを使用してそれぞれ Java 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。

● 統合データベースに接続し、mlsrv11 のコマンド・ラインで Java クラスをロードします。

次のコマンドを実行します。*c:¥MLobjexcel* 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。

```
mlsrv11 -c "dsn=mlexcel_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl java (-cp c:¥MLobjexcel)
```

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・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 6：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 サーバはすべて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、次のコマンドを実行します。

     ```
     dbinit -l -k remote1
     ```

     -l オプションと -k オプションは、それぞれ 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・サーバを起動するには、次のコマンドを実行します。

     ```
     dbeng11 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。

   次のコマンドを実行します。

   ```
   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 クライアントの詳細については、「Mobile Link 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 7：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。

   次のコマンドを実行します。

   ```
   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 行で実行します。

```
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 の処理によってコメントが *order_central.xls* ワークシートに挿入されました。*order_central.xls* ワークシートに格納された情報がクライアントにダウンロードされます。

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、mlsrv11 -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。

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

1. 以下のアプリケーションのインスタンスをすべて閉じます。
   - Interactive SQL
   - Microsoft Excel
2. Excel ワークブック *order_central.xls* を削除します。
3. SQL Anywhere、Mobile Link、同期クライアントの各ウィンドウを閉じます。
4. チュートリアルに関連するすべての DSN を削除します。
   a. ODBC アドミニストレータを起動します。
      コマンド・プロンプトで次のコマンドを入力します。

      ```
      odbcad32
      ```

   b. excel_datasource と mlexcel_db の各データ・ソースを削除します。
5. 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。
   a. 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。
   b. *MLconsolidated.db*、*MLconsolidated.log*、*remote1.db*、*remote1.log* を削除します。

詳細情報

Mobile Link 同期の詳細については、「Mobile Link 同期の概要」 3 ページを参照してください。

Mobile Link クライアントの詳細については、「Mobile Link 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。

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。

チュートリアル：XML との同期

目次

XML との同期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。このチュートリアルでは、例として Java 実装を使用します。

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。

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。

このチュートリアルでは、XML ファイルとリモート・クライアントの間でデータを同期する方法を示します。

必要なソフトウェア

- SQL Anywhere 11
- Java ソフトウェア開発キット
- XML DOM ライブラリ

前提知識と経験

次の知識と経験が必要です。

- Java の知識
- XML の知識
- XML DOM 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 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 同期の概要」 3 ページ
- 「同期の方法」 『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』

Mobile Link のコード・サンプルは、http://www.sybase.com/detail?id=1058600#319 にあります。

質問がある場合は Mobile Link ニュースグループ sybase.public.sqlanywhere.mobilink に投稿できます。

Java での XML DOM の使用方法の詳細については、「Java API for XML Processing」(JAXP) を参照してください。

レッスン 1：XML データ・ソースの設定

この項では、XML ファイルを作成します。XML ファイルには受注情報が格納されます。

♦ **XML データ・ソースを設定するには、次の手順に従います。**

1. 次の内容の XML ファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<orders></orders>
```

2. XML ファイルを保存します。

   このチュートリアルでは、*c:¥MLobjxml* をサーバサイド・コンポーネントの作業ディレクトリとします。XML ファイルを *order_comments.xml* という名前でこの作業ディレクトリに保存します。

レッスン 2：Mobile Link 統合データベースの設定

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。

♦ SQL Anywhere RDBMS を作成するには、次の手順に従います。

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Sybase Central]** を選択します。
2. Sybase Central で、**[ツール] - [SQL Anywhere 11] - [データベースの作成]** を選択します。
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. **[メイン・データベース・ファイルを保存]** フィールドに、データベースのファイル名およびパスを入力します。たとえば、*c:¥MLobjxml¥MLconsolidated.db* と入力します。**[次へ]** をクリックします。
6. **データベース作成ウィザード**の残りの指示に従い、デフォルト値をそのまま使用します。**[データベースへの接続]** ページで、**[最終切断後にデータベースを停止]** オプションをオフにします。
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・ソースを定義するには、次の手順に従います。

1. Sybase Central の **[ツール] - [SQL Anywhere 11] - [ODBC アドミニストレータを開く]** を選択します。
2. **[ユーザー DSN]** タブをクリックし、**[追加]** をクリックします。
3. **[名前]** リストで **[SQL Anywhere 11]** をクリックします。**[完了]** をクリックします。
4. **[SQL Anywhere 11 の ODBC 設定]** ウィンドウで、次の操作を行います。
   a. **[ODBC]** タブをクリックします。

b. **[データ・ソース名]** フィールドに **mlxml_db** と入力します。
c. **[ログイン]** タブをクリックします。
d. **[ユーザ ID]** フィールドに **DBA** と入力します。
e. **[パスワード]** フィールドに **sql** と入力します。
f. **[データベース]** タブをクリックします。
g. **[サーバ名]** フィールドに **MLconsolidated** と入力します。
h. **[データベース・ファイル]** フィールドに *c:¥MLobjxml¥MLconsolidated.db* と入力します。
i. **[OK]** をクリックします。

5. **[OK]** をクリックします。

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、次のコマンドを実行します。

     ```
     dbisql -c "dsn=mlxml_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 11 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。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=mlxml_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 11¥* は、SQL Anywhere 11 の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。

   ```
   read "c:¥Program Files¥SQL Anywhere 11¥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 3：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 のロー・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。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=mlxml_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 クラスを 「レッスン 4：Mobile Link のダイレクト・ロー・ハンドリングを使用する Java のクラスの作成」 259 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。

   次のコマンドを実行します。

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. handle_UploadData と handle_DownloadData の各同期イベント用の Java メソッドを登録します。

   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' );

   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・ダウンロードの処理」 『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・データベース間でのローの分割」『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 4：Mobile Link のダイレクト・ロー・ハンドリングを使用する Java のクラスの作成

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 XML ファイルに書き込みます。

次の手順では、処理用メソッドを含む Java 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 コードの全リスト (Java)」 263 ページを参照してください。

♦ ダウンロード専用のダイレクト・ロー・ハンドリング用の Java のクラスを作成するには、次の手順に従います。

1. Java を使用して、MobilinkOrders というクラスを作成します。

   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。

```
//Mobilink imports
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

//XML parser
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

//DOM Objects
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

//For writing XML objects
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

public class MobiLinkOrders {
    // ...
}
```

2. クラスレベルの DBConnectionContext インスタンスおよび Document インスタンスを宣言します。Document クラスは、XML 文書をオブジェクトとして表します。

```
DBConnectionContext _cc;
 Document _doc;
```

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。

   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。

   ```
   public MobiLinkOrders( DBConnectionContext cc ) {
     _cc = cc;
   }
   ```

4. GetUpload() メソッドを作成します。

   a. メソッドの宣言を作成します。

      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。

      ```
      // method for the handle_UploadData synchronization event
      public void GetUpload( UploadData ut ) throws SQLException, IOException {
      ```

   b. Mobile Link クライアントからアップロードされた挿入をすべて取得します。

      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。

      ```
      //  get an UploadedTableData for OrderComments
        UploadedTableData orderCommentsTbl =
          ut.getUploadedTableByName("OrderComments");

        // get inserts uploaded by the MobiLink client
        ResultSet insertResultSet = orderCommentsTbl.getInserts();
      ```

   c. 既存の XML ファイル **order_comments.xml** を読み込みます。

      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。

      ```
          readDom("order_comments.xml");
      ```

   d. 挿入された各ローを XML ファイルに追加します。

      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。

      ```
          // Write out each insert in the XML file
          while( insertResultSet.next() ) {
            buildXML(insertResultSet);
          }
      ```

   e. XML ファイルを書き込みます。

      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。

      ```
        writeXML();
      ```

   f. ResultSet を閉じます。

      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。

      ```
          // Close the result set of uploaded inserts
          insertResultSet.close();
      ```

5. buildXML メソッドを作成します。

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。

```
private void buildXML( ResultSet rs ) throws SQLException {
  int order_id = rs.getInt(1);
  int comment_id = rs.getInt(2);
  String order_comment = rs.getString(3);

  //Create the comment object to be added to the XML file
  Element comment = _doc.createElement("comment");
  comment.setAttribute("id", Integer.toString(comment_id));
  comment.appendChild(_doc.createTextNode(order_comment));

  //Get the root element (orders)
  Element root = _doc.getDocumentElement();

  //get each individual order
  NodeList rootChildren = root.getChildNodes();
  for(int i = 0; i < rootChildren.getLength(); i++) {
    //if the order exists, add the comment to the order
    Node n = rootChildren.item(i);
    if(n.getNodeType() == Node.ELEMENT_NODE) {
      Element e = (Element) n;
      int idIntVal = Integer.parseInt(e.getAttribute("id"));
      if(idIntVal == order_id) {
        e.appendChild(comment);
        //The comment has been added to the file, so exit the function
        return;
      }
    }
  }

