SYBASE iAnywhere

Mobile Link
サーバ管理

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、Mobile Link サーバ、統合データベース、および Mobile Link アプリケーションを設定および管理する方法について説明します。また、Mobile Link サーバの正常性や可用性に関する情報を示す Web ブラウザベースの管理ツールである Mobile Link 用の SQL Anywhere モニタ、および Web サーバを通じて通信する Mobile Link、Afaria、iAnywhere Mobile Office サーバとモバイル・デバイス間で安全な通信を実現するリレー・サーバについても説明します。

対象読者

このマニュアルは、分散情報システムを作成したいと考えているユーザを対象としています。中央のデータ・ソースとリモート・データ・ストアにはリレーショナル・データベース・システムを使用できますが、それに限定されるわけではありません。

始める前に

Mobile Link と他の SQL Anywhere 同期／レプリケーション・テクノロジの比較については、「同期テクノロジの比較」 『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。

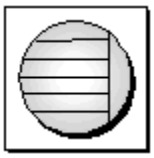
- **環境変数**　マニュアルでは、環境変数設定が引用されます。Windows のシェルでは、環境変数は構文 *%ENVVAR%* を使用して指定されます。UNIX のシェルでは、環境変数は構文 *$ENVVAR* または *${ENVVAR}* を使用して指定されます。

グラフィック・アイコン

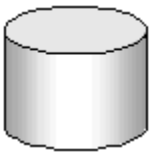
このマニュアルでは、次のアイコンを使用します。

- 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 サーバ・テクノロジの使用

この項では、Mobile Link テクノロジの概要と、Mobile Link テクノロジを使用して 2 つ以上のデータ・ソース間でデータを同期する方法について説明します。

Mobile Link 統合データベース

目次

統合データベースの概要

統合データベースには、Mobile Link で必要なシステム・オブジェクトが格納されます。ほとんどの場合、統合データベースにはアプリケーション・データも格納されますが、アプリケーション・データのすべてまたは一部は、他の方法でも格納できます。

Mobile Link は、32 ビットおよび 64 ビットの環境にある Windows および Linux 用の統合データベースをサポートしています。統合データベースとして、ODBC に準拠した以下のいずれかの RDBMS を使用できます。

- Adaptive Server Enterprise (64 ビットの Linux のサポートは提供されていません)
- IBM DB2 LUW
- IBM DB2 メインフレーム
- Microsoft SQL Server
- MySQL
- Oracle
- SQL Anywhere

サポートされているバージョンについては、http://www.ianywhere.jp/sas/os.html を参照してください。

SQL Anywhere のインストール環境には、各タイプの RDBMS の設定スクリプトが含まれています。その RDBMS を Mobile Link で使用するには、該当する設定スクリプトを実行する必要があります。設定スクリプトは、Mobile Link で必要なテーブルとストアド・プロシージャを追加します。

各タイプのデータベースを統合データベースとして設定する方法については、「統合データベースの設定」 6 ページを参照してください。

特定の統合データベース用の同期スクリプトを記述する方法については、「RDBMS 依存の同期スクリプト」 8 ページを参照してください。

他のデータ・ソースとの同期

Mobile Link 環境には、統合データベースとして設定されたデータベースが必要です。しかし、統合データベース以外のデータ・ソースとも同期できます。テキスト・ファイル、Web サービス、非リレーショナル・データベース、スプレッドシートなど、その他のどのようなデータ・ソースもほとんどは使用できます。次のことが可能です。

- 統合データベースとその他のデータ・ソースの両方に同期するハイブリッド・アプリケーションを作成します。
- 統合データベースのみに同期します。
- 別のデータ・ソースのみに同期します。

「ダイレクト・ロー・ハンドリング」 679 ページを参照してください。

統合データベースの変更に関する制限

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があります。このような場合に備えて、Mobile Link には、可能な場合に統合データベースへの変更を最小限に抑えるためのソリュー

ションがあります。たとえば、Mobile Link には、ユニークなプライマリ・キーを維持するためのさまざまなソリューションがあり、その中のいくつかは、統合データベースのスキーマへの影響を最小限に抑えます。

また、Mobile Link システム・オブジェクトを別のデータベースに置くことによって、統合データベースへのほとんどすべての影響を避けることができます。「Mobile Link システム・データベース」 7 ページを参照してください。

リモート・テーブルと統合データベースの関係

同期の設計では、リモート・データベースにあるテーブル／カラムと統合データベースにあるテーブル／カラムの間のマッピングが指定されます。通常、リモート・データベースのテーブルとカラムは、統合データベースのテーブルとカラムと完全に一致するか、それらのサブセットです。

許可されている任意の関係

リモート・データベースにあるテーブルは、統合データベースのテーブルと同じである必要はありません。1 つのリモート・アプリケーション・テーブルで同期されたデータは、異なるテーブルのカラムに配布できます。また、異なる統合データベースのテーブル間でも配布できます。これらの関係を指定するには、同期スクリプトを使用します。

シンプルな直接関係

最も簡単で一般的な設計では、統合データベースのテーブル構造のサブセットであるリモート・データベースのテーブル構造を使用します。この設計を使うと、リモート・データベースの各テーブルが統合データベースに存在するようになります。対応するテーブルの構造体と外部キーの関係が、統合データベースのものと同じになります。

統合データベースには、同期されていないカラムとテーブルが含まれていることがよくあります。これらのカラムやテーブルの一部は、同期に使用されている可能性があります。たとえば、timestamp カラムは、統合データベース内の新しいローや更新されたローを識別できます。また、シャドー・テーブルは、削除を追跡するのに使用できます。統合データベース内の同期されていないカラムやテーブルには、リモート・サイトでは必要ない情報を保存できます。

リモート・データベースにも、同期されていないテーブルやカラムが含まれていることがよくあります。

参照

- 「リモート・データベースと統合データベース間での Mobile Link データ・マッピング」 781 ページ

統合データベースの設定

設定スクリプト

Mobile Link 統合データベースとして使用できるようにデータベースを設定するには、設定スクリプトを実行します。SQL Anywhere のインストール環境には、サポートされている各 RDBMS のスクリプトが含まれています。これらのスクリプトは、すべて *install-dir¥MobiLink¥setup* にあります。Mobile Link システム設定を更新するには、次の方法を使用することもできます。

- Sybase Central の Mobile Link プラグインで、**[モード] - [管理]** を選択し、サーバ・データベースに接続します。データベース名を右クリックし、**[Mobile Link システム設定のチェック]** を選択します。データベースの設定が必要な場合は、続行のプロンプトが表示されます。
- **同期モデル作成ウィザード**または**同期モデル展開ウィザード**を使用する場合は、サーバ・データベースに接続するとシステム設定がチェックされます。データベースの設定が必要な場合は、続行のプロンプトが表示されます。「Mobile Link のモデルの概要」 『Mobile Link - クイック・スタート』を参照してください。

Mobile Link 設定スクリプトは、データベースに Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、ビューを追加します。これらのテーブルとプロシージャは Mobile Link 同期に必要です。

インストールされる Mobile Link システム・テーブルの詳細については、「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

インストールされるストアド・プロシージャの詳細については、「Mobile Link システム・プロシージャ」 698 ページを参照してください。

各設定スクリプトの機能を確認したい場合には、テキスト・エディタで表示できます。

警告
設定スクリプトを実行するデータベース・ユーザには、Mobile Link システム・テーブルを更新するパーミッションが与えられます。このパーミッションは、Mobile Link サーバを起動したり Mobile Link を設定したりする場合に必要です。「必要なパーミッション」 31 ページを参照してください。

設定スクリプトの実行方法については、使用する RDBMS についての以下の項を参照してください。

- 「Adaptive Server Enterprise 統合データベース」 10 ページ
- 「IBM DB2 LUW 統合データベース」 12 ページ
- 「IBM DB2 メインフレーム統合データベース」 15 ページ
- 「Microsoft SQL Server 統合データベース」 20 ページ
- 「MySQL 統合データベース」 22 ページ
- 「Oracle 統合データベース」 25 ページ
- 「SQL Anywhere 統合データベース」 28 ページ

注意
設定している統合データベースを QAnywhere Server Store として使用する場合は、比較や文字列演算に対してデータベースを大文字と小文字を区別しないように設定する必要があります。

ODBC 接続

Mobile Link サーバでは、統合データベースへの ODBC 接続が必要です。使用しているサーバ用の適切な ODBC ドライバを設定して、Mobile Link サーバを実行しているコンピュータのデータベース用に ODBC データ・ソースを作成してください。

Mobile Link ODBC ドライバの詳細については、「Mobile Link 対応の iAnywhere Solutions ODBC ドライバ」 841 ページを参照してください。

Mobile Link で使用できる ODBC ドライバの最新の情報と完全な機能仕様については、「Mobile Link の推奨 ODBC ドライバ」を参照してください。

Mobile Link システム・データベース

統合データベースをデータ用のデータベースと Mobile Link システム情報用のデータベースの 2 つに分割する必要がある場合もあります。このとき、統合データベースに Mobile Link システム・オブジェクトを追加する必要はありません。Mobile Link システム・データベースという別のデータベースにすべての Mobile Link システム・オブジェクトを格納できます。

Mobile Link システム・データベースは、統合データベースとしてサポートされる任意のデータベースにすることができます。統合データベースと同じ RDBMS にする必要はありません。

Mobile Link システム・データベースは簡単に設定できます。統合データベース以外のデータベースに Mobile Link 設定スクリプトを適用します。Mobile Link サーバを起動するときに、両方のデータベースに接続します。

注意

- Windows 上でのみ、Mobile Link サーバを実行できます。
- Mobile Link システム・データベースは、Mobile Link の**同期モデル作成ウィザード**およびモデル・モードで使用することはできません。
- 別のデータベースに Mobile Link システム・オブジェクトを格納するための、パフォーマンスの低下が発生します。

RDBMS 依存の同期スクリプト

Mobile Link では、データを同期するときに使用する規則を定義するために、同期スクリプトを使用しています。同期スクリプトでは次の内容を定義します。

- リモート・データベースからアップロードしたデータを、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リモート・データベースにダウンロードするデータ。

「同期スクリプトの作成」 327 ページを参照してください。

スクリプトに記述できるイベントの一覧については、「同期イベント」 359 ページを参照してください。

各タイプの統合データベースの固有の情報については、以下の項を参照してください。

- 「SQL Anywhere 統合データベース」 28 ページ
- 「Adaptive Server Enterprise 統合データベース」 10 ページ
- 「IBM DB2 LUW 統合データベース」 12 ページ
- 「IBM DB2 メインフレーム統合データベース」 15 ページ
- 「Microsoft SQL Server 統合データベース」 20 ページ
- 「MySQL 統合データベース」 22 ページ
- 「Oracle 統合データベース」 25 ページ

.NET 同期スクリプトと Java 同期スクリプト

データベースが使用している SQL 言語のバージョンで同期論理を記述できます。Java または .NET を使用して、より移植性が高く、強力なスクリプトを記述することもできます。Java と .NET は両方とも、各 RDBMS で SQL を使用した場合よりも高い柔軟性を提供し、SQL との完全な互換性も実現します。Java または .NET の同期論理を使用する場合は、セッション全体の変数の保持、ユーザ定義のプロシージャの作成、外部サーバに対する認証の統合などを行うことができます。

Java 同期論理については、「Java 同期論理の作成」 557 ページを参照してください。

.NET 同期論理については、「.NET での同期スクリプトの作成」 617 ページを参照してください。

スクリプトからのプロシージャの呼び出し

Microsoft SQL Server などのいくつかのデータベースでは、パラメータを持つプロシージャ・コールは、ODBC の構文を使用して記述する必要があります。

```
{ CALL procedure_name( {ml param1}, {ml param2}, ... ) }
```

プロシージャ定義の中でパラメータを OUT または INOUT として定義することで、戻り値を返すことができます。

CHAR カラム

他の多くの RDBMS では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート・データベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありませ

ん。SQL Anywhere を統合データベースとして使用していない場合は、統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv11 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

「-b オプション」 53 ページを参照してください。

データ変換

Mobile Link サーバが、SQL Anywhere 以外の統合データベースと通信するときに行われるデータ変換については、「リモート・データベースと統合データベース間での Mobile Link データ・マッピング」 781 ページを参照してください。

Adaptive Server Enterprise 統合データベース

Adaptive Server Enterprise を統合データベースとして設定する

Adaptive Server Enterprise を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *install-dir¥MobiLink¥setup* にある *syncase.sql* 設定スクリプトを実行します。
- Sybase Central の Mobile Link プラグインで、**[モード] - [管理]** を選択し、サーバ・データベースに接続します。データベース名を右クリックし、**[Mobile Link システム設定のチェック]** を選択します。データベースの設定が必要な場合は、続行のプロンプトが表示されます。
- **同期モデル作成ウィザード**または**同期モデル展開ウィザード**を使用する場合は、サーバ・データベースに接続するとシステム設定がチェックされます。データベースの設定が必要な場合は、続行のプロンプトが表示されます。

> **注意**
> 設定スクリプトを実行するデータベース・ユーザは、Mobile Link システム・テーブルを更新するパーミッションを持つ唯一のユーザです。Mobile Link アプリケーションを設定するには、このパーミッションが必要です。「必要なパーミッション」 31 ページを参照してください。
>
> Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システム・テーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

ODBC ドライバ

Adaptive Server Enterprise データベースで提供されている ODBC ドライバを使用して、Adaptive Server Enterprise 統合データベース用の ODBC DSN を設定してください。次の項を参照してください。

- Mobile Link の推奨 ODBC ドライバ
- Adaptive Server Enterprise のマニュアル

Adaptive Server Enterprise の問題

- **BLOB サイズ**　データ・サイズが 32 KB (デフォルト) より大きい BLOB データをダウンロードするには、次の操作を行います。
  - Windows の場合は、**[Adaptive Server Enterprise ODBC Driver Configuration]** ウィンドウの **[Advanced]** ページにある **[Text Size]** を想定される最大 BLOB サイズよりも大きい値に設定します。
  - Linux の場合は、*obdc.ini* ファイル内の TextSize エントリを想定される最大 BLOB サイズよりも大きい値に設定します。

- **CHAR カラム**　Adaptive Server Enterprise では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート・データベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありません。統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv11 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  詳細については、「-b オプション」 53 ページを参照してください。

- **データ型マッピング**　カラムのデータ型は、統合データベースとリモート・データベース間で完全に一致する必要があります。「Adaptive Server Enterprise データのマッピング」 782 ページを参照してください。

- **バージョン 11.5 以前に対する特別な注意事項**　Adaptive Server Enterprise 11.5 以前のバージョンに 255 バイトを超えるスクリプトを追加する場合は、ml_add_connection_script などの Mobile Link システム・プロシージャを使用できません。長いスクリプトを定義する場合は、Sybase Central を使用するか、直接挿入します。

- **VARBIT の制限**　Mobile Link では、長さ 0 (空) の VARBIT または LONG VARBIT 値と Adaptive Server Enterprise 統合データベースとの同期はサポートされていません。Adaptive Server Enterprise は VARBIT 型をサポートしていないので、これらの型は通常、Adaptive Server Enterprise データベースの VARCHAR または TEXT カラムと同期されます。Adaptive Server Enterprise では、空の文字列値は 1 つのスペースに変換されます。SQL Anywhere の VARBIT カラムではスペースを使用できないので、これらの値をダウンロードしようとすると、リモート・データベースでエラーとなります。

独立性レベル

「Mobile Link 独立性レベル」 170 ページを参照してください。

IBM DB2 LUW 統合データベース

Mobile Link は、Linux、UNIX、Windows 用の IBM DB2 LUW をサポートしています。AS/400 用の IBM DB2 はサポートしていません。

DB2 LUW を統合データベースとして設定する

DB2 を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *install-dir¥MobiLink¥setup* にある *syncdb2.sql* 設定スクリプトを実行します。スクリプトを実行する前に、別のロケーションにコピーして変更してください。手順については後述します。
- Sybase Central の Mobile Link プラグインで、**[モード] - [管理]** を選択し、サーバ・データベースに接続します。データベース名を右クリックし、**[Mobile Link システム設定のチェック]** を選択します。データベースの設定が必要な場合は、続行のプロンプトが表示されます。下記の手順 1 を実行する必要があることに注意してください。
- **同期モデル作成ウィザード**または**同期モデル展開ウィザード**を使用する場合は、サーバ・データベースに接続するとシステム設定がチェックされます。データベースの設定が必要な場合は、続行のプロンプトが表示されます。下記の手順 1 を実行する必要があることに注意してください。

注意

設定スクリプトを実行するデータベース・ユーザは、Mobile Link システム・テーブルを更新するパーミッションを持つ唯一のユーザです。Mobile Link アプリケーションを設定するには、このパーミッションが必要です。「必要なパーミッション」 31 ページを参照してください。

Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システム・テーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

♦ DB2 設定スクリプトを実行するには、次の手順に従います。

1. 設定スクリプトを使用して Mobile Link システム・テーブルをインストールするには、IBM DB2 LUW テーブル領域は最低でも 8 KB ページを使用します。テーブル領域が 8 KB ページを使用しない場合は、次の手順を実行します。
   - 1 つ以上のバッファ・プールに 8 KB ページがあることを確認します。ない場合は、8 KB ページのバッファ・プールを作成してください。
   - 8 KB ページのバッファ・プールを使用する、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。
2. 使用する接続情報を含むように、*syncdb2.sql* をカスタマイズします。
   a. *syncdb2.sql* を変更と保存を行う新しいロケーションにコピーします。

b. *syncdb2.sql* スクリプトには、デフォルトの接続文 connect to DB2Database が含まれています。DB2 データベースに接続するようにこの行を変更します。次の構文を使用します。

```
connect to DB2Database user userid using password ~
```

ここでは、*DB2Database*、*userid*、*password* に適切な名前を指定します。(*syncdb2.sql* スクリプトでは、チルダ (~) をコマンド・デリミタとして使用します。)

3. *syncdb2.sql* を実行します。

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

ODBC ドライバ

DB2 データベースで提供されている ODBC ドライバを使用して、DB2 統合データベース用の ODBC DSN を設定してください。次の項を参照してください。

- Mobile Link の推奨 ODBC ドライバ
- IBM DB2 LUW のマニュアル

DB2 LUW の問題

- **テーブル領域の容量**　DB2 LUW データベースを統合データベースとして使用する場合のテーブル領域とテンポラリ・テーブル領域は、8 KB ページを使用します。

  また、LONG テーブル領域が必要なカラムもあります。次の例のように、デフォルトの LONG テーブル領域がない場合は、これらのカラムを含むテーブルの作成文を正しく設定します。

  ```
  CREATE TABLE ... ( ... )
  IN tablespace
  LONG IN long-tablespace
  ```

  サンプル・アプリケーションの使用例については、「Mobile Link CustDB サンプルの解説」『Mobile Link - クイック・スタート』を参照してください。

- **セッション全体の変数**　バージョン 8 より前の DB2 LUW では、セッション全体の変数はサポートされていません。これを解決する便利な方法としては、Mobile Link ユーザ名と他のセッション・データ用のカラムがあるベース・テーブルを使用します。ベース・テーブルには、同時同期を表すローが含まれます。