  //if the order did not exist already, create it
  Element order = _doc.createElement("order");
  order.setAttribute("id", Integer.toString(order_id));
  //add the comment to the new order
  order.appendChild(comment);
  root.appendChild(order);
}
```

6. writeXML メソッドを作成します。

   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。

```
private void writeXML() {

  try {

    // Use a Transformer for output
    TransformerFactory tFactory = TransformerFactory.newInstance();
    Transformer transformer = tFactory.newTransformer();

    //The XML source is _doc
    DOMSource source = new DOMSource(_doc);
    //write the xml data to order_comments.xml
    StreamResult result = new StreamResult(new File("order_comments.xml"));
    transformer.transform(source, result);

  } catch (TransformerConfigurationException tce) {
    // Error generated by the parser
    System.out.println ("¥n** Transformer Factory error");
    System.out.println("   " + tce.getMessage() );

    // Use the contained exception, if any
    Throwable x = tce;
```

```
            if (tce.getException() != null) x = tce.getException();
            x.printStackTrace();

          } catch (TransformerException te) {
            // Error generated by the parser
            System.out.println ("¥n** Transformation error");
            System.out.println("   " + te.getMessage() );

            // Use the contained exception, if any
            Throwable x = te;
            if (te.getException() != null) x = te.getException();
            x.printStackTrace();

        }

        }
```

7. readDOM メソッドを作成します。

   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。

```
    private void readDom(String filename) {
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        try {
          //parse the Document data into _doc
          DocumentBuilder builder = factory.newDocumentBuilder();
          _doc = builder.parse( new File(filename) );

        } catch (SAXException sxe) {
          // Error generated during parsing)
          Exception x = sxe;
          if (sxe.getException() != null) x = sxe.getException();
          x.printStackTrace();

        } catch (ParserConfigurationException pce) {
          // Parser with specified options can't be built
          pce.printStackTrace();

        } catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
        }

    }
```

8. Java コードを *MobiLinkOrders.java* という名前で作業ディレクトリ *c:¥MLobjxml* に保存します。
9. クラス・ファイルをコンパイルします
   a. Java 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。
   b. Java 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、*install-dir¥Java* にある *mlscript.jar* を参照する必要があります。また、XML DOM ライブラリが正しくインストールされていることを確認する必要があります。次のコマンドを実行して Java クラスをコンパイルします。*c:¥Program Files¥SQL Anywhere 11¥* は SQL Anywhere 11 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 11¥java¥mlscript.jar" MobiLinkOrders.java
```


詳細情報

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。

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 コードの全リスト (Java)

このチュートリアルで使用している Java の MobiLinkOrders クラスの全コードを次に示します。

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

// For write operation
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

public class MobiLinkOrders {

  // class level DBConnectionContext
  DBConnectionContext _cc;
  Document _doc;

  public MobiLinkOrders( DBConnectionContext cc ) throws IOException, FileNotFoundException {
    // declare a class-level DBConnectionContext
    _cc = cc;
  }


  //  method for the handle_UploadData synchronization event
  public void GetUpload( UploadData ut ) throws SQLException, IOException {
    // Get an UploadedTableData for the remote table
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("OrderComments");

    // Get inserts uploaded by the MobiLink client
    // as a java.sql.ResultSet
    ResultSet insertResultSet = remoteOrdersTable.getInserts();

    readDom("order_comments.xml");

    // Write out each insert in the XML file
    while( insertResultSet.next() ) {
```

```
      buildXML(insertResultSet);
    }
    writeXML();

    // Close the result set of uploaded inserts
    insertResultSet.close();

  }

  private void buildXML( ResultSet rs ) throws SQLException {
    int order_id = rs.getInt(1);
    int comment_id = rs.getInt(2);
    String order_comment = rs.getString(3);

    //Create the comment object to be added to the XML file
    Element comment = _doc.createElement("comment");
    comment.setAttribute("id", Integer.toString(comment_id));
    comment.appendChild(_doc.createTextNode(order_comment));

    //Get the root element (orders)
    Element root = _doc.getDocumentElement();

    //get each individual order
    NodeList rootChildren = root.getChildNodes();
    for(int i = 0; i < rootChildren.getLength(); i++) {
      //if the order exists, add the comment to the order
      Node n = rootChildren.item(i);
      if(n.getNodeType() == Node.ELEMENT_NODE) {
        Element e = (Element) n;
        int idIntVal = Integer.parseInt(e.getAttribute("id"));
        if(idIntVal == order_id) {
          e.appendChild(comment);
          //The comment has been added to the file, so exit the function
          return;
        }
      }
    }

    //if the order did not exist already, create it
    Element order = _doc.createElement("order");
    order.setAttribute("id", Integer.toString(order_id));
    //add the comment to the new order
    order.appendChild(comment);
    root.appendChild(order);
  }

  private void writeXML() {

    try {

      // Use a Transformer for output
      TransformerFactory tFactory = TransformerFactory.newInstance();
      Transformer transformer = tFactory.newTransformer();

      //The XML source is _doc
      DOMSource source = new DOMSource(_doc);
      //write the xml data to order_comments.xml
      StreamResult result = new StreamResult(new File("order_comments.xml"));
      transformer.transform(source, result);

    } catch (TransformerConfigurationException tce) {
      // Error generated by the parser
      System.out.println ("¥n** Transformer Factory error");
      System.out.println("   " + tce.getMessage() );
```



```
        // Use the contained exception, if any
        Throwable x = tce;
        if (tce.getException() != null) x = tce.getException();
        x.printStackTrace();

      } catch (TransformerException te) {
        // Error generated by the parser
        System.out.println ("¥n** Transformation error");
        System.out.println("   " + te.getMessage() );

        // Use the contained exception, if any
        Throwable x = te;
        if (te.getException() != null) x = te.getException();
        x.printStackTrace();

    }

     }

     private void readDom(String filename) {
       DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        try {
          //parse the Document data into _doc
          DocumentBuilder builder = factory.newDocumentBuilder();
          _doc = builder.parse( new File(filename) );

        } catch (SAXException sxe) {
          // Error generated during parsing)
          Exception x = sxe;
          if (sxe.getException() != null) x = sxe.getException();
          x.printStackTrace();

        } catch (ParserConfigurationException pce) {
          // Parser with specified options can't be built
          pce.printStackTrace();

        } catch (IOException ioe) {
          // I/O error
          ioe.printStackTrace();
        }

     }

    }
```

レッスン 5：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 (mlsrv11) を起動して統合データベースに接続し、-sl java オプションを使用して Java 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。

● 統合データベースに接続し、mlsrv11 のコマンド・ラインで Java クラスをロードします。

次のコマンドを実行します。*c:¥MLobjxml* 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。

```
mlsrv11 -c "dsn=mlxml_db" -o c:¥MLobjxml¥serverOut.txt -v+ -dl -zu+ -x tcpip  -sl java (-cp c:
¥MLobjxml)
```

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・オプション」『Mobile Link - サーバ管理』を参照してください。

Java クラスのロードの詳細については、「-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 6：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 サーバはすべて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。

   a. SQL Anywhere データベースを作成するには、次のコマンドを実行します。

      ```
      dbinit -i -k remote1
      ```

      -i オプションと -k オプションは、それぞれ 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。

   b. データベース・サーバを起動するには、次のコマンドを実行します。

      ```
      dbeng11 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。

   次のコマンドを実行します。

   ```
   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 (
      order_id          integer not null,
      comment_id        integer not null,
      order_comment     varchar (255),
      primary key(comment_id),
   ```



```
    foreign key (order_id) references
     RemoteOrders (order_id)
)
```

5. 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 クライアントの詳細については、「Mobile Link 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 7：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。

   次のコマンドを実行します。

   ```
   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実行します。*c:¥MLobjxml* 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。

```
dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -e scn=on -o c:¥MLobjxml¥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 の処理によってコメントが XML ファイルに挿入されました。

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。

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

1. 以下のアプリケーションのインスタンスをすべて閉じます。
   - Interactive SQL
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。
   a. ODBC アドミニストレータを起動します。
      コマンド・プロンプトで次のコマンドを入力します。

      ```
      odbcad32
      ```

   b. mlxml_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。
   a. 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。
   b. *MLconsolidated.db*、*MLconsolidated.log*、*remote1.db*、*remote1.log* を削除します。

詳細情報

Mobile Link 同期の詳細については、「Mobile Link 同期の概要」 3 ページを参照してください。

Mobile Link クライアントの詳細については、「Mobile Link 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。

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。

用語解説

用語解説

Adaptive Server Anywhere (ASA)

SQL Anywhere Studio のリレーショナル・データベース・サーバ・コンポーネントであり、主に、モバイル環境と埋め込み環境、または小規模および中規模のビジネス用のサーバとして使用されます。バージョン 10.0.0 で、Adaptive Server Anywhere は SQL Anywhere サーバに、SQL Anywhere Studio は SQL Anywhere にそれぞれ名前が変更されました。

参照：「SQL Anywhere」 282 ページ。

Carrier

Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに保存される Mobile Link オブジェクトで、システム起動同期で使用される通信業者に関する情報が含まれます。

参照：「サーバ起動同期」 287 ページ。

DB 領域

データ用の領域をさらに作成する追加のデータベース・ファイルです。1 つのデータベースは 13 個までのファイルに保管されます (初期ファイル 1 つと 12 の DB 領域)。各テーブルは、そのインデックスとともに、単一のデータベース・ファイルに含まれている必要があります。CREATE DBSPACE という SQL コマンドで、新しいファイルをデータベースに追加できます。

参照：「データベース・ファイル」 291 ページ。

DBA 権限

ユーザに、データベース内の管理作業を許可するレベルのパーミッションです。DBA ユーザにはデフォルトで DBA 権限が与えられています。

参照：「データベース管理者 (DBA)」 291 ページ。

EBF

Express Bug Fix の略です。Express Bug Fix は、1 つ以上のバグ・フィックスが含まれる、ソフトウェアのサブセットです。これらのバグ・フィックスは、更新のリリース・ノートにリストされます。バグ・フィックス更新を適用できるのは、同じバージョン番号を持つインストール済みのソフトウェアに対してだけです。このソフトウェアについては、ある程度のテストが行われているとはいえ、完全なテストが行われたわけではありません。自分自身でソフトウェアの妥当性を確かめるまでは、アプリケーションとともにこれらのファイルを配布しないでください。

Embedded SQL

C プログラム用のプログラミング・インタフェースです。SQL Anywhere の Embedded SQL は ANSI と IBM 規格に準拠して実装されています。

FILE

SQL Remote のレプリケーションでは、レプリケーション・メッセージのやりとりのために共有ファイルを使うメッセージ・システムのことです。これは特定のメッセージ送信システムに頼らずにテストやインストールを行うのに便利です。

参照：「レプリケーション」 299 ページ。

grant オプション

他のユーザにパーミッションを許可できるレベルのパーミッションです。

iAnywhere JDBC ドライバ

iAnywhere JDBC ドライバでは、pure Java である jConnect JDBC ドライバに比べて何らかの有利なパフォーマンスや機能を備えた JDBC ドライバが提供されます。ただし、このドライバは pure Java ソリューションではありません。iAnywhere JDBC ドライバは一般に推奨されるドライバです。

参照：

- 「JDBC」 279 ページ
- 「jConnect」 279 ページ

InfoMaker

レポート作成とデータ管理用のツールです。洗練されたフォーム、レポート、グラフ、クロスタブ、テーブルを作成できます。また、これらを基本的な構成要素とするアプリケーションも作成できます。

Interactive SQL

データベース内のデータの変更や問い合わせ、データベース構造の修正ができる、SQL Anywhere のアプリケーションです。Interactive SQL では、SQL 文を入力するためのウィンドウ枠が表示されます。また、クエリの進捗情報や結果セットを返すウィンドウ枠も表示されます。

JAR ファイル

Java アーカイブ・ファイルです。Java のアプリケーションで使用される 1 つ以上のパッケージの集合からなる圧縮ファイルのフォーマットです。Java プログラムをインストールしたり実行したりするのに必要なリソースが 1 つの圧縮ファイルにすべて収められています。

Java クラス

Java のコードの主要な構造単位です。これはプロシージャや変数の集まりで、すべてがある一定のカテゴリに関連しているためグループ化されたものです。

jConnect

JavaSoft JDBC 標準を Java で実装したものです。これにより、Java 開発者は多層／異機種環境でもネイティブなデータベース・アクセスができます。iAnywhere JDBC ドライバは一般に推奨されるドライバです。

参照：

- 「JDBC」 279 ページ
- 「iAnywhere JDBC ドライバ」 278 ページ

JDBC

Java Database Connectivity の略です。Java アプリケーションからリレーショナル・データにアクセスすることを可能にする SQL 言語プログラミング・インタフェースです。推奨 JDBC ドライバは、iAnywhere JDBC ドライバです。

参照：

- 「jConnect」 279 ページ
- 「iAnywhere JDBC ドライバ」 278 ページ

Listener

Mobile Link サーバ起動同期に使用される、dblsn という名前のプログラムです。Listener はリモート・デバイスにインストールされ、Push 通知を受け取ったときにデバイス上でアクションが開始されるように設定されます。

参照：「サーバ起動同期」 287 ページ。

LTM

LTM (Log Transfer Manager) は、Replication Agent とも呼ばれます。Replication Server と併用することで、LTM はデータベース・トランザクション・ログを読み込み、コミットされた変更を Sybase Replication Server に送信します。

参照：「Replication Server」 282 ページ。

Mobile Link

Ultra Light と SQL Anywhere のリモート・データベースを統合データベースと同期させるために設計された、セッションベース同期テクノロジです。

参照：

- 「統合データベース」 306 ページ
- 「同期」 306 ページ
- 「Ultra Light」 283 ページ

Mobile Link クライアント

2 種類の Mobile Link クライアントがあります。SQL Anywhere リモート・データベース用の Mobile Link クライアントは、dbmlsync コマンド・ライン・ユーティリティです。Ultra Light リモート・データベース用の Mobile Link クライアントは、Ultra Light ランタイム・ライブラリに組み込まれています。

Mobile Link サーバ

Mobile Link 同期を実行する、mlsrv11 という名前のコンピュータ・プログラムです。

Mobile Link システム・テーブル

Mobile Link の同期に必要なシステム・テーブルです。Mobile Link 設定スクリプトによって、Mobile Link 統合データベースにインストールされます。

Mobile Link モニタ

Mobile Link の同期をモニタするためのグラフィカル・ツールです。

Mobile Link ユーザ

Mobile Link ユーザは、Mobile Link サーバに接続するのに使用されます。Mobile Link ユーザをリモート・データベースに作成し、統合データベースに登録します。Mobile Link ユーザ名はデータベース・ユーザ名から完全に独立しています。