- **ユーザ定義のプロシージャ**　バージョン 8.2 より前の DB2 LUW では、SQL プロシージャを実行可能ライブラリ (DLL など) にコンパイルする必要があります。作成された DLL/共有ライブラリは、サーバ上の特別なディレクトリにコピーする必要があります。C/C++ や Java など、さまざまな言語を使用してプロシージャを記述できることに注意してください。

  Java と .NET の同期スクリプトの詳細については、以下の項目を参照してください。

  ○ 「Java による同期スクリプトの作成」 553 ページ
  ○ 「.NET での同期スクリプトの作成」 617 ページ

- **CHAR カラム**　IBM DB2 LUW では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート・データベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め

込まれることはありません。統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv11 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

「-b オプション」 53 ページを参照してください。

- **データ型マッピング**　カラムのデータ型は、統合データベースとリモート・データベース間で完全に一致する必要があります。詳細については、「IBM DB2 LUW データのマッピング」 791 ページを参照してください。
- **システム・プロシージャ・コール内の引用符を 2 つにする**　Mobile Link システム・プロシージャを使用して、スクリプトを DB2 統合データベースに追加する場合は、引用符を 2 つにする必要があります。たとえば、ml_add_table_script を使用して追加するスクリプトに、他の統合データベースに対する SET "DELETED"="Y" という行が含まれている場合、DB2 では、これを SET "DELETED" = ""Y"" と記述する必要があります。
- **バージョン 5 以前に対する特別な注意事項**　バージョン 6 より前の IBM DB2 LUW を使用している場合、カラム名とその他の識別子には 18 文字までしか使用できません。したがって、Mobile Link システム・プロシージャの名前をトランケートする必要があります。たとえば、ml_add_connection_script を呼び出すには、ml_add_connection_ という名前を使用します。

独立性レベル

「Mobile Link 独立性レベル」 170 ページを参照してください。

IBM DB2 メインフレーム統合データベース

DB2 メインフレームを統合データベースとして設定する

DB2 メインフレームを Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、ビューを追加する必要があります。このタスクは SQL メソッドまたは JCL メソッドを使用して実行できます。

注意
設定スクリプトを実行するデータベース・ユーザは、Mobile Link システム・テーブルを更新するパーミッションを持つ唯一のユーザです。Mobile Link アプリケーションを設定するには、このパーミッションが必要です。「必要なパーミッション」 31 ページを参照してください。

Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システム・テーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

♦ DB2 メインフレーム環境を設定するための一般的な手順

1. Mobile Link スキーマ用にページ・サイズが 8 K 以上でローレベルのロックが設定されたバッファ・プールを作成します。ローレベルのロックは、同じテーブルに対する同時同期を処理するために必要です。この例では、バッファ・プールに *BP8K* という名前を付けます。
2. Mobile Link スキーマ用にページ・サイズが 8 K のバッファ・プールを持つ *MLTB8K* という名前のテーブル領域を作成します。次に例を示します。

   ```
   create tablespace MLTB8K in IANY bufferpool BP8K locksize row
   grant use of tablespace IANY.MLTB8K to public
   ```

3. Mobile Link スキーマ・プロシージャ用の Workload Manager 環境がまだない場合は作成し、*MLWLM* というような名前を付けます。
4. DB2 データベースで提供されている ODBC ドライバを使用して、DB2 メインフレーム統合データベース用の ODBC DSN を設定します。次の項を参照してください。
   - Mobile Link の推奨 ODBC ドライバ
   - IBM DB2 メインフレームのマニュアル

♦ SQL を使用して Mobile Link システム・テーブルを作成するには、次の手順に従います。

注意
SQL メソッドには、DSNTPSMP を使用してストアド・プロシージャを作成する機能が必要です。SQL ストアド・プロシージャを有効にしていない場合は、JCL による方法を使用してください。

1. 「DB2 メインフレームを統合データベースとして設定する」 15 ページに示す一般的な手順を使用して、DB2 メインフレーム環境を設定します。
2. *install-dir¥MobiLink¥setup* にある *syncd2m.sql* 設定スクリプトを変更します。

注意
続行する前に、元の *syncd2m.sql* ファイルのバックアップ・コピーを作成してください。

syncd2m.sql ファイルで、次の記述をすべて置換します。

- {MLTABLESPACE} をテーブル領域の名前である MLTB8K に置換します。
- {WLMENV} を Workload Manager、つまり MLWLM に置換します。

3. 次のコマンド・ラインを使用して *syncd2m.sql* 設定スクリプトを実行します。

```
dbisql -c "uid=user-id;pwd=password;DSN=dsn-name" -nogui syncd2m.sql
```

メッセージ・ログ・ファイル *syncd2m.txt* が生成されます。

4. *syncd2m.txt* を開いて、DSNTPSMP 呼び出しが成功したことを確認します。

♦ JCL を使用して Mobile Link システム・テーブルを作成するには、次の手順に従います。

1. 「DB2 メインフレームを統合データベースとして設定する」 15 ページに示す一般的な手順を使用して、DB2 メインフレーム環境を設定します。
2. *install-dir¥MobiLink¥setup* にある *syncd2m_jcl.sql* 設定スクリプトを変更します。

注意
続行する前に、元の *syncd2m_jcl.sql* ファイルのバックアップ・コピーを作成してください。

syncd2m_jcl.sql ファイルで、次の記述をすべて置換します。

- {MLTABLESPACE} を修飾されたテーブル領域 (たとえば MYDB.MYTS) に置換します。
- {WLMENV} を DB2 インスタンスに割り当てられた Workload Manager の名前に置換します。

3. DBISQL を開始し、DB2 メインフレームに接続します。
4. *install-dir¥MobiLink¥setup* にある *syncd2m_jcl.sql* 設定スクリプトの編集済みコピーを実行して、DB2 メインフレームに Mobile Link テーブルを作成し Mobile Link プロシージャを定義します。
5. *%SQLANY%¥MobiLink¥setup* ディレクトリからメインフレームに FTP でアクセスし、次のコマンドを実行します。

```
bin
   hash
   cd xmit
   quote site recfm=fb lrecl=80
   quote site cyl
   put d2mload.xmit
   put d2mdbrm.xmit
   quit
```

6. メインフレームには次の 2 つの xmit ファイルがあります。
   - USERID.XMIT.D2MLOAD.XMIT
   - USERID.XMIT.D2MDBRM.XMIT

USERID は、FTP 経由の接続時に指定したユーザ名です。

7. ターミナル・セッションを開き、ISPF コマンド・シェルから次のコマンドを実行します。

```
RECEIVE INDATASET('USERID.XMIT.D2MLOAD.XMIT')
  RECEIVE INDATASET('USERID.XMIT.D2MDBRM.XMIT')
```

   出力は次のとおりです。

   - USERID.ML.LOADLIB
   - USERID.ML.DBRMLIB

8. *d2mrelod.jcl* ファイルをコピーして次のように変更します。
   - USERID をメインフレームのユーザ ID に変更します。
   - DSNDB0T を DB2 DSN に変更します。
9. *install-dir¥MobiLink¥setup* にある *d2mrelod.jcl* スクリプトの編集済みコピーを実行します。
10. *d2mbdpk.jcl* ファイルをコピーして次のように変更します。
    - USERID をメインフレームのユーザ ID に変更します。
    - DB0T を DB2 SSID に変更します。
11. *d2mbdpk.jcl* の編集済みコピーを実行して、すべての SQL プロシージャをバインドします。次に、SQL プロシージャからロード・モジュール名へのマッピングのリファレンスを示します。

プロシージャ名	ロード・モジュール名
ml_add_user	mlaub
ml_delete_user	mldub
ml_del_sstate	mldssb
ml_reset_sstate	mlrssb
ml_del_sstate_b4	mldssbb
ml_add_lcs_chk	mlalcscb
ml_add_lcs	mlalcsb
ml_add_cs	mlacsb
ml_add_jcs	mlajcsb
ml_add_dcs	mladcsb
ml_add_lts_chk	mlaltscb

プロシージャ名	ロード・モジュール名
ml_add_lts	mlaltsb
ml_add_ts	mlatsb
ml_add_jts	mlajtsb
ml_add_dts	mladtsb
ml_add_property	mlapb
ml_add_column	mlacb
ml_set_device	mlsdb
ml_set_device_nt	mlsdnb
ml_set_dev_addr	mlsdab
ml_set_dev_addr_int	mlsdanb
ml_set_listening	mlslb
ml_set_listen_nt	mlslnb
ml_set_sis_sstate	mlssssb
ml_del_dev_addr	mlddab
ml_del_listen	mldlb
ml_delete_device	mlddb

DB2 メインフレームの既知の問題

- **DB2 メインフレームがモデル・モードで動作しない**　**同期モデル作成ウィザード**を使用している場合は、DB2 メインフレームを統合データベースとして使用することはできません。
- **SELECT 文には FOR READ ONLY 句が必要**　DB2 メインフレームの SELECT 文は、デフォルトで更新用に開かれます。つまり、データベースは、SELECT 文の後に UPDATE 文があることを予測して書き込みロックを取得します。

  書き込みロックを回避し、同時実行性を強化するには、UPDATE 文に先行しないすべての SELECT 文で FOR READ ONLY を追加します。SELECT 文 (特に download_cursor スクリプトと download_delete_cursor スクリプトの SELECT 文) では、可能な限り頻繁に FOR READ ONLY を使用します。
- **Sysplex では時間の同期が必要**　DB2 メインフレーム統合データベースが Sysplex で実行されている場合、Sysplex にあるすべての LPAR のクロックは同期されている必要があります。クロックを同期できない場合、データベースの同期中にデータが失われる可能性があります。

- **数字が概数になる**　概数には、可能なさまざまな値があります。次の表に例を示します。

型	入力値	DB2 メインフレームの値	ASA の値
Real	123.456	123.4559936523	123.4560012817
Float	123.456	123.45599999999999	123.4560012817
Double	123.456	123.45599999999999	123.456

DB2 メインフレーム統合データベースとの間では、倍精度カラムと浮動小数点カラムを同期しないことをおすすめします。

独立性レベル

「Mobile Link 独立性レベル」 170 ページを参照してください。

Microsoft SQL Server 統合データベース

Microsoft SQL Server を統合データベースとして設定する

注意
設定スクリプトを実行するデータベース・ユーザは、テーブル、トリガ、ストアド・プロシージャを作成できる必要があるため、db_owner ロールを持っている必要があります。

Microsoft SQL Server を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *install-dir¥MobiLink¥setup* にある *syncmss.sql* 設定スクリプトを実行します。
- まず、Sybase Central の Mobile Link プラグインで、**[モード] - [管理]** を選択します。次に、サーバ・データベースに接続し、データベース名を右クリックして、**[Mobile Link システム設定のチェック]** を選択します。データベースの設定が必要な場合は、続行のプロンプトが表示されます。既存の Mobile Link システム設定を使用する場合、default_schema を Mobile Link システム設定のスキーマにしてください。
- **同期モデル作成ウィザード**または**同期モデル展開ウィザード**を使用する場合は、サーバ・データベースに接続するとシステム設定がチェックされます。データベースの設定が必要な場合は、続行のプロンプトが表示されます。

注意
設定スクリプトを実行するデータベース・ユーザは、Mobile Link システム・テーブルを更新するパーミッションを持つ唯一のユーザです。Mobile Link アプリケーションを設定するには、このパーミッションが必要です。「必要なパーミッション」 31 ページを参照してください。

Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システム・テーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

ODBC ドライバ

SQL Server データベースで提供されている ODBC ドライバを使用して、SQL Server 統合データベース用の ODBC DSN を設定してください。次の項を参照してください。

- Mobile Link の推奨 ODBC ドライバ
- Microsoft SQL Server のマニュアル

SQL Server の問題

- **SET NOCOUNT ON**　Microsoft SQL Server の場合には、すべてのストアド・プロシージャまたは ODBC を使用して実行される SQL バッチの最初の文として SET NOCOUNT ON を指定してください。このオプションがないと、ネットワーク・バッファでオーバフローが発生し

て、何も通知されずにデータが失われる可能性があります。これは SQL Server の既知の問題です。

- **プロシージャ・コール**　Microsoft SQL Server では、パラメータを持つプロシージャ・コールは、ODBC の構文を使用して記述する必要があります。

  ```
  { CALL procedure_name( {ml param1}, {ml param2}, ... ) }
  ```

- **CHAR カラム**　Microsoft SQL Server では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート・データベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありません。統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv11 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  「-b オプション」 53 ページを参照してください。

- **データ型マッピング**　カラムのデータ型は、統合データベースとリモート・データベース間で完全に一致する必要があります。詳細については、「Microsoft SQL Server データのマッピング」 812 ページを参照してください。

- **サンプル・データベースの問題**　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。計算カラムは同期できません。カラムをダウンロードのみに設定することと、カラムを同期から除外することは可能です。

- **upload_update スクリプトでの競合検出の実装**　SQL Server の NOCOUNT オプションの動作は、Mobile Link サーバで、アップロード・スクリプトによって変更されたローの数を正確に評価できない場合があることを意味します。SQL Server で競合を検出するには、upload_update スクリプトにストアド・プロシージャを実装する方が安全です。

独立性レベル

「Mobile Link 独立性レベル」 170 ページを参照してください。

MySQL 統合データベース

Mobile Link サーバは、MySQL Community サーバと Enterprise サーバ 5.1.22 以降をサポートしています。QAnywhere モデルと Mobile Link モデルは、MySQL をサポートしていません。

MySQL を統合データベースとして設定する

MySQL を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、ビューを追加する必要があります。これは、次のような 2 つの方法で実行できます。

- MySQL コマンド・ライン・ツールまたは MySQL Query Browser を使用して、*install-dir ¥MobiLink¥setup* にある *syncmys.sql* 設定スクリプトを実行します。MySQL ユーザ ID にトリガを作成する権限があることを確認してください。
- Sybase Central の Mobile Link プラグインで、**[モード] - [管理]** を選択し、サーバ・データベースに接続します。データベース名を右クリックし、**[Mobile Link システム設定のチェック]** を選択します。データベースの設定が必要な場合は、続行のプロンプトが表示されます。既存の Mobile Link システム設定を使用する場合、default_schema を Mobile Link システム設定のスキーマにしてください。

注意

設定スクリプトを実行するデータベース・ユーザは、Mobile Link システム・テーブルを更新するパーミッションを持つ唯一のユーザです。Mobile Link アプリケーションを設定するには、このパーミッションが必要です。「必要なパーミッション」 31 ページを参照してください。

Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システム・テーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

ODBC ドライバ

MySQL Web サイトで提供されている ODBC ドライバを使用して、MySQL 統合データベース用の ODBC DSN を設定してください。Mobile Link サーバは、MySQL ODBC ドライバ 5.1.3 以降をサポートしています。次の項を参照してください。

- Mobile Link の推奨 ODBC ドライバ

UINIX で ODBC 設定ファイルを指定するには、次のいずれかを行います。

- *ODBC.INI* ファイルを現在のユーザのホーム・ディレクトリに配置します。
- **ODBCINI** 環境変数を作成し、*ODBC.INI* ファイルのディレクトリ・ロケーションに設定します。

同期スクリプトに、セミコロンで区切られた、バッチによる SQL コマンドが含まれていても、Mobile Link サーバが MySQL データベースへの接続を作成するための DSN を設定するときに、**[MySQL Connector/ODBC Data Source Configuration]** ウィンドウの **[Flags 3]** ページで **[Allow Multiple Statements]** チェックボックスをオンにすることが必要な場合があります。

MySQL の問題

- **ストレージ・エンジン**　Mobile Link サーバでは、デフォルトのストレージ・エンジンが ACID に準拠していることを必要としています。デフォルトのストレージ・エンジンが ACID に準拠していない場合は、Mobile Link サーバのすべてのテーブルが、InnoDB、Falcon など、ACID 準拠のストレージ・エンジンを使用して作成されていることを確認してください。
- **[ストアド・プロシージャ]**　ストアド・プロシージャ・コールでは、INOUT パラメータおよび OUT パラメータを使用することはできません。これらのパラメータが必要なプロシージャは、OUT 値を返す関数として実装してください。

  authenticate_user、modify_user などの INOUT パラメータが必要なサーバ・イベントは、関数として実装し、CALL 文ではなく SELECT 文を使用して実行してください。

  ユーザ定義の名前付きパラメータは、サーバ・イベントの実行後に変更されないため、サポートされていません。
- **カーソル・スクリプト**　イベント upload_fetch、download_cursor、download_delete_cursor は、SELECT 文を使用して呼び出してください。SELECT 文は、Mobile Link サーバによって、コミットされた読み出し独立性レベルを使用して実行されます。MySQL ODBC ドライバのバグによって、INSERT 文、UPDATE 文、DELETE 文など、コミットされない読み出し操作がサーバで発生し、統合データベースとリモート・データベースの間でデータの一貫性が失われます。

  この問題を回避するには、SELECT 文に LOCK IN SHARE MODE 句を追加します。次に例を示します。

  ```
  SELECT column1 FROM table1 WHERE id > 0 LOCK IN SHARE MODE
  ```

  この句は、コミットされない操作から SELECT 文を保護します。
- **バルク・アップロード**　Mobile Link サーバは、MySQL ODBC ドライバに依存しています。MySQL ODBC ドライバは現在、バルク・アップロードをサポートしていません。
- **MLSD**　Mobile Link サーバは、MySQL ODBC ドライバに依存しています。MySQL ODBC ドライバは現在、MSDTC をサポートしていません。
- **UNIX 用 64 ビット Mobile Link サーバの SQLLEN データ型**　MySQL ODBC ドライバは SQLLEN を 32 ビット整数として定義しているため、SQLLEN を 64 ビット整数として定義している 64 ビットの Mobile Link サーバと不一致が発生します。64 ビットの UNIX 環境で Mobile Link を実行している場合は、ODBC 設定ファイルに次を追加してください。

  ```
  length32=1
  ```

  このエントリによって、サーバは、SQLLEN を 32 ビット整数として読み込むことが強制されます。設定は、次の例のようになります。

```
[a_mysql_dsn]
Driver=full_path/libmyodbc5.so
server=host_name
uid=user_name
pwd=user_password
database=database_name
length32=1
```

- **MySQL サーバの設定**　Mobile Link 同期スクリプトは、ml_script テーブルに TEXT として格納され、必要に応じて取得されます。*my.ini* ファイルの max_allowed_packet は、16m 以上に設定することが必要な場合があります。

独立性レベル

「Mobile Link 独立性レベル」 170 ページを参照してください。

Oracle 統合データベース

Oracle を統合データベースとして設定する

Oracle を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *install-dir¥MobiLink¥setup* にある *syncora.sql* 設定スクリプトを実行します。
- Sybase Central の Mobile Link プラグインで、**[モード] - [管理]** を選択し、サーバ・データベースに接続します。データベース名を右クリックし、**[Mobile Link システム設定のチェック]** を選択します。データベースの設定が必要な場合は、続行のプロンプトが表示されます。既存の Mobile Link システム設定へのエイリアスを設定している場合は、Mobile Link システム設定があるスキーマのユーザとして接続してください。
- **同期モデル作成ウィザード**または**同期モデル展開ウィザード**を使用する場合は、サーバ・データベースに接続するとシステム設定がチェックされます。データベースの設定が必要な場合は、続行のプロンプトが表示されます。