Notifier

Mobile Link サーバ起動同期に使用されるプログラムです。Notifier は Mobile Link サーバに統合されており、統合データベースに Push 要求がないか確認し、Push 通知を送信します。

参照：

- 「サーバ起動同期」 287 ページ
- 「Listener」 279 ページ

ODBC

Open Database Connectivity の略です。データベース管理システムに対する Windows の標準的なインタフェースです。ODBC は、SQL Anywhere がサポートするインタフェースの 1 つです。

ODBC アドミニストレータ

Windows オペレーティング・システムに付属している Microsoft のプログラムです。ODBC データ・ソースの設定に使用します。

ODBC データ・ソース

ユーザが ODBC からアクセスするデータと、そのデータにアクセスするために必要な情報の仕様です。

PDB

Palm のデータベース・ファイルです。

PowerDesigner

データベース・モデリング・アプリケーションです。これを使用すると、データベースやデータ・ウェアハウスの設計に対する構造的なアプローチが可能となります。SQL Anywhere には、PowerDesigner の Physical Data Model コンポーネントが付属します。

PowerJ

Java アプリケーション開発に使用する Sybase 製品です。

Push 通知

QAnywhere では、メッセージ転送を開始するよう QAnywhere クライアントに対して指示するために、サーバから QAnywhere クライアントに配信される特殊なメッセージです。Mobile Link サーバ起動同期では、Push 要求データや内部情報を含むデバイスに Notifer から配信される特殊なメッセージです。

参照：

- 「QAnywhere」 281 ページ
- 「サーバ起動同期」 287 ページ

Push 要求

Mobile Link サーバ起動同期において、Push 通知をデバイスに送信する必要があるかどうかを判断するために Notifier が確認する、結果セット内の値のローです。

参照：「サーバ起動同期」 287 ページ。

QAnywhere

アプリケーション間メッセージング (モバイル・デバイス間メッセージングやモバイル・デバイスとエンタープライズの間のメッセージングなど) を使用すると、モバイル・デバイスや無線デバイスで動作しているカスタム・プログラムと、集中管理されているサーバ・アプリケーションとの間で通信できます。

QAnywhere Agent

QAnywhere では、クライアント・デバイス上で動作する独立のプロセスのことです。クライアント・メッセージ・ストアをモニタリングし、メッセージを転送するタイミングを決定します。

REMOTE DBA 権限

SQL Remote では、Message Agent (dbremote) で必要なパーミッションのレベルを指します。Mobile Link では、SQL Anywhere 同期クライアント (dbmlsync) で必要なパーミッションのレベルを指します。Message Agent (dbremote) または同期クライアントがこの権限のあるユーザとして接続した場合、DBA のフル・アクセス権が与えられます。Message Agent (dbremote) または同期クライアント (dbmlsync) から接続しない場合、このユーザ ID にはパーミッションは追加されません。

参照：「DBA 権限」 277 ページ。

Replication Agent

参照：「LTM」 279 ページ。

Replication Server

SQL Anywhere と Adaptive Server Enterprise で動作する、Sybase による接続ベースのレプリケーション・テクノロジです。Replication Server は、少数のデータベース間でほぼリアルタイムのレプリケーションを行うことを目的に設計されています。

参照：「LTM」 279 ページ。

SQL

リレーショナル・データベースとの通信に使用される言語です。SQL は ANSI により標準が定義されており、その最新版は SQL-2003 です。SQL は、公認されてはいませんが、Structured Query Language の略です。

SQL Anywhere

SQLAnywhere のリレーショナル・データベース・サーバ・コンポーネントであり、主に、モバイル環境と埋め込み環境、または小規模および中規模のビジネス用のサーバとして使用されます。SQL Anywhere は、SQL Anywhere RDBMS、Ultra Light RDBMS、Mobile Link 同期ソフトウェア、その他のコンポーネントを含むパッケージの名前でもあります。

SQL Remote

統合データベースとリモート・データベース間で双方向レプリケーションを行うための、メッセージベースのデータ・レプリケーション・テクノロジです。統合データベースとリモート・データベースは、SQL Anywhere である必要があります。

SQL ベースの同期

Mobile Link では、Mobile Link イベントを使用して、テーブル・データを Mobile Link でサポートされている統合データベースに同期する方法のことです。SQL ベースの同期では、SQL を直接使用したり、Java と .NET 用の Mobile Link サーバ API を使用して SQL を返すことができます。

SQL 文

DBMS に命令を渡すために設計された、SQL キーワードを含む文字列です。

参照：

- 「スキーマ」 289 ページ
- 「SQL」 282 ページ
- 「データベース管理システム (DBMS)」 291 ページ

Sybase Central

SQL Anywhere データベースのさまざまな設定、プロパティ、ユーティリティを使用できる、グラフィカル・ユーザ・インタフェースを持つデータベース管理ツールです。Mobile Link などの他の iAnywhere 製品を管理する場合にも使用できます。

SYS

システム・オブジェクトの大半を所有する特別なユーザです。一般のユーザは SYS でログインできません。

Ultra Light

小型デバイス、モバイル・デバイス、埋め込みデバイス用に最適化されたデータベースです。対象となるプラットフォームとして、携帯電話、ポケットベル、パーソナル・オーガナイザなどが挙げられます。

Ultra Light ランタイム

組み込みの Mobile Link 同期クライアントを含む、インプロセス・リレーショナル・データベース管理システムです。Ultra Light ランタイムは、Ultra Light の各プログラミング・インタフェースで使用されるライブラリと、Ultra Light エンジンの両方に含まれます。

Windows

Windows Vista、Windows XP、Windows 200x などの、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 CE

「Windows Mobile」 283 ページを参照してください。

Windows Mobile

Microsoft がモバイル・デバイス用に開発したオペレーティング・システムのファミリです。

アーティクル

Mobile Link または SQL Remote では、テーブル全体もしくはテーブル内のカラムとローのサブセットを表すデータベース・オブジェクトを指します。アーティクルの集合がパブリケーションです。

参照：

- 「レプリケーション」 299 ページ
- 「パブリケーション」 294 ページ

アップロード

同期中に、リモート・データベースから統合データベースにデータが転送される段階です。

アトミックなトランザクション

完全に処理されるかまったく処理されないことが保証される 1 つのトランザクションです。エラーによってアトミックなトランザクションの一部が処理されなかった場合は、データベースが一貫性のない状態になるのを防ぐために、トランザクションがロールバックされます。

アンロード

データベースをアンロードすると、データベースの構造かデータ、またはその両方がテキスト・ファイルにエクスポートされます (構造は SQL コマンド・ファイルに、データはカンマ区切りの ASCII ファイルにエクスポートされます)。データベースのアンロードには、アンロード・ユーティリティを使用します。

また、UNLOAD 文を使って、データから抜粋した部分だけをアンロードできます。

イベント・モデル

Mobile Link では、同期を構成する、begin_synchronization や download_cursor などの一連のイベントのことです。イベントは、スクリプトがイベント用に作成されると呼び出されます。

インクリメンタル・バックアップ

トランザクション・ログ専用のバックアップです。通常、フル・バックアップとフル・バックアップの間に使用します。

参照：「トランザクション・ログ」 293 ページ。

インデックス

ベース・テーブルにある 1 つ以上のカラムに関連付けられた、キーとポインタのソートされたセットです。テーブルの 1 つ以上のカラムにインデックスが設定されていると、パフォーマンスが向上します。

ウィンドウ

分析関数の実行対象となるローのグループです。ウィンドウには、ウィンドウ定義内のグループ化指定に従って分割されたデータの、1 つ、複数、またはすべてのローが含まれます。ウィンドウは、入力の現在のローについて計算を実行する必要があるローの数や範囲を含むように移動します。ウィンドウ構成の主な利点は、追加のクエリを実行しなくても、結果をグループ化して分析する機会が増えることです。

エージェント ID

参照：「クライアント・メッセージ・ストア ID」 286 ページ。

エンコード

文字コードとも呼ばれます。エンコードは、文字セットの各文字が情報の 1 つまたは複数のバイトにマッピングされる方法のことで、一般的に 16 進数で表現されます。UTF-8 はエンコードの例です。

参照：

- 「文字セット」 307 ページ
- 「コード・ページ」 287 ページ
- 「照合」 304 ページ

オブジェクト・ツリー

Sybase Central では、データベース・オブジェクトの階層を指します。オブジェクト・ツリーの最上位には、現在使用しているバージョンの Sybase Central がサポートするすべての製品が表示されます。それぞれの製品を拡張表示すると、オブジェクトの下位ツリーが表示されます。

参照：「Sybase Central」 283 ページ。

カーソル

結果セットへの関連付けに名前を付けたもので、プログラミング・インタフェースからローにアクセスしたり更新したりするときに使用します。SQL Anywhere では、カーソルはクエリ結果内で前方や後方への移動をサポートします。カーソルは、カーソル結果セット (通常 SELECT 文で定義される) とカーソル位置の 2 つの部分から構成されます。

参照：

- 「カーソル結果セット」 286 ページ
- 「カーソル位置」 285 ページ

カーソル位置

カーソル結果セット内の 1 つのローを指すポインタ。

参照：

- 「カーソル」 285 ページ
- 「カーソル結果セット」 286 ページ

カーソル結果セット

カーソルに関連付けられたクエリから生成されるローのセットです。

参照：

- 「カーソル」 285 ページ
- 「カーソル位置」 285 ページ

クエリ

データベースのデータにアクセスしたり、そのデータを操作したりする SQL 文や SQL 文のグループです。

参照：「SQL」 282 ページ。

クライアント／サーバ

あるアプリケーション (クライアント) が別のアプリケーション (サーバ) に対して情報を送受信するソフトウェア・アーキテクチャのことです。通常この 2 種類のアプリケーションは、ネットワークに接続された異なるコンピュータ上で実行されます。

クライアント・メッセージ・ストア

QAnywhere では、メッセージを保管するリモート・デバイスにある SQL Anywhere データベースのことです。

クライアント・メッセージ・ストア ID

QAnywhere では、Mobile Link リモート ID のことです。これによって、クライアント・メッセージ・ストアがユニークに識別されます。

グローバル・テンポラリ・テーブル

明示的に削除されるまでデータ定義がすべてのユーザに表示されるテンポラリ・テーブルです。グローバル・テンポラリ・テーブルを使用すると、各ユーザが、1 つのテーブルのまったく同じインスタンスを開くことができます。デフォルトでは、コミット時にローが削除され、接続終了時にもローが削除されます。

参照：

- 「テンポラリ・テーブル」 292 ページ
- 「ローカル・テンポラリ・テーブル」 300 ページ

ゲートウェイ

Mobile Link システム・テーブルまたは Notifier プロパティ・ファイルに保存される Mobile Link オブジェクトで、システム起動同期用のメッセージの送信方法に関する情報が含まれます。

参照：「サーバ起動同期」 287 ページ。

コード・ページ

コード・ページは、文字セットの文字を数値表示 (通常 0 ～ 255 の整数) にマッピングするエンコードです。Windows Code Page 1252 などのコード・ページがあります。このマニュアルの目的上、コード・ページとエンコードは同じ意味で使用されます。

参照：

- 「文字セット」 307 ページ
- 「エンコード」 285 ページ
- 「照合」 304 ページ

コマンド・ファイル

SQL 文で構成されたテキスト・ファイルです。コマンド・ファイルは手動で作成できますが、データベース・ユーティリティによって自動的に作成することもできます。たとえば、dbunload ユーティリティを使うと、指定されたデータベースの再構築に必要な SQL 文で構成されたコマンド・ファイルを作成できます。

サーバ・メッセージ・ストア

QAnywhere では、サーバ上のリレーショナル・データベースです。このデータベースは、メッセージを、クライアント・メッセージ・ストアまたは JMS システムに転送されるまで一時的に格納します。メッセージは、サーバ・メッセージ・ストアを介して、クライアント間で交換されます。

サーバ管理要求

XML 形式の QAnywhere メッセージです。サーバ・メッセージ・ストアを管理したり、QAnywhere アプリケーションをモニタリングするために QAnywhere システム・キューに送信されます。

サーバ起動同期

Mobile Link サーバから Mobile Link 同期を開始する方法です。

サービス

Windows オペレーティング・システムで、アプリケーションを実行するユーザ ID がログオンしていないときにアプリケーションを実行する方法です。

サブクエリ

別の SELECT 文、INSERT 文、UPDATE 文、DELETE 文、または別のサブクエリの中にネストされた SELECT 文です。

相関とネストの 2 種類のサブクエリがあります。

サブスクリプション

Mobile Link 同期では、パブリケーションと Mobile Link ユーザ間のクライアント・データベース内のリンクであり、そのパブリケーションが記述したデータの同期を可能にします。

SQL Remote レプリケーションでは、パブリケーションとリモート・ユーザ間のリンクのことで、これによりリモート・ユーザはそのパブリケーションの更新内容を統合データベースとの間で交換できます。

参照：

- 「パブリケーション」 294 ページ
- 「Mobile Link ユーザ」 280 ページ

システム・オブジェクト

SYS または dbo が所有するデータベース・オブジェクトです。

システム・テーブル

SYS または dbo が所有するテーブルです。メタデータが格納されています。システム・テーブル (データ辞書テーブルとしても知られています) はデータベース・サーバが作成し管理します。

システム・ビュー

すべてのデータベースに含まれているビューです。システム・テーブル内に格納されている情報をわかりやすいフォーマットで示します。

ジョイン

指定されたカラムの値を比較することによって 2 つ以上のテーブルにあるローをリンクする、リレーショナル・システムでの基本的な操作です。

ジョイン・タイプ

SQL Anywhere では、クロス・ジョイン、キー・ジョイン、ナチュラル・ジョイン、ON 句を使ったジョインの 4 種類のジョインが使用されます。

参照：「ジョイン」 288 ページ。

ジョイン条件

ジョインの結果に影響を及ぼす制限です。ジョイン条件は、JOIN の直後に ON 句か WHERE 句を挿入して指定します。ナチュラル・ジョインとキー・ジョインについては、SQL Anywhere がジョイン条件を生成します。

参照：

- 「ジョイン」 288 ページ
- 「生成されたジョイン条件」 305 ページ

スキーマ

テーブル、カラム、インデックス、それらの関係などを含んだデータベース構造です。

スクリプト

Mobile Link では、Mobile Link のイベントを処理するために記述されたコードです。スクリプトは、業務上の要求に適合するように、データ交換をプログラム的に制御します。

参照：「イベント・モデル」 284 ページ。

スクリプト・バージョン

Mobile Link では、同期を作成するために同時に適用される、一連の同期スクリプトです。

スクリプトベースのアップロード

Mobile Link では、ログ・ファイルを使用した方法の代わりとなる、アップロード処理のカスタマイズ方法です。

ストアド・プロシージャ

ストアド・プロシージャは、データベースに保存され、データベース・サーバに対する一連の操作やクエリを実行するために使用される SQL 命令のグループです。

スナップショット・アイソレーション

読み込み要求を発行するトランザクション用のデータのコミットされたバージョンを返す、独立性レベルの種類です。SQL Anywhere では、スナップショット、文のスナップショット、読み込み専用文のスナップショットの 3 つのスナップショットの独立性レベルがあります。スナップショット・アイソレーションが使用されている場合、読み込み処理は書き込み処理をブロックしません。

参照：「独立性レベル」 307 ページ。

セキュア機能

データベース・サーバが起動されたときに、そのデータベース・サーバで実行されているデータベースでは使用できないように -sf オプションによって指定される機能です。

セッション・ベースの同期

統合データベースとリモート・データベースの両方でデータ表現の一貫性が保たれる同期です。Mobile Link はセッション・ベースです。

ダイレクト・ロー・ハンドリング

Mobile Link では、テーブル・データを Mobile Link でサポートされている統合データベース以外のソースに同期する方法のことです。アップロードとダウンロードの両方をダイレクト・ロー・ハンドリングで実装できます。

参照：

- 「統合データベース」 306 ページ
- 「SQL ベースの同期」 283 ページ

ダウンロード

同期中に、統合データベースからリモート・データベースにデータが転送される段階です。

チェックサム

データベース・ページを使用して記録されたデータベース・ページのビット数の合計です。チェックサムを使用すると、データベース管理システムは、ページがディスクに書き込まれるときに数が一致しているかを確認することで、ページの整合性を検証できます。数が一致した場合は、ページが正常に書き込まれたとみなされます。

チェックポイント

データベースに加えたすべての変更内容がデータベース・ファイルに保存されるポイントです。通常、コミットされた変更内容はトランザクション・ログだけに保存されます。

データ・キューブ

同じ結果を違う方法でグループ化およびソートされた内容を各次元に反映した、多次元の結果セットです。データ・キューブは、セルフジョイン・クエリと相関サブクエリを必要とするデータの複雑な情報を提供します。データ・キューブは OLAP 機能の一部です。

データベース

プライマリ・キーと外部キーによって関連付けられているテーブルの集合です。これらのテーブルでデータベース内の情報が保管されます。また、テーブルとキーによってデータベースの構造が定義されます。データベース管理システムでこの情報にアクセスします。

参照：

- 「外部キー」 301 ページ
- 「プライマリ・キー」 296 ページ
- 「データベース管理システム (DBMS)」 291 ページ
- 「リレーショナル・データベース管理システム (RDBMS)」 299 ページ

データベース・オブジェクト

情報を保管したり受け取ったりするデータベース・コンポーネントです。テーブル、インデックス、ビュー、プロシージャ、トリガはデータベース・オブジェクトです。

データベース・サーバ

データベース内にある情報へのすべてのアクセスを規制するコンピュータ・プログラムです。SQL Anywhere には、ネットワーク・サーバとパーソナル・サーバの 2 種類のサーバがあります。

データベース・ファイル

データベースは 1 つまたは複数のデータベース・ファイルに保持されます。まず、初期ファイルがあり、それに続くファイルは DB 領域と呼ばれます。各テーブルは、それに関連付けられているインデックスとともに、単一のデータベース・ファイルに含まれている必要があります。