注意

設定スクリプトを実行するデータベース・ユーザは、Mobile Link システム・テーブルを更新するパーミッションを持つ唯一のユーザです。Mobile Link アプリケーションを設定するには、このパーミッションが必要です。「必要なパーミッション」 31 ページを参照してください。

Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システム・テーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

ODBC ドライバ

Oracle 統合データベースには、ODBC DSN を設定する必要があります。次の項を参照してください。

- 「iAnywhere Solutions Oracle ドライバ」 843 ページ
- Mobile Link の推奨 ODBC ドライバ

Oracle の問題

- **ストアド・プロシージャ**　Oracle でストアド・プロシージャを使用している場合は、Oracle ODBC ドライバに対して [プロシージャは結果を返す] オプションを選択してください。

  「iAnywhere Solutions Oracle ドライバ」 843 ページを参照してください。

- **セッション全体の変数**　Oracle ではセッション全体の変数は提供されていません。セッション全体の情報を保持するには、Oracle パッケージ内の変数を使用します。Oracle パッケージでは、変数の作成、修正、破棄が可能です。これらの変数は Oracle パッケージが現在のパッケージであるかぎり有効です。

- **オートインクリメント・メソッド**　プライマリ・キーの一意性を維持するには、Oracle シーケンスを使用して、オートインクリメント・フィールドのキーのリストに似た、キーのリストを生成します。CustDB サンプル・データベースの *Samples¥MobiLink¥CustDB¥custora.sql* に、サンプル・コーディングが用意されています。ただし、オートインクリメントとは異なり、シーケンスを明示的に参照する必要があります。オートインクリメントは、INSERT 文でカラムが参照されていない場合は、自動的にカラム値を挿入します。
- **Oracle では空の文字列をサポートしていない**　Oracle では、空の文字列は NULL として処理されます。SQL Anywhere と Ultra Light では、空の文字列は NULL とは異なる意味を持ちます。したがって、Oracle 統合データベースがある場合には、クライアント・データベースで空の文字列を使用しないようにしてください。
- **CHAR カラム**　Oracle では、CHAR データ型は固定長で、文字列の長さに合わせてブランクが埋め込まれています。Mobile Link リモート・データベース (SQL Anywhere または Ultra Light) では、CHAR は VARCHAR と同じで、固定幅に合わせて値にブランクが埋め込まれることはありません。統合データベースで CHAR の代わりに VARCHAR を使用することを強くおすすめします。CHAR を使用する必要がある場合は、mlsrv11 -b コマンド・ライン・オプションを使用すると、同期中に文字列から後続ブランクを削除できます。このオプションは、重複を検出するときに使用する文字列の比較にとって重要です。

  「-b オプション」 53 ページを参照してください。
- **データ型マッピング**　カラムのデータ型は、統合データベースとリモート・データベース間で完全に一致する必要があります。詳細については、「Oracle データのマッピング」 826 ページを参照してください。

独立性レベル

「Mobile Link 独立性レベル」 170 ページを参照してください。

Oracle varray の使用

Oracle 用の ODBC ドライバである iAnywhere Solutions 11 - Oracle では、ストアド・プロシージャでの Oracle varray の使用がサポートされます。ストアド・プロシージャ内に作成されたアップロード・スクリプト (upload_insert、upload_update、upload_delete) で varray を使用すると、ストアド・プロシージャ内に作成されたアップロード・スクリプトで varray を使用しない場合と比べて、Mobile Link サーバのパフォーマンスが向上することがあります。ストアド・プロシージャを使用しない INSERT、UPDATE、DELETE などの単純な SQL 文を使用すると、通常はパフォーマンスが最高になるという利点はありますが、ストアド・プロシージャを使用することにより (varray を使用する方法を含む)、ビジネス論理を適用できるようになります。

varray の例

次に、varray を使用する簡単な例を示します。

1. 3 つのカラムが含まれる my_table という名前のテーブルを作成します。

   ```
   create table my_table ( pk integer primary key, c1 number(20), c2 varchar2(4000) )
   ```
2. varray を使用するユーザ定義のコレクション型を作成します。


```
create type my_integer is varray(100) of integer;
create type my_number is varray(100) of number(20);
create type my_varchar is varray(100) of varchar2(8000);
```

my_varchar は、100 個の要素を含む varray として定義されます。各要素のデータ型は varchar2 で、幅は 8000 です。この幅は、my_table に指定された幅の 2 倍の大きさです。

3. 挿入用のストアド・プロシージャを作成します。

```
create or replace procedure my_insert_proc( pk_v my_integer, c1_v my_number, c2_v my_varchar )
is
c2_value my_varchar;
begin
    c2_value := c2_v;  -- Work around an Oracle bug
    FORALL i in 1 .. pk_v.COUNT
        insert into my_table ( pk, c1, c2 ) values( pk_v(i), c1_v(i), c2_value(i) );
end;
```

varray の制限

ストアド・プロシージャで varray を使用するときは、次の制限があります。

- DSN では、**[Microsoft 分散トランザクションを有効にする]** チェックボックスがオフにされている必要があります。
- BLOB と CLOB の varray はサポートされません。
- varray が含まれるストアド・プロシージャは、パッケージ化されたプロシージャではなく、スタンドアロン・プロシージャでなければなりません。
- varray のデータ型が CHAR、VARCHAR、NCHAR、または NVARCHAR である場合、ユーザ定義の varray 型は、テーブルのカラムに指定された長さの 2 倍の大きさでなければなりません。
- Mobile Link サーバから Oracle 統合データベースに送信される varray のローの数は、varray 型で宣言された varray のサイズではなく、-s オプションで設定されます。-s オプションは、同期スクリプトで使用されている varray 型のうち最小のサイズより大きくしてはなりません。大きくすると、Mobile Link サーバはエラーを発行します。「-s オプション」 91 ページを参照してください。

SQL Anywhere 統合データベース

SQL Anywhere を統合データベースとして設定する

SQL Anywhere を Mobile Link 統合データベースとして動作するよう設定するには、設定プロシージャを実行して、Mobile Link 同期に必要な Mobile Link システム・テーブル、ストアド・プロシージャ、トリガ、ビューを追加する必要があります。次のような方法で実行できます。

- *install-dir¥MobiLink¥setup* にある *syncsa.sql* 設定スクリプトを実行します。
- Sybase Central の Mobile Link プラグインで、**[モード] - [管理]** を選択し、サーバ・データベースに接続します。データベース名を右クリックし、**[Mobile Link システム設定のチェック]** を選択します。データベースの設定が必要な場合は、続行のプロンプトが表示されます。
- **同期モデル作成ウィザード**または**同期モデル展開ウィザード**を使用する場合は、サーバ・データベースに接続するとシステム設定がチェックされます。データベースの設定が必要な場合は、続行のプロンプトが表示されます。

注意

設定スクリプトを実行するデータベース・ユーザは、Mobile Link システム・テーブルを更新するパーミッションを持つ唯一のユーザです。Mobile Link アプリケーションを設定するには、このパーミッションが必要です。「必要なパーミッション」 31 ページを参照してください。

Mobile Link サーバを統合データベースに接続するために使用される RDBMS ユーザは、Mobile Link システム・テーブル、プロシージャなどを修飾子なしで使用できる必要があります (SELECT * from ml_user など)。「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

ODBC ドライバの設定

SQL Anywhere 統合データベースには、ODBC DSN を設定する必要があります。SQL Anywhere 用の ODBC ドライバは、SQL Anywhere とともにインストールされます。

SQL Anywhere ODBC ドライバについては、「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。

独立性レベル

「Mobile Link 独立性レベル」 170 ページを参照してください。

Mobile Link サーバ

目次

Mobile Link サーバの実行

すべての Mobile Link クライアントは、Mobile Link サーバを介して同期します。データベース・サーバには、直接接続できません。Mobile Link サーバを起動してから、Mobile Link クライアントの同期を行います。

mlsrv11 コマンド・ライン・オプションのリストは、「Mobile Link サーバ・オプション」 43 ページを参照してください。

Mobile Link サーバは、統合データベース・サーバとの接続を ODBC を介して開きます。その後、リモート・アプリケーションからの接続を受け入れ、同期処理を制御します。

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

● mlsrv11 を実行します。-c オプションを使用して統合データベースの ODBC 接続パラメータを指定します。

接続パラメータの詳細については、「-c オプション」 56 ページを参照してください。

接続パラメータの指定は必須です。他のオプションは、必要に応じて使用します。これらのオプションを使用すると、サーバの動作方法を指定できます。たとえば、キャッシュ・サイズ・オプションとログ・オプションを指定できます。

mlsrv11 オプションの詳細については、「mlsrv11 の構文」 45 ページを参照してください。

注意
mlsrv11 のオプションを指定すると、Mobile Link サーバの動作方法を指定できます。サーバの動作を制御するには、同期イベントで呼び出されるスクリプトを定義します。「同期イベント」 359 ページを参照してください。

例

次の例は、ODBC データ・ソース *SQL Anywhere 11 CustDB* を使用して Mobile Link サーバを起動して、統合データベースを識別します。コマンド全体を 1 行に入力してください。

```
mlsrv11
  -c "dsn=SQL Anywhere 11 CustDB;uid=ml_server;pwd=sql"
  -zs MyServer
  -o mlsrv.log
  -vcr
  -x tcpip(port=3303)
  -xo tcpip
```

この例では -c オプションで、ODBC データ・ソース名 (DSN) と認証を含む接続文字列を指定しています。-zs オプションでサーバ名を指定しています。-o オプションは、ログ・ファイル名を *mlsrv.log* と指定します。-vcr オプションが指定されているので、*mlsrv.log* の内容は冗長です。-x オプションはバージョン 10 と 11 のクライアント用のポートを開き、-xo オプションはバージョン 8 と 9 のクライアント用のポートを開きます。-x オプションまたは -xo オプションのどちらかでポートを指定してください。どちらも指定しなかった場合、両方のオプションに対してデフォルトのポートが使用されるため、コマンドは失敗します。

Mobile Link サーバを Windows サービスまたは UNIX デーモンとして起動することもできます。「現在のセッション外での Mobile Link サーバの起動」 35 ページを参照してください。

必要なパーミッション

Mobile Link サーバをデータベース・サーバに接続するには、データベース・ユーザを指定する必要があります。mlsrv11 -c オプションまたは DSN でデータベース・ユーザを指定します。

このデータベース・ユーザには Mobile Link システム・テーブルに対する完全な SELECT、INSERT、UPDATE、および DELETE パーミッションが必要です。また、Mobile Link システム・プロシージャに対する EXECUTE パーミッションも必要です。デフォルトでは、Mobile Link 設定スクリプトを実行するデータベース・ユーザはこれらのパーミッションを持っています。別のデータベース・ユーザを使用して Mobile Link サーバを実行する場合は、そのユーザに ml_* テーブルと ml_add_*_script システム・プロシージャに対するパーミッションを付与してください。

たとえば、SQL Anywhere 統合データベースでは、必要なパーミッションを次のように設定できます。

```
CREATE USER DBUser IDENTIFIED BY SQL;
GRANT ALL ON dbo.ml_user to DBUser;
...
GRANT EXECUTE ON dbo.ml_add_table_script TO DBUser;
...
```

各 Mobile Link システム・テーブルおよびシステム・プロシージャについてパーミッションを与える必要があります。すべての Mobile Link システム・テーブルとシステム・プロシージャのリストについては、「Mobile Link サーバのシステム・テーブル」 733 ページと「Mobile Link サーバ・システム・プロシージャ」 697 ページを参照してください。

このデータベース・ユーザは、Mobile Link スクリプトで参照されているすべてのテーブルに対する適切なパーミッションと、Mobile Link スクリプトで参照されているプロシージャに対する EXECUTE パーミッションも必要です。

パーミッションの設定の詳細については、「GRANT 文」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

設定スクリプトの詳細については、「統合データベースの設定」 6 ページを参照してください。

Mobile Link サーバの停止

Mobile Link サーバは、サーバを起動したコンピュータから停止します。次の方法で、Mobile Link サーバを停止できます。

- Mobile Link 停止ユーティリティ (mlstop) を使用する
- Mobile Link サーバ・ウィンドウで **[シャットダウン]** をクリックする
- Windows のシステム・トレイ内のアイコンを右クリックして、**[シャットダウン]** を選択する
- UNIX 上で実行中に Mobile Link サーバ・ウィンドウが開いていない状態で、Q と入力する
- Mobile Link サーバ API の shutdown メソッドを使用する

参照

- 「Mobile Link 停止ユーティリティ (mlstop)」 727 ページ
- Java 用サーバ API：「shutdown メソッド」 601 ページ
- .NET 用サーバ API：「ShutDown メソッド」 660 ページ

Mobile Link サーバの動作のロギング

サーバの動作をロギングすると、開発プロセスとトラブルシューティングのときに特に役立ちます。パフォーマンスが低下するため、運用環境の通常の操作には冗長出力を使用しないでください。

ファイルへのロギング結果の出力

ロギング結果は、Mobile Link サーバのメッセージ・ウィンドウに送信されます。また、-o オプションを使用して結果をメッセージ・ログ・ファイルにも送信できます。次のコマンドでは、結果を *mlsrv.log* という名前のメッセージ・ログ・ファイルに送ります。

```
mlsrv11 -o mlsrv.log -c ...
```

ログ・ファイルのサイズを制御したり、ファイルが最大サイズに達したときの処理を指定したりできます。

- ログ・ファイルを使用することを指定する場合は -o オプションを使用します。
- -ot オプションを使用して、ログ・ファイルを使用することを指定すると、メッセージが送信される前にログ・ファイルの前の内容が削除されます。
- -o または -ot に加えて -on オプションを使用してサイズを指定すると、そのサイズに達したときに、これまでのログ・ファイルが拡張子 .old の付いた名前に変更され、元の名前を持つ新しいファイルが使用されます。
- -o または -ot に加えて -os オプションを使用してサイズを指定すると、そのサイズに達したときに、日付と連番に基づいた新しい名前を持つ新しいログ・ファイルが使用されます。

次の項を参照してください。

- 「-o オプション」 79 ページ
- 「-on オプション」 80 ページ
- 「-os オプション」 82 ページ
- 「-ot オプション」 83 ページ

ロギング結果の出力容量の制御

-v オプションを使用すると、メッセージ・ログ・ファイルに記録され、Mobile Link サーバのウィンドウに表示される情報を制御できます。「-v オプション」 104 ページを参照してください。

レポートされる警告メッセージの制御

レポートされる警告メッセージを制御することもできます。

詳細については、次の項を参照してください。

- 「-zw オプション」 127 ページ
- 「-zwd オプション」 128 ページ
- 「-zwe オプション」 129 ページ

Mobile Link サーバ・ログの表示

Mobile Link ログは次の方法で表示できます。

- Mobile Link サーバのメッセージ・ウィンドウの場合
- ログ・ファイルを開く
- Sybase Central で Mobile Link サーバ・ログ・ファイル・ビューワを使用する

Mobile Link サーバ・メッセージ・ウィンドウ外でログ情報を表示するには、情報をファイルに記録する必要があります。「ファイルへのロギング結果の出力」 33 ページを参照してください。

Mobile Link サーバ・ログ・ファイル・ビューワ

Mobile Link サーバ・ログを表示するには、Sybase Central を開き、**[ツール]** - **[Mobile Link 11]** - **[Mobile Link サーバ・ログ・ファイル・ビューワ]** を選択します。表示するログ・ファイルを選択するよう要求されます。Shift キーを押しながら選択すると、複数のログ・ファイルを同時に開くことができます。

Mobile Link サーバ・ログ・ファイル・ビューワは、Mobile Link ログ・ファイルに格納された情報を読み取ります。Mobile Link サーバに接続したりログ・ファイルの構成を変更したりすることはありません。

Mobile Link サーバ・ログ・ファイル・ビューワでは、表示する情報をフィルタリングできます。また、ログの情報に基づく統計情報も表示できます。

現在のセッション外での Mobile Link サーバの起動

Mobile Link サーバは、常時利用できるように設定できます。これを簡単に行うには、コンピュータをログオフしてもサーバは稼働し続けるように、Windows 版または UNIX 版の Mobile Link サーバを実行します。これを行う方法は、使用するオペレーティング・システムによって異なります。

- **UNIX デーモン**　mlsrv11 -ud オプションを使用すると、Mobile Link サーバをデーモンとして実行できます。これによって、Mobile Link サーバをバックグラウンドで実行し、ログオフ後も引き続き実行させることができます。
- **Windows サービス**　Windows Mobile Link サーバを、サービスとして実行させることができます。

  サービスとして実行されている Mobile Link サーバを停止するには、mlstop、dbsvc、または Windows サービス・マネージャを使用できます。

参照

- 「-ud オプション」 101 ページ
- 「Linux 用サービス・ユーティリティ (dbsvc)」 『SQL Anywhere サーバ - データベース管理』
- 「現在のセッション外でのサーバの起動」 『SQL Anywhere サーバ - データベース管理』

デーモンとしての UNIX Mobile Link サーバの実行

UNIX において Mobile Link サーバをバックグラウンドで実行し、現在のセッションから独立して稼働させるには、Mobile Link サーバを「デーモン」として起動します。

♦ **UNIX Mobile Link サーバをデーモンとして実行するには、次の手順に従います。**

- Mobile Link サーバの起動時に、-ud オプションを使用する。次に例を示します。

  ```
  mlsrv11 -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql" -ud
  ```

「-ud オプション」 101 ページを参照してください。

参照

- 「Linux 用サービス・ユーティリティ (dbsvc)」 『SQL Anywhere サーバ - データベース管理』

サービスとしての Windows Mobile Link サーバの実行

Windows Mobile Link サーバをバックグラウンドで実行し、現在のセッションから独立して稼働させるには、Mobile Link サーバを「サービス」として起動します。

コマンド・ラインから、または Sybase Central の **[サービス]** タブで、以下のサービス管理タスクを実行できます。

- サービスの追加、編集、削除。
- サービスの開始と停止。
- サービスを制御するパラメータの変更。

参照

- 「Windows 用サービス・ユーティリティ (dbsvc)」 『SQL Anywhere サーバ - データベース管理』

サービスの追加、変更、削除

Sybase Central のサービス・アイコンは、サービスが実行中であるか停止されているかを示すアイコンを使用して、各サービスの現在の状況を表示します。

♦ **新しいサービスを追加するには、次の手順に従います (Sybase Central の場合)。**

1. Sybase Central の左ウィンドウ枠で [Mobile Link 11] をクリックします。
2. 右ウィンドウ枠で、**[サービス]** タブをクリックします。
3. 右ウィンドウ枠で右クリックし、**[新規]** - **[サービス]** を選択します。
4. **サービス作成ウィザード**の指示に従います。

dbsvc ユーティリティを使用してサービスを作成することもできます。「Windows 用サービス・ユーティリティ (dbsvc)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

♦ **サービスを削除するには、次の手順に従います (Sybase Central の場合)。**

- サービスを選択し、**[編集]** - **[削除]** をクリックします。

♦ **サービスのパラメータを変更するには、次の手順に従います。**

- サービスを右クリックし、**[プロパティ]** を選択します。

サービス設定の変更は、次回のサービス実行時から有効となります。

起動オプションの設定

次のオプションは、Mobile Link サービスの起動時の動作を制御します。これらのオプションは、**[サービスのプロパティ]** ウィンドウの **[一般]** タブで設定できます。

- **[自動]** **[自動]** を選択すると、サービスは Windows オペレーティング・システムが起動すると必ず起動します。この設定は、データベース・サーバやその他の常に稼働しているアプリケーションに適しています。
- **[手動]** **[手動]** を選択すると、サービスは Administrator 権限を持つユーザが起動したときにのみ起動します。Administrator 権限については、Windows のマニュアルを参照してください。

- **[無効]**　**[無効]** を選択すると、サービスは起動しません。

起動オプションは、次回 Windows を起動するときに適用されます。

コマンド・ライン・オプションの指定

[サービスのプロパティ] ウィンドウの **[設定]** タブには、実行プログラムのパスを入力する **[ファイル名]** テキスト・ボックスと、サービスのコマンド・ライン・オプションを入力する **[パラメータ]** テキスト・ボックスがあります。**[パラメータ]** ボックスには、実行プログラムの名前は入力しないでください。

たとえば、冗長ロギングを指定して Mobile Link 同期サービスを開始するには、[パラメータ] ボックスに次のように入力します。

```
-c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
-vc
```

サービスのコマンド・ライン・オプションは、実行プログラムのものと同じです。「Mobile Link サーバ・オプション」 43 ページを参照してください。

アカウント・オプションの設定

サービスが実行されるアカウントを選択できます。ほとんどのサービスは、特別なアカウントの LocalSystem で実行され、これがサービスのデフォルト・オプションになっています。**[サービスのプロパティ]** ウィンドウの **[アカウント]** タブを開き、アカウント情報を入力すると、別のアカウントでログオンするようにサービスを設定できます。

LocalSystem 以外のアカウントでサービスを実行するには、そのアカウントに「サービスとしてログオンする」権限が必要です。詳細なユーザ権限については、Microsoft Windows のマニュアルを参照してください。

サービスのアイコンがタスクバーまたはデスクトップに表示されるかどうかは、次のように、選択したアカウントと、**[デスクトップとの対話をサービスに許可]** チェックボックスが選択されているかどうかによって決まります。

- サービスが LocalSystem で実行されているときに、**[サービスのプロパティ]** ウィンドウの **[デスクトップとの対話をサービスに許可]** チェックボックスがオンになっている場合は、サービスを実行するコンピュータ上の Windows XP/200x にどのユーザがログインしてもデスクトップにアイコンが表示されます。すべてのユーザがアプリケーション・ウィンドウを開き、サービスとして実行されているプログラムを停止できます。
- サービスが LocalSystem で実行されているときに、**[サービスのプロパティ]** ウィンドウの **[デスクトップとの対話をサービスに許可]** チェックボックスがオフになっている場合は、ユーザのデスクトップにアイコンは表示されません。サービスの状態を変更する権限を与えられているユーザのみ、サービスを停止できます。
- サービスが他のアカウントで実行されている場合、デスクトップにアイコンは表示されません。サービスの状態を変更する権限を与えられているユーザのみ、サービスを停止できます。

実行ファイルの変更

Sybase Central で、サービスに関連付けられたプログラム実行ファイルを変更するには、**[サービスのプロパティ]** ウィンドウの **[設定]** タブをクリックし、**[ファイル名]** ボックスに新しいパスとファイル名を入力します。

実行ファイルを新しいディレクトリに移動する場合は、この内容も変更します。

開始と停止

♦ **サービスを開始または停止するには、次の手順に従います。**

1. Sybase Central の左ウィンドウ枠で [Mobile Link 11] をクリックし、右ウィンドウ枠で **[サービス]** タブを開きます。
2. サービスを右クリックし、**[起動]** または **[停止]** を選択します。

サービスを開始すると、停止するまで実行を続けます。Sybase Central を閉じたり、ログオフしてもサービスは停止しません。

サービスを停止すると、データベースへの接続はすべて閉じられ、データベース・サーバは停止されます。他のアプリケーションのプログラムは終了します。

一度に複数のサービスを実行する

Windows の [コントロール パネル] にあるサービス マネージャを使用していくつかのタスクを実行することはできますが、Windows のサービス マネージャで Mobile Link サービスをインストールしたり設定したりすることはできません。Mobile Link のすべてのサービス管理は、Sybase Central から行うことができます。

Windows の [コントロール パネル] から [サービス マネージャ] を開くと、サービスのリストが表示されます。SQL Anywhere サービスの名前は、サービスをインストールしたときに入力したサービス名に、SQL Anywhere がプレフィクスを付けたものです。インストールされたすべてのサービスが、リストに表示されます。

この項では、一度に複数のサービスを実行する方法について説明します。

サービスの依存性

状況によっては、複数の実行プログラムをサービスとして実行する必要があり、これらの実行プログラムが相互に依存する場合があります。たとえば、Mobile Link サーバを実行することと、同期するデータベース・サーバを実行することが必要な場合です。

このような場合には、サービスを正しい順序で開始する必要があります。Mobile Link 同期サービスは、統合データベース・サーバの起動が完了する前に起動されると、統合データベース・サーバを検出できないためエラーになります。Mobile Link サーバを起動するときにデータベース・サーバが実行されているような順序にしてください。(これは、統合データベース・サーバが別のコンピュータで実行されている場合は適用されません。)

サービス・グループを使用すると、これらの問題を防ぐことができます。サービス・グループは Sybase Central で管理します。

サービス・グループ

システムの各サービスを、サービス・グループの各メンバに割り当てることができます。デフォルトでは、各サービスはグループに属しています。Mobile Link サーバのデフォルトのグループは、SQLANYMobiLink です。

サービスが適切なグループのメンバであることをチェックしてから、サービスが正しい順序で開始するように設定してください。Sybase Central では、サービスが割り当てられているグループを調べ、このグループを変更できます。

♦ **サービスが割り当てられているグループを調べて変更するには、次の手順に従います。**

1. Sybase Central の左ウィンドウ枠で [Mobile Link 11] をクリックし、右ウィンドウ枠で **[サービス]** タブを開きます。
2. サービスを右クリックし、**[プロパティ]** を選択します。
3. **[依存性]** タブをクリックします。最上部のテキスト・ボックスにサービスが割り当てられているグループの名前が表示されます。
4. **[変更]** をクリックして、システムで使用可能なグループのリストを表示します。
5. いずれかのグループを選択するか、新しいグループの名前を入力します。
6. **[OK]** をクリックして、そのグループにサービスを割り当てます。

サービス依存性の管理

Sybase Central を使用して、サービスの依存性を指定できます。次に例を示します。

- 1 つ以上のグループを開始してから現在のサービスを開始するようにできます。
- 任意のサービスを開始してから現在のサービスを開始するようにできます。

♦ **サービスまたはグループを依存性のリストに追加するには、次の手順に従います。**

1. Sybase Central の左ウィンドウ枠で [Mobile Link 11] をクリックし、右ウィンドウ枠で **[サービス]** タブを開きます。
2. サービスを右クリックし、**[プロパティ]** を選択します。
3. **[依存性]** タブをクリックします。
4. **[サービスの追加]** または **[サービス・グループの追加]** をクリックして、サービスまたはグループを依存性リストに追加します。
5. リストからサービスまたはグループを 1 つ選択します。
6. **[OK]** をクリックして、そのサービスまたはグループを依存性リストに追加します。

Mobile Link サーバをサーバ・ファームで実行する

別途ライセンスが必要な必須コンポーネント
共有ステータス・モードは、Mobile Link の高可用性オプションの機能であり、別途ライセンスが必要です。「別途ライセンスが必要なコンポーネント」『SQL Anywhere 11 - 紹介』を参照してください。

Mobile Link サーバ・ファームは、リモート・データベースの同じセットを同じ統合データベースと同期する Mobile Link サーバが複数ある環境です。これは多くの場合、大規模な配備やフェールオーバ機能のために必要になります。このような Mobile Link サーバ・ファーム配備では、Mobile Link を共有ステータス・モードで実行することが必要になります。また、HTTP 通信リンクが使用される場合は、リレー・サーバの使用が必要になることがあります。TCP ベースのストリームでは、TCP 負荷分散装置が機能している必要があります。複数のサーバを使用している場合、再起動可能なダウンロードは機能しません。

Mobile Link は、デフォルトでは、共有サーバ・ステータスで実行されるわけではありません。

♦ 共有サーバ・ステータスを有効にするには

1. 各 Mobile Link サーバに、-zs コマンド・ライン・オプションを使用してユニークな名前を指定します。これらの名前は、統合データベースでファームのステータスを管理するために使用されます。「-zs オプション」 123 ページを参照してください。
2. -ss オプションを使用して、Mobile Link を共有サーバ・ステータス・モードで起動します。このオプションが設定されている場合、Mobile Link サーバは、起動時に「**このサーバは Mobile Link サーバ・ファームで実行されています**」というメッセージをログに出力します。「-ss オプション」 97 ページを参照してください。
3. サーバによって開始された Sync で Notifier を使用する場合は、-lsc オプションを使用してローカル・サーバ接続設定を指定します。これらの設定はファーム内の他のサーバに渡されるため、サーバが相互に接続して通知の処理を共有できます。たとえば、ホスト **farm_host22** で実行している場合は、次のようになります：mlsrv11 -x tcpip(port=3245) -zs -ss server5 -lsc tcpip(host=farm_host22;port=3245) -c ...

   Notifier を使用しない場合は、-lsc オプションを使用する必要ありません。

参照

- 「-lsc オプション」 74 ページ
- 「-zs オプション」 123 ページ
- 「-ss オプション」 97 ページ

Mobile Link サーバ起動時のトラブルシューティング

この項では、Mobile Link サーバの起動時に発生する一般的な問題について説明します。

ネットワーク通信ソフトウェアが実行されているかを確認する

適切なネットワーク通信ソフトウェアをインストールし、実行してから、Mobile Link サーバを実行します。信頼性の高いネットワーク・ソフトウェアを 1 つのネットワークだけが導入された状態で実行している場合は、問題はありません。Mobile Link サーバを実行する前に、ネットワーク通信を必要とする他のソフトウェアが正しく動作していること確認してください。

TCP/IP プロトコルを使用している場合は、ping と telnet が正しく動作していることを確認します。ping アプリケーションと telnet アプリケーションは、多数の TCP/IP プロトコル・スタックに付属します。

ネットワーク通信の起動時の問題をデバッグする

ネットワークで接続を確立するときに問題が生じた場合は、クライアントとサーバの両方でデバッグ・オプションを使用し、問題点を診断できます。サーバ・ウィンドウに起動情報が表示されます。-o オプションを使用して、出力ファイルに結果のログを取ります。

「Mobile Link サーバの動作のロギング」 33 ページを参照してください。

Mobile Link サーバ・オプション

目次

mlsrv11 の構文

機能

Mobile Link サーバを起動します。

構文

mlsrv11 -c "*connection-string***"** [*options*]

オプション	説明
@*data*	指定された環境変数または設定ファイルからオプションを読み込みます。「@data オプション」 51 ページを参照してください。
-a	同期エラー時の自動再接続を無効にします。「-a オプション」 52 ページを参照してください。
-b	文字列のブランクの埋め込みを削除します。「-b オプション」 53 ページを参照してください。
-bn *size*	競合の検出のため BLOB を比較するときに考慮する最大バイト数を指定します。「-bn オプション」 55 ページを参照してください。
-c *"keyword=value; ..."*	統合データベース用の ODBC データベース接続パラメータを指定します。「-c オプション」 56 ページを参照してください。
-cm *size*	サーバのメモリ・キャッシュのサイズを指定します。「-cm オプション」 57 ページを参照してください。
-cn *connections*	統合データベース・サーバとの最大同時接続数を設定します。「-cn オプション」 58 ページを参照してください。
-cr *count*	データベース接続リトライの最大回数を設定します。「-cr オプション」 59 ページを参照してください。
-cs *"keyword=value; ..."*	Mobile Link システム・データベース (MLSD) 用のシステム・データベース接続パラメータを指定します。
-ct *connection-timeout*	接続が使用されなくなってからタイムアウトになるまでの時間を設定します。「-ct オプション」 61 ページを参照してください。

オプション	説明
-dl	Mobile Link サーバのメッセージ・ウィンドウにすべてのログ・メッセージを表示します。「-dl オプション」 62 ページを参照してください。
-dr	Adaptive Server Enterprise の場合のみ。同期に関係するテーブルで、DataRow ロック・スキームを使用しないようにします。「-dr オプション」 63 ページを参照してください。
-ds *size*	すべての再起動可能なダウンロードで使用される保存できるデータの最大量を指定します。「-ds オプション」 64 ページを参照してください。
-dsd	SQL Anywhere と Microsoft SQL Server 統合データベースのデフォルトのダウンロード独立性レベルであるスナップショット・アイソレーションを無効にします。「-dsd オプション」 65 ページを参照してください。
-dt	現在のデータベース内のみでトランザクションを検出します。「-dt オプション」 66 ページを参照してください。
-e *filename*	送信されたリモート・エラー・ログを指定のファイルに格納します。「-e オプション」 67 ページを参照してください。
-esu	アップロードにスナップショット・アイソレーションを使用します。「-esu オプション」 68 ページを参照してください。
-et *filename*	指定のファイルが存在する場合は、最初に内容を削除してから、送信されたリモート・エラー・ログをそのファイルに格納します。「-et オプション」 69 ページを参照してください。
-f	同期スクリプトに変更がないものとみなします。「-f オプション」 70 ページを参照してください。
-fips	すべての Mobile Link セキュア・ストリームを強制的に FIPS 準拠にします。「-fips オプション」 71 ページを参照してください。
-fr	テーブル・データ・スクリプトがない場合は、同期を終了せず、警告の発行だけを行います。「-fr オプション」 72 ページを参照してください。

オプション	説明
-ftr *path*	mlfiletransfer ユーティリティによって使用されるファイル用のロケーションを作成します。「-ftr オプション」 73 ページを参照してください。
-lsc *protocol*[*protocol-options*	ローカル・サーバの接続情報を指定します。「-lsc オプション」 74 ページを参照してください。
-m [*filename*]	QAnywhere のメッセージ機能を有効にします。「-m オプション」 75 ページを参照してください。
-nba { + | - }	サーバ・ダウンロード確認の動作モードを設定します。「-nba オプション」 76 ページを参照してください。
-nc *connections*	同時接続の最大数を設定します。「-nc オプション」 77 ページを参照してください。
-notifier	サーバ起動同期用に Notifier を起動します。「-notifier オプション」 78 ページを参照してください。
-o *logfile*	ファイルにメッセージのログを取ります。「-o オプション」 79 ページを参照してください。
-on *size*	ログ・ファイルの最大サイズを設定します。「-on オプション」 80 ページを参照してください。
-oq	起動エラーの発生時にポップアップ・ウィンドウを表示しません。「-oq オプション」 81 ページを参照してください。
-os *size*	出力ファイルの最大サイズ。「-os オプション」 82 ページを参照してください。
-ot *logfile*	最初に内容を削除してから、ファイルにメッセージのログを取ります。「-ot オプション」 83 ページを参照してください。
-q	Mobile Link サーバのメッセージ・ウィンドウを最小化します。「-q オプション」 88 ページを参照してください。
-r *retries*	デッドロックされたアップロードを、この回数までリトライします。「-r オプション」 89 ページを参照してください。

オプション	説明
-rd *delay*	デッドロックされたトランザクションをリトライするまでの最大遅延秒数を設定します。「-rd オプション」 90 ページを参照してください。
-s *count*	一度にフェッチされ、送信されるローの最大数を指定します。「-s オプション」 91 ページを参照してください。
-sl dnet *script-options*	.NET CLR のオプションを設定し、起動時に仮想マシンを強制的にロードします。「-sl dnet オプション」 92 ページを参照してください。
-sl java *script-options*	Java 仮想マシンのオプションを設定し、起動時に仮想マシンを強制的にロードします。「-sl java オプション」 94 ページを参照してください。
-sm *number*	アクティブに動作できる同期の最大数を設定します。「-sm オプション」 96 ページを参照してください。
-ss	サーバを共有サーバ・ステータス・モードにします。「-ss オプション」 97 ページを参照してください。
-tc *minutes*	SQL スクリプト実行のカウント・ダウン・タイマを設定します。「-tc オプション」 98 ページを参照してください。
-tf	カウント・ダウン・タイマの期限が切れたら、SQL スクリプトを実行できないようにします (Oracle は対象外)。「-tf オプション」 99 ページを参照してください。
-tx *count*	アップロードのトランザクションで、トランザクションのグループをバッチにし、まとめてコミットします。「-tx オプション」 100 ページを参照してください。
-ud	UNIX プラットフォーム上でデーモンとして実行します。「-ud オプション」 101 ページを参照してください。
-ui	X-Window がサポートされている Linux で、使用可能な表示がない場合にシェル・モードで Mobile Link サーバを起動します。「-ui オプション」 102 ページを参照してください。

オプション	説明
-ux	Mobile Link サーバのメッセージ・ウィンドウを開きます。「-ux オプション」 103 ページを参照してください。
-v [*levels*]	ログ・ファイルに書き込まれるメッセージのタイプを制御します。「-v オプション」 104 ページを参照してください。
-w *count*	データベース・ワーカ・スレッド数を設定します。「-w オプション」 108 ページを参照してください。
-wu *count*	アップロード処理の同時実行を許可されるデータベース・ワーカ・スレッドの最大数を設定します。「-wu オプション」 109 ページを参照してください。
-x *protocol*[**(***network-parameters***)**]	通信プロトコルを指定します。必要に応じて、*parameter=value* の形式でネットワーク・パラメータを指定します。複数のパラメータを指定する場合はセミコロンで区切ります。「-x オプション」 110 ページを参照してください。
-xo *protocol*[**(***network-parameters***)**]	バージョン 8 と 9 のクライアントの場合は、通信プロトコルを指定します。必要に応じて、*parameter=value* の形式でネットワーク・パラメータを指定します。複数のパラメータを指定する場合はセミコロンで区切ります。「-xo オプション」 117 ページを参照してください。
-zp	統合データベースとリモート・データベース間にタイムスタンプの競合がある場合に、最小精度よりも高い精度を持つタイムスタンプ値を競合検出に使用します。「-zp オプション」 122 ページを参照してください。
-zs *name*	サーバ名を指定します。「-zs オプション」 123 ページを参照してください。
-zt *number*	Mobile Link サーバを実行するのに使用されるプロセッサの最大数を指定します。「-zt オプション」 124 ページを参照してください。
-zu { + | - }	authenticate_user スクリプトが未定義の場合に、ユーザの自動的な追加を制御します。「-zu オプション」 125 ページを参照してください。

オプション	説明
-zus	Mobile Link がアップロード内容のないテーブルのアップロード・スクリプトを呼び出すようにします。「-zus オプション」 126 ページを参照してください。
-zw 1,...5	表示する警告メッセージのレベルを制御します。「-zw オプション」 127 ページを参照してください。
-zwd *code*	特定の警告コードを無効にします。「-zwd オプション」 128 ページを参照してください。
-zwe *code*	特定の警告コードを有効にします。「-zwe オプション」 129 ページを参照してください。

説明

Mobile Link サーバは、統合データベース・サーバとの接続を ODBC を介して開きます。その後、クライアント・アプリケーションからの接続を受け入れ、同期処理を制御します。

-c オプションを使用して統合データベースの接続パラメータを指定してください。コマンド・ライン・オプションは、任意の順序で指定できます。ここでは、便宜上、-c オプションをコマンド文字列の最初の項目としています。接続文字列の前であれば、オプション・リストのどこにでも指定できます。

ODBC データ・ソースが統合データベースを自動的に起動するように設定していない場合には、統合データベースを実行してから Mobile Link サーバを起動してください。

参照

- 「Mobile Link サーバ」 29 ページ

@data オプション

指定された環境変数または設定ファイルからオプションを読み込みます。

構文

mlsrv11 -c "*connection-string***" @***data* ...