参照：「DB 領域」 277 ページ。

データベース管理システム (DBMS)

データベースを作成したり使用したりするためのプログラムの集合です。

参照：「リレーショナル・データベース管理システム (RDBMS)」 299 ページ。

データベース管理者 (DBA)

データベースの管理に必要なパーミッションを持つユーザです。DBA は、データベース・スキーマのあらゆる変更や、ユーザやグループの管理に対して、全般的な責任を負います。データベース管理者のロールはデータベース内に自動的に作成されます。その場合、ユーザ ID は DBA であり、パスワードは sql です。

データベース所有者 (dbo)

SYS が所有しないシステム・オブジェクトを所有する特別なユーザです。

参照：

- 「データベース管理者 (DBA)」 291 ページ
- 「SYS」 283 ページ

データベース接続

クライアント・アプリケーションとデータベース間の通信チャネルです。接続を確立するためには有効なユーザ ID とパスワードが必要です。接続中に実行できるアクションは、そのユーザ ID に付与された権限によって決まります。

データベース名

サーバがデータベースをロードするとき、そのデータベースに指定する名前です。デフォルトのデータベース名は、初期データベース・ファイルのルート名です。

参照：「データベース・ファイル」 291 ページ。

データ型

CHAR や NUMERIC などのデータのフォーマットです。ANSI SQL 規格では、サイズ、文字セット、照合に関する制限もデータ型に組み込めます。

参照：「ドメイン」 292 ページ。

データ操作言語 (DML)

データベース内のデータの操作に使う SQL 文のサブセットです。DML 文は、データベース内のデータを検索、挿入、更新、削除します。

データ定義言語 (DDL)

データベース内のデータの構造を定義するときに使う SQL 文のサブセットです。DDL 文は、テーブルやユーザなどのデータベース・オブジェクトを作成、変更、削除できます。

デッドロック

先へ進めない場所に一連のトランザクションが到達する状態です。

デバイス・トラッキング

Mobile Link サーバ起動同期において、デバイスを特定する Mobile Link のユーザ名を使用して、メッセージのアドレスを指定できる機能です。

参照：「サーバ起動同期」 287 ページ。

テンポラリ・テーブル

データを一時的に保管するために作成されるテーブルです。グローバルとローカルの 2 種類があります。

参照：

- 「ローカル・テンポラリ・テーブル」 300 ページ
- 「グローバル・テンポラリ・テーブル」 286 ページ

ドメイン

適切な位置に精度や小数点以下の桁数を含み、さらにオプションとしてデフォルト値や CHECK 条件などを含んでいる、組み込みデータ型のエイリアスです。ドメインには、通貨データ型のように SQL Anywhere が事前に定義したものもあります。ユーザ定義データ型とも呼ばれます。

参照：「データ型」 292 ページ。

トランザクション

作業の論理単位を構成する一連の SQL 文です。1 つのトランザクションは完全に処理されるかまったく処理されないかのどちらかです。SQL Anywhere は、ロック機能のあるトランザクション処理をサポートしているので、複数のトランザクションが同時にデータベースにアクセスしてもデータを壊すことがありません。トランザクションは、データに加えた変更を永久なものにする COMMIT 文か、トランザクション中に加えられたすべての変更を元に戻す ROLLBACK 文のいずれかで終了します。

トランザクション・ログ

データベースに対するすべての変更内容が、変更された順に格納されるファイルです。パフォーマンスを向上させ、データベース・ファイルが破損した場合でもデータをリカバリできます。

トランザクション・ログ・ミラー

オプションで設定できる、トランザクション・ログ・ファイルの完全なコピーのことで、トランザクション・ログと同時に管理されます。データベースの変更がトランザクション・ログへ書き込まれると、トランザクション・ログ・ミラーにも同じ内容が書き込まれます。

ミラー・ファイルは、トランザクション・ログとは別のデバイスに置いてください。一方のデバイスに障害が発生しても、もう一方のログにリカバリのためのデータが確保されます。

参照：「トランザクション・ログ」 293 ページ。

トランザクション単位の整合性

Mobile Link で、同期システム全体でのトランザクションの管理を保証します。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。

トリガ

データを修正するクエリをユーザが実行すると、自動的に実行されるストアド・プロシージャの特別な形式です。

参照：

- 「ロー・レベルのトリガ」 300 ページ
- 「文レベルのトリガ」 307 ページ
- 「整合性」 304 ページ

ネットワーク・サーバ

共通ネットワークを共有するコンピュータからの接続を受け入れるデータベース・サーバです。

参照：「パーソナル・サーバ」 294 ページ。

ネットワーク・プロトコル

TCP/IP や HTTP などの通信の種類です。

パーソナル・サーバ

クライアント・アプリケーションが実行されているコンピュータと同じマシンで実行されているデータベース・サーバです。パーソナル・データベース・サーバは、単一のコンピュータ上で単一のユーザが使用しますが、そのユーザからの複数の同時接続をサポートできます。

パッケージ

Java では、それぞれが互いに関連のあるクラスの集合を指します。

ハッシュ

ハッシュは、インデックスのエントリをキーに変換する、インデックスの最適化のことです。インデックスのハッシュの目的は、必要なだけの実際のロー・データをロー ID に含めることで、インデックスされた値を特定するためのローの検索、ロード、アンパックという負荷の高い処理を避けることです。

パフォーマンス統計値

データベース・システムのパフォーマンスを反映する値です。たとえば、CURRREAD 統計値は、データベース・サーバが要求したファイル読み込みのうち、現在まだ完了していないものの数を表します。

パブリケーション

Mobile Link または SQL Remote では、同期されるデータを識別するデータベース・オブジェクトのことです。Mobile Link では、クライアント上にのみ存在します。1 つのパブリケーションは複数のアーティクルから構成されています。SQL Remote ユーザは、パブリケーションに対してサブスクリプションを作成することによって、パブリケーションを受信できます。Mobile Link ユーザは、パブリケーションに対して同期サブスクリプションを作成することによって、パブリケーションを同期できます。

参照：

- 「レプリケーション」 299 ページ
- 「アーティクル」 284 ページ
- 「パブリケーションの更新」 294 ページ

パブリケーションの更新

SQL Remote レプリケーションでは、単一のデータベース内の 1 つまたは複数のパブリケーションに対して加えられた変更のリストを指します。パブリケーションの更新は、レプリケーション・メッセージの一部として定期的にリモート・データベースへ送られます。

参照：

- 「レプリケーション」 299 ページ
- 「パブリケーション」 294 ページ

パブリッシャ

SQL Remote レプリケーションでは、レプリケートできる他のデータベースとレプリケーション・メッセージを交換できるデータベースの単一ユーザを指します。

参照：「レプリケーション」 299 ページ。

ビジネス・ルール

実世界の要求に基づくガイドラインです。通常ビジネス・ルールは、検査制約、ユーザ定義データ型、適切なトランザクションの使用により実装されます。

参照：

- 「制約」 304 ページ
- 「ユーザ定義データ型」 298 ページ

ヒストグラム

ヒストグラムは、カラム統計のもっとも重要なコンポーネントであり、データ分散を表します。SQL Anywhere は、ヒストグラムを維持して、カラムの値の分散に関する統計情報をオプティマイザに提供します。

ビット配列

ビット配列は、一連のビットを効率的に保管するのに使用される配列データ構造の種類です。ビット配列は文字列に似てますが、使用される要素は文字ではなく 0 (ゼロ) と 1 になります。ビット配列は、一般的にブール値の文字列を保持するのに使用されます。

ビュー

データベースにオブジェクトとして格納される SELECT 文です。ビューを使用すると、ユーザは 1 つまたは複数のテーブルのローやカラムのサブセットを参照できます。ユーザが特定のテーブルやテーブルの組み合わせのビューを使うたびに、テーブルに保持されているデータから再計算されます。ビューは、セキュリティの目的に有用です。またデータベース情報の表示を調整して、データへのアクセスが簡単になるようにする場合も役立ちます。

ファイルベースのダウンロード

Mobile Link では、ダウンロードがファイルとして配布されるデータの同期方法であり、同期変更のオフライン配布を可能にします。

ファイル定義データベース

Mobile Link では、ダウンロード・ファイルの作成に使用される SQL Anywhere データベースのことです。

参照：「ファイルベースのダウンロード」 295 ページ。

フェールオーバ

アクティブなサーバ、システム、またはネットワークで障害や予定外の停止が発生したときに、冗長な (スタンバイ) サーバ、システム、またはネットワークに切り替えることです。フェールオーバは自動的に発生します。

プライマリ・キー

テーブル内のすべてのローをユニークに識別する値を持つカラムまたはカラムのリストです。

参照：「外部キー」 301 ページ。

プライマリ・キー制約

プライマリ・キーのカラムに対する一意性制約です。テーブルにはプライマリ・キー制約を 1 つしか設定できません。

参照：

- 「制約」 304 ページ
- 「検査制約」 303 ページ
- 「外部キー制約」 302 ページ
- 「一意性制約」 301 ページ
- 「整合性」 304 ページ

プライマリ・テーブル

外部キー関係でプライマリ・キーを含むテーブルです。

プラグイン・モジュール

Sybase Central で、製品にアクセスしたり管理したりする方法です。プラグインは、通常、インストールすると Sybase Central にもインストールされ、自動的に登録されます。プラグインは、多くの場合、Sybase Central のメイン・ウィンドウに最上位のコンテナとして、その製品名 (たとえば SQL Anywhere) で表示されます。

参照：「Sybase Central」 283 ページ。

フル・バックアップ