備考

このオプションを使用すると、指定された環境変数または設定ファイルから mlsrv11 コマンド・ライン・オプションを読み込むことができます。指定された名前と同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。

設定ファイルの詳細については、「設定ファイルの使用」『SQL Anywhere サーバ - データベース管理』を参照してください。

設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化できます。

「ファイル難読化ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』を参照してください。

-a オプション

Mobile Link サーバに対して同期エラーの発生時に再接続しないように指示します。

構文

mlsrv11 -c "*connection-string***" -a** ...

備考

同期中にエラーが発生すると、Mobile Link サーバは自動的に統合データベースとの接続を切断してから、接続を再確立します。再接続が行われると、認識されている状態から確実に次の同期が開始します。このような動作が不要な場合は、このオプションを使用して無効にしてください。ステータス情報の管理は、プログラマの要求や、DBMS を処理する Mobile Link スクリプトの設定方法に応じて異なります。このことは、Oracle や SQL Anywhere のデータベース、またはサポートされている他の製品にもあてはまります。クライアント・アプリケーションによっては、一部のステータス情報を初期化し直す操作が必要になることがあります。

-b オプション

VARCHAR、CHAR、LONG VARCHAR、または LONG CHAR 型のカラムの場合、同期中に文字列から後続ブランクを削除します。

構文

mlsrv11 -c "*connection-string***" -b** ...

備考

> **注意**
> この問題が発生しないように、統合データベースで CHAR の代わりに VARCHAR を使用することをおすすめします。

このオプションを使用すると、SQL Anywhere の CHAR データ型と統合データベースが使用する CHAR または VARCHAR データ型の違いを解決できます。SQL Anywhere の CHAR データ型は VARCHAR と同じです。しかし、SQL Anywhere ではないほとんどの統合データベースで、CHAR(n) データ型は n 文字までブランクが埋め込まれます。

-b が指定されていると、Mobile Link サーバは、リモートのカラムが文字列の場合に CHAR、VARCHAR、LONG CHAR、または LONG VARCHAR 型のカラムに対して文字列から後続ブランクを削除します。これを実行した後に、現在の同期でアップロードされたローをフィルタします。トリムされたデータは、次にリモート・データベースへダウンロードされます。

このオプションは、競合する更新を検出するためにも使用できます。各アップロード更新ローに対して、Mobile Link サーバは統合データベースから指定されたプライマリ・キーのローをフェッチし、そのローを更新前イメージと比較して、この更新が競合する更新であるかどうかを判断します。-b を使用する場合、Mobile Link は CHAR、VARCHAR、LONG CHAR、または LONG VARCHAR 型のカラムから後続ブランクをトリムしてから、この比較を行います。

参照

- 「CHAR カラム」 8 ページ
- 「NVARCHAR データ型」 『SQL Anywhere サーバ - SQL リファレンス』

例

-b オプションを使用しない場合、SQL Anywhere または Ultra Light リモートから統合データベースの CHAR(10) カラムにアップロードされるプライマリ・キー値 'abc' は、'abc' の後に 7 個のブランク・スペースが続く値になります。同じローがダウンロードされると、リモートでは 'abc' の後に 7 個のブランク・スペースが続く値として扱われます。リモート・データベースがブランクを埋め込まない場合、リモートは 2 つのローを格納します。それは、'abc' というローと 'abc' の後に 7 個のブランク・スペースが続くローです。これで、リモートで重複するローが存在することになります。

-b オプションを使用する場合、SQL Anywhere または Ultra Light リモートから統合データベースの CHAR(10) カラムにアップロードされるプライマリ・キー値 'abc' は、'abc' の後に 7 個のスペースが続く値になります。7 個のスペースが埋め込まれた値は 10 個の文字になりますが、同じローがダウンロードされるときに Mobile Link サーバは後続のスペースを削除するため、その値はリ

モートでは 'abc' として扱われます。このように、-b オプションは重複するローの問題を修正します。

-bn オプション

競合検出中に比較する最大 BLOB バイト数を設定します。

構文

mlsrv11 -c "*connection-string***" -bn** *size* ...

備考

2 つの BLOB が類似する値または同じ値で構成されている場合は、処理対象のデータの量が多くなるため、フィルタや競合検出などで両者の比較操作が高負荷になることがあります。このオプションを指定すると、Mobile Link サーバは、*size* で指定された最初のバイト数だけを比較対象とします。デフォルトでは、2 つの BLOB 全体を比較します。

比較するデータの最大量を制限することで、同期の速度をかなり上げることができますが、エラーが発生する可能性もあります。たとえば、2 つの大きな BLOB の最後の数バイトだけが異なる場合、Mobile Link サーバは、実際には異なるのに、同一であるとみなす可能性があります。

-c オプション

統合データベースの接続パラメータを指定します。

構文

mlsrv11 -c "*connection-string***"** ...

備考

接続文字列には、Mobile Link サーバが統合データベースに接続するために十分な情報を指定します。接続文字列は必須です。

接続文字列では、接続パラメータを *keyword=value* の形式で指定します。パラメータとパラメータの間はセミコロンで区切り、スペースは入れないでください。

接続パラメータをコマンド・ラインで指定しない場合は、ODBC データ・ソースの指定に含めてください。RDBMS と ODBC データ・ソースを調べて、必要な接続データを判断してください。

SQL Anywhere 接続パラメータの完全なリストについては、「接続パラメータ」『SQL Anywhere サーバ - データベース管理』を参照してください。

パスワードを非表示にする方法については、「ファイル難読化ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』を参照してください。

例

```
mlsrv11 -c "dsn=odbcname;uid=DBA;pwd=sql"
```


-cm オプション

サーバのメモリ・キャッシュの最大サイズを設定します。

構文

mlsrv11 -c "*connection-string***" -cm** *size*[**k** | **m** | **g**] ...

備考

テーブル・データ、ネットワーク・バッファ、キャッシュされたダウンロード・データ、同期に使用されるその他の構造体を保持するためにサーバが使用するメモリの最大容量を指定します。このメモリ・プールに保持できる量を超えるデータがサーバにある場合、データはディスクに保存されます。

size には、予約するメモリ容量をバイト単位で指定します。単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、**g** のいずれかを使用してください。デフォルトは **50M** です。

-cn オプション

統合データベースとの同時接続の最大数を設定します。

構文

mlsrv11 -c "*connection-string***" -cn** *value* ...

備考

Mobile Link サーバから統合データベースへの最大同時接続数を指定します。最小値 (デフォルト) は、データベース・ワーカ・スレッド数に 1 を加えた値です。指定した値が小さすぎると、警告が表示されます。

Mobile Link データベース接続は、1 つのスクリプト・バージョンを使用する同期にのみ使用されます。Mobile Link サーバが -cn オプションで許可されているすべてのデータベース接続を使用している場合、同期が保留中でも、そのスクリプト・バージョンのデータベース接続が現在存在しないときには、Mobile Link サーバは接続を切断してから、保留中の同期のスクリプト・バージョンに対して新しいデータベース接続を作成します。

データベース・ワーカ・スレッド数よりも大きい値を指定すると、統合データベースへの接続が低速の場合や、複数のスクリプトのバージョンが使用されている場合に、特にパフォーマンスが向上します。データベース接続の最適な最大数は、スクリプトのバージョン数とデータベース・ワーカ・スレッド数を乗算し、それに 1 を加えた値です。この最適値を上回る数の接続を使用しても、同期速度が上がるとはかぎりません。また、Mobile Link サーバと統合データベース・サーバの両方で、必要以上にリソースが消費されます。

参照

- 「-w オプション」 108 ページ

-cr オプション

データベース接続リトライの最大回数を設定します。

構文

mlsrv11 -c "*connection-string***" -cr** *value* ...

備考

接続不良のときに、Mobile Link サーバが終了前にデータベースへの接続を試行する最大回数を設定します。デフォルト値では、接続が 3 回リトライされます。

-cs オプション

Mobile Link システム・データベース (MLSD) 用の接続パラメータを指定します。

構文

mlsrv11 -c "*connection-string***" -cs "***connection-string***"** ...

備考

システム・テーブル、プロシージャ、トリガ、ビューなどの Mobile Link サーバ・システム・オブジェクトを、統合データベース以外のデータベースに格納できます。Mobile Link システム オブジェクトを格納するデータベースは、MLSD と呼ばれます。

コマンド・ラインでこのコマンド・オプションが指定されると、Mobile Link サーバは、ユーザ定義スクリプトをフェッチしたり、ML ユーザ名、リモート ID、進行状況オフセット、最終アップロード・タイムスタンプおよび最終ダウンロード・タイムスタンプなどの同期ステータスを維持したりするために、MLSD への接続を作成します。Mobile Link サーバは、元の -c コマンド・ライン・オプション接続を使用して統合データベースに接続し、これらの接続を使用してクライアント・データベースとのデータのアップロードやダウンロードを行います。統合データベースには、Mobile Link サーバ・システム・オブジェクトは必要ありません。エラー・レポート・スクリプトやエラー処理スクリプトを含むすべてのユーザ定義スクリプトは、MLSD からフェッチされ、統合データベースで実行されます。

このオプションが使用されると、Mobile Link サーバは、Microsoft 分散トランザクション・コーディネータ (MSDTC) を使用します。

統合データベースおよび MLSD には、サポートされている任意の Mobile Link 統合データベースを使用できます。ただし、対応する ODBC ドライバが Microsoft 分散トランザクションをサポートしている必要があります。

MSDTC を使用するには、統合データベースおよび MLSD にトランザクション・ログが必要です。

このオプションは、Windows オペレーティング・システムでしか使用できません。

-ct オプション

接続が使用されなくなってから、タイムアウトになって Mobile Link サーバによって切断されるまでの時間を分単位で設定します。

構文

mlsrv11 -c "*connection-string***" -ct** *connection-timeout ...*

備考

指定された時間にわたって使用されなかった Mobile Link データベース接続は、サーバによって解放されます。-ct オプションを使用してタイムアウトを設定できます。デフォルトのタイムアウト期間は 60 分です。

-dl オプション

画面に Mobile Link サーバのすべてのメッセージを表示します。

構文

mlsrv11 -c "*connection-string***" -v -dl** ...

備考

Mobile Link サーバのメッセージ・ウィンドウに Mobile Link サーバのすべてのメッセージを表示します。デフォルトでは、Mobile Link サーバのメッセージ・ログ・ファイルへの出力時には (-o を使用)、メッセージの一部だけがウィンドウに表示されます。多数のメッセージがある場合は、このオプションを指定するとパフォーマンスが低下することがあります。

参照

- 「-o オプション」 79 ページ
- 「データベース・サーバ・メッセージをファイルにロギングする」 『SQL Anywhere サーバ - データベース管理』

-dr オプション

Adaptive Server Enterprise の場合のみ。同期に関係するテーブルで、DataRow ロック・スキームを使用しないようにします。

構文

mlsrv11 -c "*connection-string***" -dr** ...

備考

このオプションは、DataRow ロック・スキーマを使用して作成した同期テーブルがない場合にのみ使用してください。

このオプションを使用すると、Mobile Link サーバによって送信される重複データを減らすことができます。

参照

- 「Mobile Link 独立性レベル」 170 ページ

-ds オプション

再起動可能なダウンロードで使用します。すべての再起動可能なダウンロードを保存するために Mobile Link サーバが使用できるデータの最大量を指定します。

構文

mlsrv11 -c "*connection-string***" -ds** *size*[**k** | **m** | **g**] ...

備考

Mobile Link サーバは、再起動可能なダウンロードで使用するために、クライアントに受信されなかったダウンロード・データを保持します。このオプションは、組み合わされたすべての同期に対してサーバが保持するデータの量を制限します。

size が小さ過ぎると、サーバがダウンロード・データを廃棄して、ダウンロードの再起動ができなくなる場合があります。サーバがダウンロード・データを破棄するのは、次のいずれかの場合です。

- ユーザがダウンロードを正常に完了した場合。
- 再開を有効にしていない状況で、ユーザが新しい同期要求で復帰した場合。
- 受信要求のキャッシュが必要な場合。正常にダウンロードできなかった最も古いデータが、最初にクリアされます。

単位をキロバイト、メガバイト、またはギガバイトで指定するには、それぞれ **k**、**m**、**g** のいずれかを使用してください。デフォルトは **10m** です。

参照

- 「失敗したダウンロードの再開」 163 ページ
- 「-dc オプション」 『Mobile Link - クライアント管理』

-dsd オプション

スナップショット・アイソレーションを無効にします。

構文

mlsrv11 -c "*connection-string***" -dsd** ...

備考

統合データベースが SQL Anywhere (バージョン 10 以降) または Microsoft SQL Server (2005 以降) の場合、ダウンロードのデフォルトの独立性レベルはスナップショット・アイソレーションです。統合データベースがこれらのデータベースのそれ以前のバージョンの場合には、デフォルトのダウンロード独立性レベルはコミットされた読み出しです。

スクリプトでデフォルトの独立性レベルを変更することもできます。しかし、SQL Anywhere バージョン 10 以降と Microsoft SQL Server 2005 以降のデータベースでは、独立性レベルはアップロードおよびダウンロード・トランザクションの開始時に設定されます。つまり、begin_connection スクリプトで独立性レベルを設定しても、begin_upload スクリプトと begin_download スクリプトで上書きされる場合があります。

このオプションは、SQL Anywhere バージョン 10 と Microsoft SQL Server 2005 の統合データベースのみに適用されます。

参照

- 「Mobile Link 独立性レベル」 170 ページ
- 「-dt オプション」 66 ページ
- 「-esu オプション」 68 ページ

-dt オプション

Microsoft SQL Server データベースおよび Adaptive Server Enterprise データベースの場合のみ。Mobile Link が現在のデータベース内のみでトランザクションを検出するようにします。

構文

mlsrv11 -c "*connection-string***" -dt** ...

備考

このオプションにより、Mobile Link は、現在のデータベース内のトランザクションを除く、すべてのトランザクションを無視します。これにより、スループットが向上し、ダウンロードされるローの重複が減少します。

次の場合に、このオプションを使用します。

- 統合データベースが、他のデータベースも実行している Microsoft SQL Server または Adaptive Server Enterprise の場合。
- Microsoft SQL Server との組み合わせでのアップロードまたはダウンロードにスナップショット・アイソレーションを使用している場合。
- テーブルを Adaptive Server Enterprise と同期するために DataRow ロック・スキームを使用している場合。
- アップロード・スクリプトまたはダウンロード・スクリプトが、サーバ上の他のデータベースにアクセスしない場合。

このオプションは、スナップショット・アイソレーションを使用している Microsoft SQL Server データベースと、同期に関係するテーブルに DataRow ロック・スキームを使用している Adaptive Server Enterprise データベースのみに適用されます。

参照

- 「Mobile Link 独立性レベル」 170 ページ
- 「-dsd オプション」 65 ページ
- 「-esu オプション」 68 ページ

-e オプション

SQL Anywhere Mobile Link クライアントから送信されたエラー・ログを格納します。

構文

mlsrv11 -c "*connection-string***" -e** *filename* ...

備考

-e オプションを指定しないと、SQL Anywhere Mobile Link クライアントからのエラー・ログは、ファイル *mlsrv11.mle* に格納されます。-e オプションは、Mobile Link サーバに対してエラー・ログを指定のファイルに格納するように指示します。デフォルトでは、リモート・サイトでエラーが発生すると、dbmlsync は Mobile Link サーバにリモート・ログ・メッセージを 32 キロバイトまで送信します。

このオプションは、同期の問題を診断するために、リモート・エラー・ログへの一元化されたアクセスを可能にします。

リモート・サイトから配信される情報量は、dbmlsync の拡張オプション ErrorLogSendLimit で制御できます。

参照

- 「-et オプション」 69 ページ
- 「ErrorLogSendLimit (el) 拡張オプション」 『Mobile Link - クライアント管理』

-esu オプション

アップロードにスナップショット・アイソレーションを使用します。

構文

mlsrv11 -c "*connection-string***" -esu** ...

備考

デフォルトでは、Mobile Link はアップロードに SQL_TXN_READ_COMMITTED 独立性レベルを使用します。ほとんどの場合、これが最適な独立性レベルです。

アップロードにスナップショット・アイソレーションを使用すると、アップロードの更新中にスナップショット・トランザクションで競合が発生する場合があります。競合が発生した場合、Mobile Link サーバはアップロード全体をロールバックして、アップロードのリトライを行います。この場合、Mobile Link サーバ・オプション -r または -rd の設定を調整して、リトライ間隔の遅延時間と最大リトライ回数を指定できます。

スクリプトでデフォルトの独立性レベルを変更することもできます。アップロードの独立性レベルを変更するには、通常、begin_upload スクリプトを使用します。

このオプションは、SQL Anywhere バージョン 10 と Microsoft SQL Server 2005 の統合データベースのみに適用されます。

参照

- 「Mobile Link 独立性レベル」 170 ページ
- 「-dsd オプション」 65 ページ
- 「-dt オプション」 66 ページ
- 「-r オプション」 89 ページ
- 「-rd オプション」 90 ページ

-et オプション

既存のファイルをトランケートした後で、SQL Anywhere Mobile Link クライアントから送信されたエラー・ログを指定のファイルに格納します。

構文

mlsrv11 -c "*connection-string***" -et** *filename ...*

備考

-et オプションは -e オプションと同じですが、エラー・ログ・ファイルがトランケートされてから、そのファイルに新しいエラーが追加されます。

リモート・サイトから配信される情報量は、dbmlsync の拡張オプション ErrorLogSendLimit で制御できます。

参照

- 「ErrorLogSendLimit (el) 拡張オプション」 『Mobile Link - クライアント管理』
- 「-e オプション」 67 ページ

-f オプション

同期スクリプトを一度だけロードして、パフォーマンスを向上させます。

構文

mlsrv11 -c "*connection-string***" -f** ...

備考

-f オプションを指定しないと、Mobile Link サーバは、通常の操作中に同期スクリプトが変更されたかどうかを調べます。このチェックは開発中には役立ちますが、運用環境ではパフォーマンスが低下することがあります。-f オプションを使用すると、Mobile Link サーバは、Mobile Link セッションごとに 1 回だけ同期スクリプトをロードします。

-fips オプション

すべての Mobile Link セキュア・ストリームを強制的に FIPS 準拠にします。

構文

mlsrv11 -c *connection-string*" **-fips** ...

備考

このオプションを指定すると、すべての Mobile Link サーバ暗号で FIPS 認定のアルゴリズムが使用されます。-fips オプションが指定されているときには、暗号化されていない接続は使用できますが、単純暗号化は使用できません。

このオプションを指定すると、FIPS 認定のアルゴリズムが指定されているかどうかに関係なく、FIPS 認定のアルゴリズムが接続に使用されます。たとえば、オプション -fips とオプション -x tls(...;fips=no;...) を使用して Mobile Link サーバを起動した場合、fips=no 設定は無視され、サーバは fips=yes として起動されます。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

Mobile Link トランスポート・レイヤ・セキュリティの場合、FIPS 認定でない RSA が指定されていても、-fips オプションにより、サーバは FIPS 認定の RSA 暗号を解く鍵を使用します。ECC を指定した場合、FIPS 認定の楕円曲線アルゴリズムは使用できないため、エラーが発生します。

参照

- 「Mobile Link クライアント／サーバ通信の暗号化」 『SQL Anywhere サーバ - データベース管理』
- 「FIPS 認定の暗号化テクノロジ」 『SQL Anywhere サーバ - データベース管理』

-fr オプション

必要なテーブル・データ・スクリプトがない場合は、同期をアボートせず、警告の発行だけを行います。

構文

mlsrv11 -c "*connection-string***" -fr** ...

備考

デフォルトでは、必要な同期スクリプトがない場合、Mobile Link サーバはアボートします。このオプションを使用すると、Mobile Link はアボートする代わりに警告を発行します。

参照

- 「必要なスクリプト」 344 ページ
- 「アップロード専用の同期とダウンロード専用の同期」 142 ページ
- 「同期イベント」 359 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

-ftr オプション

mlfiletransfer ユーティリティによって使用されるファイル用のロケーションを作成します。

構文

mlsrv11 -c "*connection-string***" -ftr** *path* ...

備考

このオプションはファイル転送ルート・ディレクトリを設定します。ユーザに転送するファイルは、ルート・ディレクトリに配置することも、ユーザ名を使用したサブディレクトリに配置することもできます。Mobile Link はまず、要求されたファイルを、接続したクライアントのユーザ名を使用した、ファイル転送ルート・ディレクトリのサブディレクトリで検索します。ファイルがサブディレクトリになかった場合、Mobile Link はファイル転送ルート・ディレクトリで検索します。

このオプションは、mlfiletransfer ユーティリティを使用する場合に必須です。

参照

- 「Mobile Link ファイル転送ユーティリティ (mlfiletransfer)」『Mobile Link - クライアント管理』
- 「authenticate_file_transfer 接続イベント」 372 ページ

-lsc オプション

ローカル・サーバの接続情報を指定します。この情報は、サーバ・ファームのその他のサーバに渡されます。

構文

mlsrv11 -c "*connection-string***" -lsc** *protocol*[*protocol-options*] ...

protocol : **tcpip** | **tls** | **http** | **https**

protocol-options : **(** *option*=*value*; ... **)**

備考

このオプションは、Mobile Link サーバ・ファームで Notifier を実行する場合にのみ必要です。この情報は、他のサーバがローカルの Mobile Link サーバに接続するときに、他のサーバに渡されます。

たとえば、server_rack10 というホストでサーバが実行されている場合、コマンド・ラインは次のように始めることができます。

```
mlsrv11  -x tcpip(port=200) -zs -ss server5 -lsc tcpip(host=server_rack10;port=200) -c ...
```

この例では、他のサーバは、統合データベースで共有ステータスを使用して接続文字列 tcpip(host=server_rack10;port=200) を取得し、その文字列を使用して、起動されたサーバに接続できます。

参照

- 「Mobile Link サーバをサーバ・ファームで実行する」 40 ページ
- 「-zs オプション」 123 ページ
- 「-ss オプション」 97 ページ
- 「Mobile Link サーバ・ファームでの Notifier」 『Mobile Link - サーバ起動同期』

-m オプション

QAnywhere のメッセージ機能を有効にします。

構文

mlsrv11 -c "*connection-string***" -m** [*message-properties-file*] ...

備考

この省略可能な *message-properties-file* は推奨されなくなりました。プロパティは、Sybase Central で指定してデータベースに格納されるか、サーバ管理要求で指定されます。

message-properties-file では、1 行に 1 つのプロパティを、プロパティ名、等号 (=)、プロパティ値の形式で指定します。

設定できるプロパティのリストについては、「サーバ・プロパティ」 『QAnywhere』を参照してください。

参照

- 「QAnywhere テクノロジの概要」 『QAnywhere』
- 「有効な Mobile Link での QAnywhere の起動」 『QAnywhere』
- 「サーバ管理要求」 『QAnywhere』

例

サンプルのサーバ・メッセージ・ストア (*samples-dir¥QAnywhere¥server¥qanyserv.db*) を使用して QAnywhere のメッセージングを開始するには、次のコマンドを実行します。

```
mlsrv11 -m -c "dsn=QAnywhere 10 Demo"
```

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。

-nba オプション

サーバ・ダウンロード確認の動作モードを設定します。

構文

mlsrv11 -c "*connection-string***" -nba{ + | - }** ...

備考

ダウンロード確認を指定する場合は、非ブロッキング (デフォルトであり、-nba+ によって設定) かブロッキング (-nba- によって設定) のどちらかのモードを選択できます。ブロッキング・ダウンロード確認 (-nba-) は推奨されなくなりました。可能な場合は常に非ブロッキングを使用してください。

ブロッキング・ダウンロード確認よりもパフォーマンスが大幅に優れている非ブロッキング・ダウンロード確認をおすすめします。ただし、次の場合は、非ブロッキング・ダウンロード確認を使用できません。

- 10.0.0 よりも前のクライアントは、非ブロッキング確認をサポートしていません。

mlsrv11 -m オプションを使用して QAnywhere メッセージングを有効にした場合は、ブロッキング・ダウンロード確認を使用できません。-m と -nba- の両方を指定することはできません。

参照

- dbmlsync：「SendDownloadACK (sa) 拡張オプション」『Mobile Link - クライアント管理』
- Ultra Light：「Send Download Acknowledgement 同期パラメータ」『Ultra Light データベース管理とリファレンス』
- 「nonblocking_download_ack 接続イベント」 495 ページ
- 「publication_nonblocking_download_ack 接続イベント」 500 ページ

-nc オプション

同時ネットワーク接続の最大数を設定します。

構文

mlsrv11 -c "*connection-string***" -nc** *connections* ...

備考

制限値に達すると、Mobile Link サーバは新しい同期接続を拒否します。クライアントでは、接続が拒否されたことを示すエラー・コードで通信エラーが発行されます。

デフォルトは 1024 です。

非永続 HTTP/HTTPS の同時同期の数を制限するには、-sm よりも大幅に大きい値を -nc に設定してください。-sm の制限に達すると、Mobile Link サーバは HTTP エラー 503 (Service Unavailable) をリモート・クライアントに返します。ただし -nc の制限に達すると、ソケット・エラーが発行されます。-nc と -sm の差が大きくなると、拒否された接続によって生成されるエラーが、わかりにくいソケット・エラーではなく HTTP 503 エラーになる可能性が高くなります。たとえば -sm を 100 に設定し、-nc を 1000 に設定してください。「-sm オプション」 96 ページを参照してください。

-notifier オプション

サーバによって開始される同期用に Notifier を起動します。

構文

mlsrv11 -c "*connection-string***" -notifier** [*notifier-properties-file*] ...

備考

Notifier の設定ファイル名を指定した場合、またはファイル名を指定していないが、デフォルトの Notifier プロパティ・ファイル (*config.notifier*) がある場合は、そのファイルを使用して Notifier が設定されます。このファイルは、統合データベースの ml_properties テーブルに格納されている設定情報よりも優先されます。

それ以外の場合は、統合データベースの ml_properties テーブルに格納されている設定情報が使用されます。

-notifier オプションを使用する場合、有効にしたすべての Notifier が起動されます。

Notifier の有効化の詳細については、「Notifier プロパティ」 『Mobile Link - サーバ起動同期』を参照してください。

参照

- 「サーバ起動同期の Mobile Link サーバ設定」 『Mobile Link - サーバ起動同期』
- 「Notifier 設定ファイルを使用したサーバ側設定の実行」 『Mobile Link - サーバ起動同期』
- 「Notifier」 『Mobile Link - サーバ起動同期』
- 「Mobile Link サーバ・ファームでの Notifier」 『Mobile Link - サーバ起動同期』

-o オプション

Mobile Link サーバのメッセージ・ログ・ファイルに出力メッセージのログを取り、Mobile Link サーバのメッセージ・ウィンドウに記録されるデータを制限します。

構文

mlsrv11 -c "*connection-string***" -o** *logfile ...*

備考

指定したファイルにすべてのログ・メッセージを書き込みます。Mobile Link サーバ・ウィンドウには (表示されている場合)、通常はログが取られたすべてのメッセージのうち、一部だけが表示されることに注意してください。

同期中にエラーが発生した場合、Mobile Link サーバは、この出力ファイルに完全なエラー・コンテキストを示します。エラー・コンテキストには次のような情報が含まれます。

- **ユーザ名**　同期中に Mobile Link SQL Anywhere アプリケーションによって提供される実際のユーザ名。
- **変更後のユーザ名**　modify_user スクリプトで変更されたユーザ名。
- **トランザクション**　エラーが発生したトランザクションのリスト。トランザクションには、authenticate_user、begin_synchronization、upload、prepare_for_download、download、または end_synchronization があります。
- **テーブル名**　テーブル名がある場合はテーブル名。テーブル名がない場合は NULL。
- **ローの操作**　操作には、INSERT、UPDATE、DELETE、または FETCH がある。
- **ロー・データ**　エラーが発生したローのすべてのカラム値。
- **スクリプト・バージョン**　現在同期に使用されているスクリプト・バージョン。
- **スクリプト**　エラーの原因となったスクリプト。

選択した冗長性のレベルに関係なく、ログにエラー・コンテキスト情報が表示されます。

参照

- 「-os オプション」 82 ページ
- 「-dl オプション」 62 ページ
- 「-ot オプション」 83 ページ
- 「-on オプション」 80 ページ
- 「-v オプション」 104 ページ

-on オプション

Mobile Link サーバのメッセージ・ログ・ファイルの最大サイズを指定します。ログ・ファイルがこのサイズに達すると、現在のファイルが拡張子 .old の付いた名前に変更され、新しいファイルが作成されます。

構文

mlsrv11 -c "*connection-string***" -on** *size* [**k** | **m**]...

備考

size には、メッセージ・ログの最大ファイル・サイズを、バイト単位で指定します。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小サイズは 10 KB です。

ログ・ファイルが指定されたサイズに達すると、Mobile Link サーバは出力ファイルを拡張子 .old の付いた名前に変更し、元の名前で新しいファイルを開始します。

注意
.old ファイルがすでに存在する場合は、そのファイルが上書きされます。古いログ・ファイルを削除しないようにするには、代わりに -os オプションを使用します。

このオプションは、-os オプションと同時に使用できません。

参照

- 「-o オプション」 79 ページ
- 「-ot オプション」 83 ページ
- 「-on オプション」 80 ページ
- 「-os オプション」 82 ページ
- 「-v オプション」 104 ページ

-oq オプション

Windows で、起動エラーの発生時にエラー・ウィンドウが表示されないようにします。

構文

mlsrv11 -c "*connection-string***" -oq** ...

備考

デフォルトでは、起動エラーが発生すると、Mobile Link サーバにウィンドウが表示されます。-oq オプションを指定すると、このウィンドウは表示されません。

-os オプション

Mobile Link サーバのメッセージ・ログ・ファイルの最大サイズを設定します。その後、新しい名前を持つ新しいログ・ファイルが作成されて使用されます。

構文

mlsrv11 -c "*connection-string***" -os** *size* [**k** | **m**] ...

備考

size には、出力メッセージのログを取るファイルの最大サイズを指定します。デフォルトの単位はバイトです。キロバイトまたはメガバイトの単位を指定するには、それぞれサフィックス k、m を使用します。最小サイズは 10 KB です。

Mobile Link サーバは現在のファイル・サイズを確認してから、ファイルに出力メッセージのログを取ります。ログ・メッセージによって、ファイル・サイズが指定したサイズより大きくなると、Mobile Link サーバはメッセージ・ログ・ファイルの名前を *yymmddxx.mls* に変更します。*xx* は 00 ～ 99 の数字で、*yymmdd* は現在の年月日を表します。

このオプションを使用すると、古いメッセージ・ログ・ファイルを削除して、ディスク領域を解放できます。常に最新の出力が -o または -ot で指定したファイルに追加されます。

このオプションは、-on オプションと一緒に使用することはできません。

注意
このオプションを使用すると、ログ・ファイルが無制限に作成されます。この状況を回避するには、-o または -on を使用します。

参照

- 「-o オプション」 79 ページ
- 「-on オプション」 80 ページ
- 「-ot オプション」 83 ページ
- 「-v オプション」 104 ページ

-ot オプション

最初に Mobile Link サーバのメッセージ・ログ・ファイルの内容を削除してから、そのファイルに出力メッセージのログを取ります。

構文

mlsrv11 -c "*connection-string***" -ot** *logfilename ...*

備考

デフォルトでは、画面に出力を送信します。

参照

- 「-on オプション」 80 ページ
- 「-os オプション」 82 ページ
- 「-v オプション」 104 ページ
- 「-o オプション」 79 ページ

-ppv オプション

指定された期間に従って、Mobile Link が新しい定期モニタリングの値を出力するようにします。期間は秒数で指定します。

構文

mlsrv11 -c "*connection-string***" -ppv** *period ...*

備考

これらの値は、サーバの状態を把握するために使用でき、Mobile Link サーバの正常性やパフォーマンスを判断するときに便利です。たとえば、DB_CONNECTIONS と LONGEST_DB_WAIT の値を確認して、-w オプションや同期スクリプトに存在する潜在的な問題を調べることができます。これらの値を使用して、1 秒間にアップロードまたはダウンロードされたロー数や、1 秒間に成功した同期数など、システム全体のスループット測定を簡単に追跡することもできます。

期間の推奨値は 60 秒です。

設定した期間の値が小さすぎると、ログが急速に増大します。

出力の各ローは、値の検索やフィルタが容易になるように、**PERIODIC:** で始まります。

出力される値には、次の情報が含まれます。

- **CMD_PROCESSOR_STAGE_LEN**　同期タスクのキューの長さ。
- **CPU_USAGE**　Mobile Link サーバによって使用される CPU 時間の量 (マイクロ秒)。
- **DB_CONNECTIONS**　使用中のデータベース接続数。
- **FREE_DISK_SPACE**　テンポラリ・ディスクの空きディスク領域 (バイト数)。
- **HEARTBEAT_STAGE_LEN**　同期以外の定期的なタスクのキューの長さ。
- **LONGEST_DB_WAIT**　アクティブな同期でデータベースを待機していた最長時間。
- **LONGEST_SYNC**　最も古いアクティブな同期の経過時間 (マイクロ秒)。
- **MEMORY_USED**　使用中の RAM のバイト数 (Windows のみ)。
- **ML_NUM_CONNECTED_CLIENTS**　接続された同期クライアント数。
- **NUM_COMMITS**　コミットの合計数。
- **NUM_CONNECTED_FILE_XFERS**　現在接続されている mlfiletransfer 数。
- **NUM_CONNECTED_LISTENERS**　現在接続されている Listener 数。
- **NUM_CONNECTED_MONITORS**　現在接続されているモニタ数。
- **NUM_CONNECTED_PINGS**　現在接続されている、ping を実行するクライアント数。
- **NUM_CONNECTED_SYNCS**　現在接続されているデータ同期数。
- **NUM_ERRORS**　エラーの合計数。
- **NUM_FAILED_SYNCS**　失敗した同期の合計数。

- **NUM_IN_APPLY_UPLOAD**　現在、アップロードの適用フェーズにある同期の数。
- **NUM_IN_AUTH_USER**　現在、ユーザ認証フェーズにある同期の数。
- **NUM_IN_BEGIN_SYNC**　現在、同期開始フェーズにある同期の数。
- **NUM_IN_CONNECT**　現在、接続フェーズにある同期の数。
- **NUM_IN_CONNECT_FOR_ACK**　現在、ダウンロード確認の接続フェーズにある同期の数。
- **NUM_IN_END_SYNC**　現在、同期終了フェーズにある同期の数。
- **NUM_IN_FETCH_DNLD**　現在、ダウンロードのフェッチ・フェーズにある同期の数。
- **NUM_IN_GET_DB_WORKER_FOR_ACK**　現在、確認の DB ワーカの取得フェーズにある同期の数。
- **NUM_IN_NON_BLOCKING_ACK**　現在、非ブロッキング・ダウンロード確認フェーズにある同期の数。
- **NUM_IN_PREP_FOR_DNLD**　現在、ダウンロードの準備フェーズにある同期の数。
- **NUM_IN_RECVING_UPLOAD**　現在、アップロードの受信フェーズにある同期の数。
- **NUM_IN_SEND_DNLD**　現在、ダウンロードの送信フェーズにある同期の数。
- **NUM_IN_SYNC_REQUEST**　現在、同期要求フェーズにある同期の数。
- **NUM_IN_WAIT_FOR_DNLD_ACK**　現在、ダウンロード確認の待機フェーズにある同期の数。
- **NUM_ROLLBACKS**　ロールバックの合計数。
- **NUM_ROWS_DOWNLOADED**　リモートに送信されたローの合計数。
- **NUM_ROWS_UPLOADED**　リモートから受信したローの合計数。
- **NUM_SUCCESS_SYNCS**　成功した同期の合計数。
- **NUM_UNSUBMITTED_ERROR_RPTS**　未送信のエラー・レポートの数。
- **NUM_UPLOAD_CONNS_IN_USE**　現在使用中のアップロード接続の数。
- **NUM_WAITING_CONS**　統合データベースを現在待機している同期の数。
- **NUM_WARNINGS**　警告の合計数。
- **PAGES_IN_STREAMSTACK**　ネットワーク・ストリームで保持されているページの数。
- **PAGES_LOCKED**　メモリにロードされているキャッシュ・ページ数。
- **PAGES_LOCKED_MAX**　メモリ・キャッシュ内のページの数。-cm を使用して設定されます。「-cm オプション」 57 ページを参照してください。
- **PAGES_SWAPPED_IN**　ディスクから読み込まれたページの合計数。
- **PAGES_SWAPPED_OUT**　ディスクへスワップされたページの合計数。
- **PAGES_USED**　使用されたキャッシュ・ページ数。ディスクにスワップされたページが含まれるため、キャッシュ・サイズよりも大きくなることがあります。
- **RAW_TCP_STAGE_LEN**　ネットワーク・ワーク・キューの長さ。