データベース全体をバックアップすることです。オプションでトランザクション・ログのバックアップも可能です。フル・バックアップには、データベース内のすべての情報が含まれており、システム障害やメディア障害が発生した場合の保護として機能します。

参照：「インクリメンタル・バックアップ」 284 ページ。

プロキシ・テーブル

メタデータを含むローカル・テーブルです。リモート・データベース・サーバのテーブルに、ローカル・テーブルであるかのようにアクセスするときに使用します。

参照：「メタデータ」 297 ページ。

ベース・テーブル

データを格納する永久テーブルです。テーブルは、テンポラリ・テーブルやビューと区別するために、「ベース・テーブル」と呼ばれることがあります。

参照：

- 「テンポラリ・テーブル」 292 ページ
- 「ビュー」 295 ページ

ポーリング

Mobile Link サーバ起動同期において、Mobile Link Listerner などのライト・ウェイト・ポーラが Notifier から Push 通知を要求する方法です。

参照：「サーバ起動同期」 287 ページ。

ポリシー

QAnywhere では、メッセージ転送の発生時期を指定する方法のことです。

マテリアライズド・ビュー

計算され、ディスクに保存されたビューのことです。マテリアライズド・ビューは、ビュー (クエリ指定を使用して定義される) とテーブル (ほとんどのテーブルの操作をそのテーブル上で実行できる) の両方の特性を持ちます。

参照：

- 「ベース・テーブル」 297 ページ
- 「ビュー」 295 ページ

ミラー・ログ

参照：「トランザクション・ログ・ミラー」 293 ページ。

メタデータ

データについて説明したデータです。メタデータは、他のデータの特質と内容について記述しています。

参照：「スキーマ」 289 ページ。

メッセージ・システム

SQL Remote のレプリケーションでは、統合データベースとリモート・データベースの間でのメッセージのやりとりに使用するプロトコルのことです。SQL Anywhere では、FILE、FTP、SMTP のメッセージ・システムがサポートされています。

参照：

- 「レプリケーション」 299 ページ
- 「FILE」 278 ページ

メッセージ・ストア

QAnywhere では、メッセージを格納するクライアントおよびサーバ・デバイスのデータベースのことです。

参照：

- 「クライアント・メッセージ・ストア」 286 ページ
- 「サーバ・メッセージ・ストア」 287 ページ

メッセージ・タイプ

SQL Remote のレプリケーションでは、リモート・ユーザと統合データベースのパブリッシャとの通信方法を指定するデータベース・オブジェクトのことを指します。統合データベースには、複数のメッセージ・タイプが定義されていることがあります。これによって、リモート・ユーザはさまざまなメッセージ・システムを使って統合データベースと通信できることになります。

参照：

- 「レプリケーション」 299 ページ
- 「統合データベース」 306 ページ

メッセージ・ログ

データベース・サーバや Mobile Link サーバなどのアプリケーションからのメッセージを格納できるログです。この情報は、メッセージ・ウィンドウに表示されたり、ファイルに記録されたりすることもあります。メッセージ・ログには、情報メッセージ、エラー、警告、MESSAGE 文からのメッセージが含まれます。

メンテナンス・リリース

メンテナンス・リリースは、同じメジャー・バージョン番号を持つ旧バージョンのインストール済みソフトウェアをアップグレードするための完全なソフトウェア・セットです (バージョン番号のフォーマットは、*メジャー.マイナー.パッチ.ビルド*です)。バグ・フィックスとその他の変更については、アップグレードのリリース・ノートにリストされます。

ユーザ定義データ型

参照：「ドメイン」 292 ページ。

ライト・ウェイト・ポーラ

Mobile Link サーバ起動同期において、Mobile Link サーバからの Push 通知をポーリングするデバイス・アプリケーションです。

参照：「サーバ起動同期」 287 ページ。

リダイレクタ

クライアントと Mobile Link サーバ間で要求と応答をルート指定する Web サーバ・プラグインです。このプラグインによって、負荷分散メカニズムとフェールオーバ・メカニズムも実装されます。

リファレンス・データベース

Mobile Link では、Ultra Light クライアントの開発に使用される SQL Anywhere データベースです。開発中は、1 つの SQL Anywhere データベースをリファレンス・データベースとしても統合データベースとしても使用できます。他の製品によって作成されたデータベースは、リファレンス・データベースとして使用できません。

リモート ID

SQL Anywhere と Ultra Light データベース内のユニークな識別子で、Mobile Link によって使用されます。リモート ID は NULL に初期設定されていますが、データベースの最初の同期時に GUID に設定されます。

リモート・データベース

Mobile Link または SQL Remote では、統合データベースとデータを交換するデータベースを指します。リモート・データベースは、統合データベース内のすべてまたは一部のデータを共有できます。

参照：

- 「同期」 306 ページ
- 「統合データベース」 306 ページ

リレーショナル・データベース管理システム (RDBMS)

関連するテーブルの形式でデータを格納するデータベース管理システムです。

参照：「データベース管理システム (DBMS)」 291 ページ。

レプリケーション

物理的に異なるデータベース間でデータを共有することです。Sybase では、Mobile Link、SQL Remote、Replication Server の 3 種類のレプリケーション・テクノロジを提供しています。

レプリケーション・メッセージ

SQL Remote または Replication Server では、パブリッシュするデータベースとサブスクリプションを作成するデータベース間で送信される通信内容を指します。メッセージにはデータを含み、レプリケーション・システムで必要なパススルー文、情報があります。

参照：

- 「レプリケーション」 299 ページ
- 「パブリケーションの更新」 294 ページ

レプリケーションの頻度

SQL Remote レプリケーションでは、リモート・ユーザに対する設定の 1 つで、パブリッシャの Message Agent がレプリケーション・メッセージを他のリモート・ユーザに送信する頻度を定義します。

参照：「レプリケーション」 299 ページ。

ロー・レベルのトリガ

変更されているローごとに一回実行するトリガです。

参照：

- 「トリガ」 293 ページ
- 「文レベルのトリガ」 307 ページ

ローカル・テンポラリ・テーブル

複合文を実行する間だけ存在したり、接続が終了するまで存在したりするテンポラリ・テーブルです。データのセットを 1 回だけロードする必要がある場合にローカル・テンポラリ・テーブルが便利です。デフォルトでは、COMMIT を実行するとローが削除されます。

参照：

- 「テンポラリ・テーブル」 292 ページ
- 「グローバル・テンポラリ・テーブル」 286 ページ

ロール

概念データベース・モデルで、ある視点からの関係を説明する動詞またはフレーズを指します。各関係は 2 つのロールを使用して表すことができます。"contains (A は B を含む)" や "is a member of (B は A のメンバ)" などのロールがあります。

ロールバック・ログ

コミットされていない各トランザクションの最中に行われた変更のレコードです。ROLLBACK 要求やシステム障害が発生した場合、コミットされていないトランザクションはデータベースから破棄され、データベースは前の状態に戻ります。各トランザクションにはそれぞれロールバック・ログが作成されます。このログは、トランザクションが完了すると削除されます。

参照：「トランザクション」 293 ページ。

ロール名

外部キーの名前です。この外部キーがロール名と呼ばれるのは、外部テーブルとプライマリ・テーブル間の関係に名前を指定するためです。デフォルトでは、テーブル名がロール名になります。ただし、別の外部キーがそのテーブル名を使用している場合、デフォルトのロール名はテーブル名に 3 桁のユニークな数字を付けたものになります。ロール名は独自に作成することもできます。

参照：「外部キー」 301 ページ。

ログ・ファイル

SQL Anywhere によって管理されているトランザクションのログです。ログ・ファイルを使用すると、システム障害やメディア障害が発生してもデータベースを回復させることができます。また、データベースのパフォーマンスを向上させたり、SQL Remote を使用してデータをレプリケートしたりする場合にも使用できます。

参照：

- 「トランザクション・ログ」 293 ページ
- 「トランザクション・ログ・ミラー」 293 ページ
- 「フル・バックアップ」 296 ページ

ロック

複数のトランザクションを同時に実行しているときにデータの整合性を保護する同時制御メカニズムです。SQL Anywhere では、2 つの接続によって同じデータが同時に変更されないようにするために、また変更処理の最中に他の接続によってデータが読み込まれないようにするために、自動的にロックが適用されます。

ロックの制御は、独立性レベルを設定して行います。

参照：

- 「独立性レベル」 307 ページ
- 「同時性 (同時実行性)」 307 ページ
- 「整合性」 304 ページ

ワーク・テーブル

クエリの最適化の最中に中間結果を保管する内部保管領域です。

一意性制約

NULL 以外のすべての値が重複しないことを要求するカラムまたはカラムのセットに対する制限です。テーブルには複数の一意性制約を指定できます。

参照：

- 「外部キー制約」 302 ページ
- 「プライマリ・キー制約」 296 ページ
- 「制約」 304 ページ

解析ツリー

クエリを代数で表現したものです。

外部キー

別のテーブルにあるプライマリ・キーの値を複製する、テーブルの 1 つ以上のカラムです。テーブル間の関係は、外部キーによって確立されます。

参照：

- 「プライマリ・キー」 296 ページ
- 「外部テーブル」 302 ページ

外部キー制約

カラムまたはカラムのセットに対する制約で、テーブルのデータが別のテーブルのデータとどのように関係しているかを指定するものです。カラムのセットに外部キー制約を加えると、それらのカラムが外部キーになります。

参照：

- 「制約」 304 ページ