- **SERVER_IS_PRIMARY**　サーバがプライマリかセカンダリかを示します。サーバがプライマリの場合は 1、それ以外の場合は 0 を示します。
- **STREAM_STAGE_LEN**　高レベルのネットワーク処理キューの長さ。
- **TCP_BYTES_READ**　これまでに読み込まれたバイトの合計数。
- **TCP_BYTES_WRITTEN**　これまでに書き込まれたバイトの合計数。
- **TCP_CONNECTIONS**　現在開いている TCP 接続の数。
- **TCP_CONNECTIONS_CLOSED**　これまでに閉じられた接続の合計数。
- **TCP_CONNECTIONS_OPENED**　これまでに開かれた接続の合計数。
- **TCP_CONNECTIONS_REJECTED**　これまでに拒否された接続の合計数。

例

次は、定期モニタリングの値を示す出力の例です。

```
I. 2008-10-14 10:34:43. <Main> PERIODIC: TCP_CONNECTIONS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: PAGES_USED: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: PAGES_LOCKED: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: PAGES_LOCKED_MAX: 12692
I. 2008-10-14 10:34:43. <Main> PERIODIC: TCP_CONNECTIONS_OPENED: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: TCP_CONNECTIONS_CLOSED: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: TCP_CONNECTIONS_REJECTED: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: TCP_BYTES_READ: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: TCP_BYTES_WRITTEN: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: ML_NUM_CONNECTED_CLIENTS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: PAGES_SWAPPED_OUT: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: PAGES_SWAPPED_IN: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: PAGES_IN_STREAMSTACK: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: CPU_USAGE: 468750
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_COMMITS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_ROLLBACKS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_SUCCESS_SYNCS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_FAILED_SYNCS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_ERRORS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_WARNINGS: 1
I. 2008-10-14 10:34:43. <Main> PERIODIC: DB_CONNECTIONS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: RAW_TCP_STAGE_LEN: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: STREAM_STAGE_LEN: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: HEARTBEAT_STAGE_LEN: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: CMD_PROCESSOR_STAGE_LEN: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_ROWS_DOWNLOADED: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_ROWS_UPLOADED: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: FREE_DISK_SPACE: 162552295424
I. 2008-10-14 10:34:43. <Main> PERIODIC: LONGEST_DB_WAIT: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: LONGEST_SYNC: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_UNSUBMITTED_ERROR_RPTS: 247
I. 2008-10-14 10:34:43. <Main> PERIODIC: MEMORY_USED: 94375936
I. 2008-10-14 10:34:43. <Main> PERIODIC: SERVER_IS_PRIMARY: 1
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_CONNECTED_SYNCS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_CONNECTED_PINGS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_CONNECTED_FILE_XFERS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_CONNECTED_MONITORS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_CONNECTED_LISTENERS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_WAITING_CONS: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_SYNC_REQUEST: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_RECVING_UPLOAD: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_CONNECT: 0
```

```
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_AUTH_USER: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_BEGIN_SYNC: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_APPLY_UPLOAD: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_PREP_FOR_DNLD: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_FETCH_DNLD: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_END_SYNC: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_SEND_DNLD: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_WAIT_FOR_DNLD_ACK: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_GET_DB_WORKER_FOR_ACK: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_CONNECT_FOR_ACK: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_IN_NON_BLOCKING_ACK: 0
I. 2008-10-14 10:34:43. <Main> PERIODIC: NUM_UPLOAD_CONNS_IN_USE: 0
```

-q オプション

Mobile Link を起動時に最小化ウィンドウで実行するように指示します。

構文

mlsrv11 -c "*connection-string***" -q** ...

備考

Mobile Link サーバのウィンドウを最小化します。

-r オプション

デッドロックの最大リトライ回数を設定します。

構文

mlsrv11 -c "*connection-string***" -r** *retries* ...

備考

デフォルトでは、Mobile Link サーバはデッドロックされたアップロードを最大 10 回リトライします。デッドロックは解消できる保証がないため、デッドロックが解除されない場合は同期が失敗します。このオプションで、リトライの制限回数を任意に設定できます。サーバによるデッドロック・トランザクションのリトライを停止するには、**-r 0** を指定します。この設定の上限は、2 の 32 乗から 1 を引いた値です。

-rd オプション

デッドロック・リトライ間の最大遅延時間を設定します。

構文

mlsrv11 -c "*connection-string***" -rd** *delay ...*

備考

アップロード・トランザクションがデッドロックされると、Mobile Link サーバは不特定の時間待機してから、そのトランザクションをリトライします。遅延時間がランダムなため、その後の試行が成功する可能性が高くなります。このオプションを使用すると、最大遅延時間を秒単位で指定できます。値を 0 にするとリトライが即座に行われますが、より大きな値を指定した方がリトライの成功率が高くなります。デフォルトの最大遅延値は **30** です。

-s オプション

一度にアップロードできるローの最大数を設定します。

構文

mlsrv11 -c "*connection-string***" -s** *count* ...

備考

一度に挿入、更新、または削除できるローの最大数を *count* に設定します。

Mobile Link サーバは、ODBC ドライバを使用してアップロード・ローを統合データベースに送信します。このオプションは、各バッチでデータベース・サーバに送信されるローの数を制御します。この値を増やすと、アップロード・ストリームの処理速度を向上させ、ネットワーク時間を短縮できます。しかし、設定を高くすると、Mobile Link サーバでは、アップロード・ストリームに適用するリソースをより多く必要とする可能性があります。

一度にアップロードされるローの数は、ログ・ファイルで **rowset size** として参照できます。

デフォルトは 10 です。

-sl dnet オプション

.NET Common Language Runtime (CLR) オプションを設定し、起動時に CLR を強制的にロードします。

構文

mlsrv11 -c "*connection-string***" -sl dnet** *options ...*

備考

.NET CLR に直接渡すオプションを設定します。オプションは次のとおりです。

オプション	説明
-D*name=value*	環境変数を設定します。次に例を示します。 -Dsynchtype=far -Dextra_rows=yes 詳細については、.NET フレームワークのクラス System.Environment を参照してください。
-MLAutoLoadPath=*path*	基本アセンブリのロケーションを設定します。プライベート・アセンブリでのみ有効です。アセンブリのロケーションを Mobile Link に指示するには、このオプションまたは -MLDomConfigFile を使用します。両方同時には使用できません。-MLAutoLoadPath を使用する場合、イベント・スクリプトでドメインを指定できません。デフォルトは現在のディレクトリです。
-MLDomConfigFile=*file*	基本アセンブリのロケーションを設定します。共有アセンブリがある場合、すべてのアセンブリをディレクトリにロードしない場合、またはその他の理由で MLAutoLoadPath を使用できない場合に使用します。アセンブリのロケーションを Mobile Link に指示するには、-MLDomConfigFile または -MLAutoLoadPath を使用します。両方同時には使用できません。
-MLStartClasses= *classnames*	サーバの起動時に、ユーザ定義の起動クラスをリスト内の順序でロードしインスタンス化します。
-clrConGC	CLR での同時ガーベジ・コレクションを有効にします。
-clrFlavor=(wks | svr)	ロードする .NET CLR の種類。サーバは **svr** で、ワークステーションは **wks** です。デフォルトでは、**wks** がロードされます。

オプション	説明
-clrVersion=*version*	ロードする .NET CLR のバージョン。プレフィクス **v** が必要です。たとえば、**v1.0.3705** と指定すると、ディレクトリ *¥Microsoft.NET¥Framework¥v1.0.3705* がロードされます。

このオプション・リストを表示するには、次のコマンドを実行します。

```
mlsrv11 -sl dnet (?)
```

参照

- 「.NET での同期スクリプトの作成」 617 ページ

-sl java オプション

Java 仮想マシンのオプションを設定し、起動時に仮想マシンを強制的にロードします。

構文

mlsrv11 -c "*connection-string***" -sl java (*options*)** ...

備考

Java 仮想マシンに直接渡す -jrepath とその他のオプションを設定します。オプションは次のとおりです。

オプション	説明
-hotspot | **-server** | **-classic**	使用する Java VM のデフォルトのオプションを上書きします。
-cp *location*;...	クラスの検索先となる一連のディレクトリまたは JAR ファイルを指定します。-cp の代わりに -classpath も使用できます。
-D*name=value*	システム・プロパティを設定します。次に例を示します。 -Dsynchtype=far -Dextra_rows=yes
-DMLStartClasses=*classname*, ...	サーバの起動時に、ユーザ定義の起動クラスをリスト内の順序でロードしインスタンス化します。
-jrepath *path*	デフォルトの JRE パス (*install-dir¥Sun¥jre160_chip* ディレクトリ) を上書きします。chip は、x86 などのサポートされる任意のチップの値です。
-verbose [**:class** |**:gc** | **:jni**]	冗長出力を有効にします。
-X *vm-option*	ファイル *install-dir¥Sun¥jre160_chip¥bin¥client ¥Xusage.txt* での記述に従って VM 固有のオプションを設定します。chip は、x86 などのサポートされる任意のチップの値です。

使用できる Java オプションのリストを表示するには、次のように入力します。

```
java
```

UNIX に関する注意事項

オプションはカッコで囲んでください。これは、丸カッコでも、上記の構文に示すように中カッコ { } でもかまいません。

-jrepath オプションは、Windows でのみ使用できます。UNIX で特定の JRE をロードする場合は、LD_LIBRARY_PATH (AIX の場合は LIBPATH、HP-UX の場合は SHLIB_PATH) を設定して、JRE

を含むディレクトリを指定します。ディレクトリは、すべての SQL Anywhere インストール・ディレクトリの前に指定します。

UNIX では、-cp オプションをコロンで区切る必要があります。

参照

- 「Java による同期スクリプトの作成」 553 ページ

例

たとえば、Windows では、次の mlsrv11 コマンド・ラインの一部では、システム・アサートを有効にする Java 仮想マシン・オプションを設定します。

```
mlsrv11 -sl java (-cp ;¥myclasses; -esa) ...
```

Windows では、次の mlsrv11 コマンド・ラインの一部は、LDAP_SERVER システム・プロパティを定義します。

```
mlsrv11 -sl java ( -cp ;¥myclasses; -DLDAP_SERVER=huron-ldap ) ...
```

UNIX では、次の mlsrv11 コマンド・ラインの一部を使用できます。

```
mlsrv11 -sl java { -cp .:$CLASSPATH:/opt/myclasses:/opt/my.jar: }
```


-sm オプション

アクティブに動作できる同期の最大数を設定します。これによって、ネットワーク接続の最大数も制限されます。

構文

mlsrv11 -c "*connection-string***" -sm** *number ...*

備考

Mobile Link サーバは次のタスクを同時に実行します。

1. ネットワークからアップロード・データを読み込んでアンパックする。
2. アップロードを統合データベースに適用する。
3. 統合データベースからダウンロードするローをフェッチする。
4. ダウンロード・データをパックして、リモート・データベースに送信する。

各タスクの同期の数は次のように制限されます。

- タスク 2 と 3 を実行する同期の数は、mlsrv11 -w オプションの設定以下
- タスク 2 を実行する同期の数は、mlsrv11 -wu オプションの設定以下
- 4 つのタスクすべてを実行する同期の数は、-sm オプションの設定以下

-sm の値を高くすると、特に -w より高く設定した場合に、Mobile Link サーバはネットワーク・タスク (1 と 4) をデータベース・タスク (2 と 3) より多く処理できます。このようにすると、そのままではネットワーク・パフォーマンスがボトルネックになるような状況で、データベース・ワーカがタスクを待つ必要がなくなります。これによりスループットが向上します。しかし、-sm の設定が高すぎると、Mobile Link サーバは、直接使用可能な量を超えるメモリを割り付けできるようになります。その結果、オペレーティング・システムの仮想メモリ・ページングがアクティブになり、メモリはデスクにスワップされて、スループットが極端に低下します。

参照

- 「-w オプション」 108 ページ
- 「-wu オプション」 109 ページ

-ss オプション

別途ライセンスが必要な必須コンポーネント
-ss オプションは、Mobile Link の高可用性オプションの機能であり、別途ライセンスが必要です。「別途ライセンスが必要なコンポーネント」『SQL Anywhere 11 - 紹介』を参照してください。

Mobile Link サーバをサーバ・ファームで実行できます。

構文

mlsrv11 -c "*connection string***" -ss** ...

備考

デフォルトでは、Mobile Link サーバはサーバ・ファームでは実行されません。現在の Mobile Link サーバをサーバ・ファームで実行できるようにする場合は、このオプションを指定してください。Mobile Link サーバは、-ss で起動されると、それ自体を統合データベースの ml_server テーブルに追加し、同じ統合データベースに接続されていても -ss で起動されていない Mobile Link サーバがあれば、そのサーバを停止します。「Mobile Link サーバをサーバ・ファームで実行する」 40 ページを参照してください。

このオプションが指定される場合、サーバは -zs オプションを使用して指定される名前を持つことが必要です。たとえば、サーバのコマンド・ラインは次のようにすることができます。

mlsrv11 -c "*connection-string***" -ss -zs** *server5*

参照

- 「Mobile Link サーバをサーバ・ファームで実行する」 40 ページ
- 「-zs オプション」 123 ページ
- 「-lsc オプション」 74 ページ

-tc オプション

SQL スクリプトの実行において、実行時間の長い文のスレッショルドを設定します。

構文

mlsrv11 -c "*connection string***" -tc** *minutes ...*

備考

デフォルトでは、Mobile Link サーバは、各 SQL スクリプトの実行時間を監視し、スクリプトの実行時間が 10 分間 (デフォルト) に達すると警告メッセージを発行します。-tf オプションが使用され、統合データベースが Oracle サーバで実行されている場合、Mobile Link サーバはスクリプトを失敗にします。

デフォルト値は 0 にリセットすることも、正の整数にリセットすることもでき、単位は分です。値が 0 に設定されると、-tc スイッチが無効になり、Mobile Link サーバはスクリプトの実行を監視しません。

警告時間が 0 以外の値の場合、Mobile Link サーバは指数関数的な方法で警告メッセージを表示します。警告は、実行時間が指定された時間を初めて過ぎたときに表示されます。また、指定された時間の 2 倍、次に 4 倍、というような時間を過ぎたときにも表示されます。

警告メッセージには、現在の同期に使用されている接続 ID と、タイムアウト警告コンテキストが含まれます。タイムアウト警告コンテキストには、リモート ID、ML ユーザ名、変更後のユーザ名、トランザクション、テーブル名、ロー値、スクリプト・バージョンがあれば表示されます。タイムアウト警告コンテキストは、Mobile Link サーバの冗長設定に関係なく表示されます。

統合データベースが Oracle データベース・サーバで実行されていて、タイムアウト警告メッセージが表示されたときは、DBA 権限を持つデータベース・ユーザが統合データベースをチェックして問題の原因を特定することが必要な場合があります。警告メッセージから、同期によって使用された接続の SID および SERIAL# を知ることができます。同期接続が停止されると、Mobile Link サーバは現在の同期を終了します。

参照

- 「-tf オプション」 99 ページ

-tf オプション

このオプションは、実行時間が -tc によって指定された時間を過ぎた場合に、Mobile Link サーバが SQL スクリプトを失敗させる場合に使用します。このオプションは、統合データベースが Oracle サーバで実行されている場合には使用できません。

構文

mlsrv11 -c "*connection string***" -tf** ...

備考

SQL スクリプトが失敗した場合、Mobile Link サーバは、ローをスキップし (スクリプトがアップロード・スクリプトの場合や、handle_error スクリプトが 1000 を返した場合)、同期を継続するか、同期をアボートします。

このオプションが指定され、Mobile Link サーバが Oracle サーバに対して実行されている場合、Mobile Link サーバは、警告メッセージを表示します。

-tc 0 が指定されている場合、このオプションは無視されます。

-tx オプション

アップロードのトランザクションを使用している場合、このオプションは、トランザクションのグループをバッチにし、まとめてコミットします。

構文

mlsrv11 -c "*connection-string***" -tx** *count* ...

備考

このオプションを使用すると、アップロードのトランザクションを行うときのパフォーマンスが向上します。

count は、負以外の任意の値にすることができます。デフォルトは 1 であり、これは各トランザクションを別々にコミットすることを意味します。値 0 を使用すると、すべてのトランザクションがアップロードされてからコミットが 1 回実行されます。

参照

- 「-tu オプション」 『Mobile Link - クライアント管理』

-ud オプション

Mobile Link をデーモンとして実行するように指示します。

構文

mlsrv11 -c "*connection-string***" -ud** ...

備考

UNIX プラットフォーム専用です。

参照

- 「現在のセッション外での Mobile Link サーバの起動」 35 ページ

-ui オプション

X Window サーバがサポートされている Linux で、使用可能な表示がない場合にシェル・モードで Mobile Link サーバを起動します。

構文

mlsrv11 -c "*connection-string***" -ui** ...

備考

このオプションを使用すると、X-Window で mlsrv11 の起動が試みられます。それが失敗した場合は、シェル・モードで起動されます。

-ui を指定すると、サーバは使用可能な表示を探そうとします。X-Window Server が実行されていなかったなどの理由で、使用可能な表示が見つからなかった場合は、Mobile Link サーバはシェル・モードで起動されます。

-ux オプション

Linux の場合に、メッセージを表示する、Mobile Link サーバのメッセージ・ウィンドウを開きます。

構文

mlsrv11 -c "*connection-string* **-ux** ...

備考

-ux が指定されている場合、Mobile Link サーバは使用可能な表示を見つけます。たとえば、DISPLAY 環境変数が設定されていなかったり、X-Window Server が実行されていなかったりしたために、使用可能な表示が見つからなかった場合、Mobile Link サーバは起動できません。

クワイエット・モードで Mobile Link サーバのメッセージ・ウィンドウを実行するには、-q を使用します。

Windows では、Mobile Link サーバのメッセージ・ウィンドウが自動的に表示されます。

参照

- 「-q オプション」 88 ページ

-v オプション

メッセージ・ログ・ファイルにログを取り、同期ウィンドウに表示する情報を指定できます。

構文

mlsrv11 -c "*connection-string***" -v[** *levels* **]** ...