- 「検査制約」 303 ページ
- 「プライマリ・キー制約」 296 ページ
- 「一意性制約」 301 ページ

外部ジョイン

テーブル内のすべてのローを保護するジョインです。SQL Anywhere では、左外部ジョイン、右外部ジョイン、全外部ジョインがサポートされています。左外部ジョインは JOIN 演算子の左側にあるテーブルのローを保護し、右側にあるテーブルのローがジョイン条件を満たさない場合には NULL を返します。全外部ジョインは両方のテーブルに含まれるすべてのローを保護します。

参照：

- 「ジョイン」 288 ページ
- 「内部ジョイン」 307 ページ

外部テーブル

外部キーを持つテーブルです。

参照：「外部キー」 301 ページ。

外部ログイン

リモート・サーバとの通信に使用される代替のログイン名とパスワードです。デフォルトでは、SQL Anywhere は、クライアントに代わってリモート・サーバに接続するときは、常にそのクライアントの名前とパスワードを使用します。外部ログインを作成することによって、このデフォルトを上書きできます。外部ログインは、リモート・サーバと通信するときに使用する代替のログイン名とパスワードです。

競合

リソースについて対立する動作のことです。たとえば、データベース用語では、複数のユーザがデータベースの同じローを編集しようとした場合、そのローの編集権についての競合が発生します。

競合解決

Mobile Link では、競合解決は 2 人のユーザが別々のリモート・データベースの同じローを変更した場合にどう処理するかを指定するロジックのことです。

検査制約

指定された条件をカラムやカラムのセットに課す制約です。

参照：

- 「制約」 304 ページ
- 「外部キー制約」 302 ページ
- 「プライマリ・キー制約」 296 ページ
- 「一意性制約」 301 ページ

検証

データベース、テーブル、またはインデックスについて、特定のタイプのファイル破損をテストすることです。

作成者 ID

Ultra Light の Palm OS アプリケーションでは、アプリケーションが作成されたときに割り当てられる ID のことです。

参照元オブジェクト

テーブルなどのデータベースの別のオブジェクトをオブジェクト定義が直接参照する、ビューなどのオブジェクトです。

参照：「外部キー」 301 ページ。

参照整合性

データの整合性、特に異なるテーブルのプライマリ・キー値と外部キー値との関係を管理する規則を厳守することです。参照整合性を備えるには、それぞれの外部キーの値が、参照テーブルにあるローのプライマリ・キー値に対応するようにします。

参照：

- 「プライマリ・キー」 296 ページ
- 「外部キー」 301 ページ

参照先オブジェクト

ビューなどの別のオブジェクトの定義で直接参照される、テーブルなどのオブジェクトです。

参照：「プライマリ・キー」 296 ページ。

識別子

テーブルやカラムなどのデータベース・オブジェクトを参照するときに使う文字列です。A ～ Z、a ～ z、0 ～ 9、アンダースコア (_)、アットマーク (@)、シャープ記号 (#)、ドル記号 ($) のうち、任意の文字を識別子として使用できます。

述部

条件式です。オプションで論理演算子 AND や OR と組み合わせて、WHERE 句または HAVING 句に条件のセットを作成します。SQL では、unknown と評価される述部が false と解釈されます。

照合

データベース内のテキストのプロパティを定義する文字セットとソート順の組み合わせのことです。SQL Anywhere データベースでは、サーバを実行しているオペレーティング・システムと言語によって、デフォルトの照合が決まります。たとえば、英語版 Windows システムのデフォルトの照合は 1252LATIN1 です。照合は、照合順とも呼ばれ、文字列の比較とソートに使用します。

参照：

- 「文字セット」 307 ページ
- 「コード・ページ」 287 ページ
- 「エンコード」 285 ページ

世代番号

Mobile Link では、リモート・データベースがデータをアップロードしてからダウンロード・ファイルを適用するようにするためのメカニズムのことです。

参照：「ファイルベースのダウンロード」 295 ページ。

制約

テーブルやカラムなど、特定のデータベース・オブジェクトに含まれた値に関する制約です。たとえば、一意性制約があるカラム内の値は、すべて異なっている必要があります。テーブルに、そのテーブルの情報と他のテーブルのデータがどのように関係しているのかを指定する外部キー制約が設定されていることもあります。

参照：

- 「検査制約」 303 ページ
- 「外部キー制約」 302 ページ
- 「プライマリ・キー制約」 296 ページ
- 「一意性制約」 301 ページ

整合性

データが適切かつ正確であり、データベースの関係構造が保たれていることを保証する規則を厳守することです。

参照：「参照整合性」 303 ページ。

正規化

データベース・スキーマを改善することです。リレーショナル・データベース理論に基づく規則に従って、冗長性を排除したり、編成を改良します。

正規表現

正規表現は、文字列内で検索するパターンを定義する、一連の文字、ワイルドカード、演算子です。

生成されたジョイン条件

自動的に生成される、ジョインの結果に対する制限です。キーとナチュラルの 2 種類があります。キー・ジョインは、KEY JOIN を指定したとき、またはキーワード JOIN を指定したが、CROSS、NATURAL、または ON を使用しなかった場合に生成されます。キー・ジョインの場合、生成されたジョイン条件はテーブル間の外部キー関係に基づいています。ナチュラル・ジョインは NATURAL JOIN を指定したときに生成され、生成されたジョイン条件は、2 つのテーブルの共通のカラム名に基づきます。

参照：

- 「ジョイン」 288 ページ
- 「ジョイン条件」 289 ページ

接続 ID

クライアント・アプリケーションとデータベース間の特定の接続に付けられるユニークな識別番号です。現在の接続 ID を確認するには、次の SQL 文を使用します。

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

接続プロファイル

ユーザ名、パスワード、サーバ名などの、データベースに接続するために必要なパラメータのセットです。便宜的に保管され使用されます。

接続起動同期

Mobile Link のサーバ起動同期の 1 つの形式で、接続が変更されたときに同期が開始されます。

参照：「サーバ起動同期」 287 ページ。

相関名

クエリの FROM 句内で使用されるテーブルやビューの名前です。テーブルやビューの元の名前か、FROM 句で定義した代替名のいずれかになります。

抽出

SQL Remote レプリケーションでは、統合データベースから適切な構造とデータをアンロードする動作を指します。この情報は、リモート・データベースを初期化するときに使用されます。

参照：「レプリケーション」 299 ページ。

通信ストリーム

Mobile Link では、Mobile Link クライアントと Mobile Link サーバ間での通信にネットワーク・プロトコルが使用されます。

転送ルール

QAnywhere では、メッセージの転送を発生させる時期、転送するメッセージ、メッセージを削除する時期を決定する論理のことです。

統合データベース

分散データベース環境で、データのマスタ・コピーを格納するデータベースです。競合や不一致が発生した場合、データのプライマリ・コピーは統合データベースにあるとみなされます。

参照：

- 「同期」 306 ページ
- 「レプリケーション」 299 ページ

統合化ログイン

オペレーティング・システムへのログイン、ネットワークへのログイン、データベースへの接続に、同一のユーザ ID とパスワードを使用するログイン機能の 1 つです。

動的 SQL

実行される前に作成したプログラムによって生成される SQL です。Ultra Light の動的 SQL は、占有容量の小さいデバイス用に設計された変形型です。

同期

Mobile Link テクノロジを使用してデータベース間でデータをレプリケートする処理です。

SQL Remote では、同期はデータの初期セットを使ってリモート・データベースを初期化する処理を表すために特に使用されます。

参照：

- 「Mobile Link」 279 ページ
- 「SQL Remote」 282 ページ

同時性 (同時実行性)

互いに独立し、場合によっては競合する可能性のある 2 つ以上の処理を同時に実行することです。SQL Anywhere では、自動的にロックを使用して各トランザクションを独立させ、同時に稼働するそれぞれのアプリケーションが一貫したデータのセットを参照できるようにします。

参照：

- 「トランザクション」 293 ページ
- 「独立性レベル」 307 ページ

独立性レベル

あるトランザクションの操作が、同時に処理されている別のトランザクションの操作からどの程度参照できるかを示します。独立性レベルには 0 から 3 までの 4 つのレベルがあります。最も高い独立性レベルには 3 が設定されます。デフォルトでは、レベルは 0 に設定されています。SQL Anywhere では、スナップショット、文のスナップショット、読み込み専用文のスナップショットの 3 つのスナップショットの独立性レベルがあります。

参照：「スナップショット・アイソレーション」 289 ページ。

内部ジョイン

2 つのテーブルがジョイン条件を満たす場合だけ、結果セットにローが表示されるジョインです。内部ジョインがデフォルトです。

参照：

- 「ジョイン」 288 ページ
- 「外部ジョイン」 302 ページ

物理インデックス

インデックスがディスクに保存されるときの実際のインデックス構造です。

文レベルのトリガ

トリガ付きの文の処理が完了した後に実行されるトリガです。

参照：

- 「トリガ」 293 ページ
- 「ロー・レベルのトリガ」 300 ページ

文字セット

文字セットは記号、文字、数字、スペースなどから成ります。"ISO-8859-1" は文字セットの例です。Latin1 とも呼ばれます。

参照：

- 「コード・ページ」 287 ページ
- 「エンコード」 285 ページ
- 「照合」 304 ページ

文字列リテラル

文字列リテラルとは、一重引用符 (') で囲まれ、シーケンスで並べられた文字のことです。

論理インデックス

物理インデックスへの参照 (ポインタ) です。ディスクに保存される論理インデックス用のインデックス構造はありません。

索引

記号

A

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

ふ