備考

このオプションは、dbmlsync のトランザクション・レベルのアップロードを使用している場合に特に便利です。

このオプションは、メッセージ・ログ・ファイルに書き込まれるメッセージのタイプを制御します。

-v を単独で指定すると、Mobile Link サーバは、各同期について最小量の情報を書き込みます。

冗長レベルを上げ過ぎるとパフォーマンスに影響する可能性があるので、冗長レベルを上げるのは開発中だけにしてください。

levels の値は次のとおりです。たとえば -vnrsu など、一度に 1 つまたは複数のオプションを使用できます。

- **+** 冗長性を高めるすべてのログ・オプションを on にします。
- **c** 呼び出された各同期スクリプトの内容を表示します。このレベルには、s の機能が含まれます。
- **e** システム・イベント・スクリプトを表示します。システム・イベント・スクリプトを使用して、Mobile Link システム・テーブルと、アップロードを制御する SQL スクリプトを管理します。
- **f** 最初の読み込みエラーを表示します。このオプションは、負荷分散装置がサーバの活性を確認するとき、データを送信しない接続を確立して同期が失敗したために発生したエラーのログを取ります。

  TCP/IP 接続の場合は、TCP/IP の **ignore** オプションを使用する方がよいでしょう。詳細については、「-x オプション」 110 ページを参照してください。
- **h** 同期中にアップロードされたリモート・スキーマを表示します。
- **i** アップロードされた各ローのカラム値を表示します。アップロードとダウンロードされた各ローのカラム値を表示する -vr オプションの代わりにこのオプションを使用すると、ログに記録されるデータ量が減少します。-vi と -vq を同時に指定することは、-vr を指定することと同じです。
- **m** 同期が完了するたびに、各同期と各同期フェーズの継続時間をログに出力します。同期フェーズは以下に示します。これらは、Mobile Link モニタに表示されるフェーズと同じです。すべての時間はミリ秒 (ms) で表されます。
  - **[同期要求]** リモート・データベースと Mobile Link サーバの間のネットワーク接続を確立してから、アップロード・ストリームの最初のバイトを受信するまでの時間。-sm を -nc よりも小さい値に設定した場合以外はこの時間はわずかです。-sm を -nc よりも小さい

値に設定した場合は、-sm で指定したアクティブな同期の最大数よりも同期の数が多かった場合に同期が一時停止した時間がこの時間に含まれる可能性があります。

- **[アップロードの受信]** Mobile Link サーバでアップロード・ストリームの最初のバイトが受信されてから、リモート・データベースからのアップロード・ストリームが完全に受信されるまでの時間。アップロード・ストリームには、テーブルの定義や、アップロードされているリモート・データベースのローが含まれるので、ダウンロード専用同期でもこの時間が長くなる場合があります。時間は、アップロード・ストリームのサイズと、転送用のネットワーク帯域幅によって異なります。
- **[DB ワーカの取得]** 空いているデータベース・ワーカ・スレッドを取得するために必要な時間。
- **[接続]** 新しいデータベース接続が必要な場合に、データベース・ワーカ・スレッドでデータベース接続を確立するために必要な時間。たとえば、エラーの後や、スクリプトのバージョンが変更された場合に新しい接続が必要になります。
- **[ユーザを認証]** ユーザの認証に必要な時間。
- **[同期の開始]** begin_synchronization イベントが定義されている場合はこのイベントに必要な時間と、サブスクリプションごとに last_upload_time をフェッチするのに必要な時間の合計 (共有 Mobile Link 管理接続を使用)。
- **[アップロードの適用]** アップロードしたデータを統合データベースに適用するために必要な時間。
- **[ダウンロードの準備]** prepare_for_download イベントに必要な時間。
- **[ダウンロードのフェッチ]** 統合データベースからダウンロードするローをフェッチしてダウンロード・ストリームを作成するために必要な時間。
- **[同期の終了]** end_synchronization イベントに必要な時間。この後、データベース・ワーカ・スレッドが解放されます。ブロッキング・ダウンロード確認を使用している場合、このフェーズは**ダウンロード確認の待機**フェーズの後に発生します。それ以外の場合は、ダウンロード・ストリームがリモート・データベースに送信される前に発生します。
- **[ダウンロードの送信]** ダウンロード・ストリームをリモート・データベースに送信するために必要な時間。時間は、ダウンロード・ストリームのサイズと、転送用のネットワーク帯域幅によって異なります。アップロード専用同期の場合、ダウンロード・ストリームは単なるアップロード確認です。
- **[ダウンロード確認の待機]** ダウンロードがリモート・データベースに適用され、リモート・データベースからダウンロード確認が送信されるのを待つ時間。このフェーズは、リモート・データベースでダウンロード確認が有効になっている場合にのみ表示されます。
- **[ダウンロード確認の DB ワーカの取得]** ダウンロード確認を受信してから、データベース・ワーカ・スレッドが空くのを待つ時間。このフェーズは、リモート・データベースでダウンロード確認が有効になっていて、また Mobile Link サーバで非ブロッキング・ダウンロード確認が使用されている場合にのみ表示されます。
- **[ダウンロード確認の接続]** 新しいデータベース接続が必要な場合に、データベース・ワーカ・スレッドでデータベース接続を確立するために必要な時間。このフェーズは、リモート・データベースでダウンロード確認が有効になっていて、また Mobile Link サーバで非ブロッキング・ダウンロード確認が使用されている場合にのみ表示されます。

○ **[非ブロッキング・ダウンロード確認]** publication_nonblocking_download_ack 接続と nonblocking_download_ack 接続のイベントに必要な時間。このフェーズは、リモート・データベースでダウンロード確認が有効になっていて、また Mobile Link サーバで非ブロッキング・ダウンロード確認が使用されている場合にのみ表示されます。

各値は、値の検索や出力が容易になるように、"PHASE:" で始まります。

次に、さまざまな同期フェーズの期間を表している出力例を示します。

```
I. 2008-06-05 14:48:36. <1> PHASE: start_time: 2008-06-05 14:48:36.048
I. 2008-06-05 14:48:36. <1> PHASE: duration: 175
I. 2008-06-05 14:48:36. <1> PHASE: sync_request: 0
I. 2008-06-05 14:48:36. <1> PHASE: receive_upload: 19
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker: 0
I. 2008-06-05 14:48:36. <1> PHASE: connect: 18
I. 2008-06-05 14:48:36. <1> PHASE: authenticate_user: 51
I. 2008-06-05 14:48:36. <1> PHASE: begin_sync: 69
I. 2008-06-05 14:48:36. <1> PHASE: apply_upload: 0
I. 2008-06-05 14:48:36. <1> PHASE: prepare_for_download: 1
I. 2008-06-05 14:48:36. <1> PHASE: fetch_download: 4
I. 2008-06-05 14:48:36. <1> PHASE: wait_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: end_sync: 0
I. 2008-06-05 14:48:36. <1> PHASE: send_download: 10
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: connect_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: nonblocking_download_ack: 0
```

- **n** ロー・カウントの要約を表示します。
- **o** SQL パススルー・アクティビティを表示します。
- **p** 進行オフセットを表示します。
- **q** ダウンロードされた各ローのカラム値を表示します。アップロードとダウンロードされた各ローのカラム値を表示する -vr オプションの代わりにこのオプションを使用すると、ログに記録されるデータ量が減少します。-vi と -vq を同時に指定することは、-vr を指定することと同じです。
- **r** アップロードまたはダウンロードされた各ローのカラム値を表示します。アップロードされた各ローのカラム値だけをログに記録する場合は、-vi を使用します。ダウンロードされた各ローのカラム値だけをログに記録する場合は、-vq を使用します。
- **s** 呼び出された各同期スクリプトの名前を表示します。
- **t** ODBC 標準フォーマットのスクリプトから生成された、変換された SQL を表示します。このレベルには、c の機能が含まれます。次の例は、SQL Anywhere での文の自動変換を示します。

```
I. 02/11 11:02:14. [102]: begin_upload synch2
{ call SynchLogLine( ?, ?, 'begin_upload' ) }
I. 02/11 11:02:14. [102]: Translated SQL:
call SynchLogLine( ?, ?, 'begin_upload' )
```

次の例は、同じ文の Microsoft SQLServer での変換を示します。

```
I. 02/11 11:03:21. [102]: begin_upload synch2
{ call SynchLogLine( ?, ?, 'begin_upload' ) }
I. 02/11 11:03:21. [102]: Translated SQL:
EXEC SynchLogLine ?, ?, 'begin_upload'
```


- **u**　未定義のテーブル・スクリプトを表示します。これは、経験の浅いユーザが同期処理を理解する上で役立ちます。

参照

- 「Mobile Link の統計のプロパティ」 200 ページ

-w オプション

データベース・ワーカ・スレッド数を設定します。

構文

mlsrv11 -c "*connection-string***" -w** *count* ...

備考

各データベース・ワーカ・スレッドでは、同期要求を一度に 1 つ受け入れます。

データベース・ワーカ・スレッドはそれぞれ、統合データベースへの接続を 1 つ使用します。Mobile Link サーバは、接続をさらに 1 つ管理用に開きます。したがって、Mobile Link サーバから統合データベースへの接続の最小数は、*count* + 1 となります。

データベース・ワーカ・スレッド数は Mobile Link 同期スループットに大きく影響するので、特定の同期設定に最適なデータベース・ワーカ・スレッド数を判別するためのテストを実行する必要があります。データベース・ワーカ・スレッド数によって同時に統合データベースでアクティブにできる同期の数が決まります。残りの同期はキューイングされてデータベース・ワーカ・スレッドが使用可能になるのを待機します。データベース・ワーカ・スレッドを追加するとスループットが向上しますが、アクティブな同期の間で競合が発生する確率も高くなります。データベース・ワーカ・スレッドを追加していくと、ある時点で、複数の同期を同時に行うメリットよりも競合が発生する確率の方が大きくなり、スループットが低下します。

このオプションの設定値は -wu オプションのデフォルト設定にもなります。-wu オプションは、統合データベースに同時にアップロードできるスレッド数を制限するためのオプションです。これは、ダウンロードを行うデータベース・ワーカ・スレッドの最適数が、アップロードを行うデータベース・ワーカ・スレッドの最適数より多い場合に役立ちます。データベース・ワーカ・スレッド数を大きく設定し (-w を使用)、アップロードを同時に適用できるデータベース・ワーカ・スレッド数を小さく設定する (-wu を使用) と、最高のスループットを達成できる場合があります。通常、-wu の最適数は統合データベースによって異なり、リモート・データベースの処理速度またはネットワーク速度とはあまり関係ありません。したがって、-w を使用してスレッド数を増やす場合、-wu を使用して同時にアップロードできるスレッド数を制限できます。詳細については、「-wu オプション」 109 ページを参照してください。

データベース・ワーカ・スレッド数のデフォルトは **5** です。

参照

- 「-wu オプション」 109 ページ
- 「-sm オプション」 96 ページ
- 「-cn オプション」 58 ページ

-wu オプション

アップロードを同時に統合データベースに適用できるデータベース・ワーカ・スレッドの最大数を設定します。

構文

mlsrv11 -c "*connection-string***" -wu** *count* ...

備考

-wu オプションを使用して、アップロードを同時に統合データベースに適用できるデータベース・ワーカ・スレッド数を制限します。制限値に達すると、統合データベースへのアップロードの適用準備が完了しているデータベース・ワーカ・スレッドは、別のデータベース・ワーカ・スレッドがアップロードを終了するまで待機します。

統合データベースで発生する競合の最も一般的な原因は、アップロードを同時に適用するデータベース・ワーカ・スレッドが多すぎることです。ダウンロードで競合が発生することははるかに少ないので、ダウンロードは mlsrv11 -w オプションだけで制限されます。そのため、-w の設定は -wu の設定以上にしてください。

デフォルトでは、すべてのデータベース・ワーカ・スレッドで同時にアップロードを適用できます。使用されるデータベース・ワーカ・スレッドの数は -w オプションで設定します。デフォルトは **5** です。

例

LAN と PC 上のリモート・データベースを使用するパイロット設定では、アップロード専用同期とダウンロード専用同期の両方に対して、データベース・ワーカ・スレッドの最適数は約 10 であり、それが統合データベースの CPU 使用率 100% に相当します。データベース・ワーカ・スレッドの数が少ないと、スループットが低下し、統合データベースの CPU 使用率が低下します。データベース・ワーカ・スレッドの数が多くても、統合データベースの処理速度はデータベース・ワーカ・スレッド数が 10 の場合と同じなので、スループットは向上しません。

参照

- 「-w オプション」 108 ページ
- 「-sm オプション」 96 ページ

-x オプション

Mobile Link クライアントのネットワーク・プロトコルとプロトコル・オプションを設定します。Mobile Link サーバは、これらのプロトコルとパラメータを使用して同期要求を受信します。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

構文

mlsrv11 -c "*connection-string***" -x** *protocol*[*protocol-options*] ...

protocol : **tcpip** | **tls** | **http** | **https**

protocol-options : **(** *option*=*value*; ... **)**

デフォルト

デフォルトは TCPIP でポート 2439 を使用します。

パラメータ

許可されている protocol の値は次のとおりです。

- **tcpip** TCP/IP を使用した接続を受け入れます。
- **tls** トランスポート・レイヤ・セキュリティを使用する、TCP/IP を使用した接続を受け入れます。
- **http** 標準の Web プロトコルを使用した接続を受け入れます。
- **https** 安全なトランザクションを処理する HTTP の変形プロトコルを使用した接続を受け入れます。HTTPS プロトコルは、RSA または ECC 暗号化を使用して HTTP over SSL/TLS を実装します。

option=*value* の形式で次のネットワーク・プロトコル・オプションを指定することもできます。個々の複数のオプションは、セミコロンで区切ってください。

- **TCP/IP オプション** tcpip プロトコルを指定する場合は、次のプロトコル・オプションを任意で指定できます (これらのオプションでは、大文字と小文字は区別されます)。

TCP/IP プロトコル・オプション	説明
host=*hostname*	Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。

TCP/IP プロトコル・オプション	説明
ignore=*hostname*	接続を確立する場合に、Mobile Link サーバが無視するホスト名または IP アドレス。このオプションを使用すると、最も可能性が低いレベルにある負荷分散装置からの要求を無視することができ、Mobile Link サーバ・ログと Mobile Link モニタの出力ファイルへの過剰な出力を回避できます。無視するホストは複数指定できます。たとえば、-x tcpip(ignore=lb1;ignore=123.45.67.89) の形式で指定します。コマンド・ラインで -x の複数のインスタンスを指定した場合、すべてのインスタンスでホストが無視されます。たとえば、-x tcpip(ignore=1.1.1.1) -x http と指定した場合、1.1.1.1 の接続は TCP/IP と HTTP の両方のストリームで無視されます。ただし、これは -xo オプションを使用した接続には影響しません。
port=*portnumber*	Mobile Link サーバが受信に使用するソケット・ポート番号。デフォルトのポートは 2439 です。これは、Mobile Link サーバの IANA 登録ポート番号です。

- **TLS (TCP/IP とトランスポート・レイヤ・セキュリティ) オプション** tls プロトコル (トランスポート・レイヤ・セキュリティを使用する TCP/IP) を指定する場合は、次のプロトコル・オプションを任意で指定できます (これらのオプションでは、大文字と小文字は区別されます)。

TLS プロトコル・オプション	説明
fips={**yes**|**no**}	tls_type=rsa を使用して TLS プロトコルを指定した場合、fips=yes を指定して、TCP/IP プロトコルと、FIPS 認定の暗号化アルゴリズムを使用して接続を受け入れることができます。FIPS 接続では、別の FIPS 140-2 認定ソフトウェアを使用します。FIPS 認定でない RSA 暗号化を使用するサーバは、FIPS 認定の RSA 暗号化を使用するクライアントと互換性があり、FIPS 認定の RSA 暗号化を使用しているサーバは、FIPS 認定でない RSA 暗号化を使用しているクライアントと互換性があります。
host=*hostname*	Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。

TLS プロトコル・オプション	説明
ignore=*hostname*	接続を確立する場合に、Mobile Link サーバが無視するホスト名または IP アドレス。このオプションを使用すると、最も可能性が低いレベルにある負荷分散装置からの要求を無視することができ、Mobile Link サーバ・ログと Mobile Link モニタの出力ファイルへの過剰な出力を回避できます。無視するホストは複数指定できます。たとえば、-x tcpip(ignore=lb1;ignore=123.45.67.89) の形式で指定します。
port=*portnumber*	Mobile Link サーバが受信に使用するソケット・ポート番号。デフォルトのポートは 2439 です。これは、Mobile Link サーバの IANA 登録ポート番号です。
tls_type={rsa|ecc}	tls として TCP/IP プロトコルを指定した場合、楕円曲線暗号方式 (ecc) または RSA 暗号化 (rsa) を指定できます。下位互換のために、ecc を certicom と指定することもできます。デフォルトの tls_type は rsa です。 TLS を指定する場合は、次のように ID と ID パスワードを指定してください。 ○ **identity=*identity-file*** サーバ認証で使用される ID ファイルのパスとファイル名を指定します。 ○ **identity_password=*password*** ID のパスワードを指定します。 「トランスポート・レイヤ・セキュリティを使用する Mobile Link サーバの起動」『SQL Anywhere サーバ - データベース管理』を参照してください。
e2ee_type={rsa|ecc}	セッション・キーの交換に使用するキーのタイプ。rsa か ecc のどちらかにし、プライベート・キー・ファイルのキー・タイプに一致するタイプを指定してください (次のオプションを参照)。デフォルトの e2ee_type は rsa です。

TLS プロトコル・オプション	説明
e2ee_private_key=*file*	rsa プライベート・キーまたは ecc プライベート・キーを含む、PEM でコード化されたファイル。エンドツーエンド暗号化を有効にするためには、このオプションが必須です。 PEM でコード化されたファイルは、createkey ユーティリティを使用して作成します。「キー・ペア・ジェネレータ・ユーティリティ (createkey)」『SQL Anywhere サーバ - データベース管理』を参照してください。
e2ee_private_key_password=*password*	プライベート・キー・ファイルのパスワード。エンドツーエンド暗号化を有効にするためには、このオプションが必須です。

- **HTTP オプション**　http プロトコルを指定する場合は、次のプロトコル・オプションを任意で指定できます (これらのオプションでは、大文字と小文字は区別されます)。

HTTP オプション	説明
buffer_size=*number*	Mobile Link サーバから送信される HTTP メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTP メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65535 バイトです。
host=*hostname*	Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
port=*portnumber*	Mobile Link サーバが受信に使用するソケット・ポート番号。ポート番号は、Mobile Link サーバがモニタするように設定されているポートと一致させます。デフォルトのポートは 80 です。
version=*http-version*	Mobile Link サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルト・バージョンを指定する文字列です。1.0 または 1.1 を選択できます。デフォルト値は 1.1 です。

- **HTTP オプション**　HTTPS プロトコルでは、トランスポート・レイヤ・セキュリティで RSA デジタル証明書を使用します。FIPS 暗号化を指定すると、プロトコルは、https と互換性のある、別の FIPS 140-2 認定ソフトウェアを使用します。

  詳細については、「トランスポート・レイヤ・セキュリティを使用する Mobile Link サーバの起動」『SQL Anywhere サーバ - データベース管理』を参照してください。

  https プロトコルを指定する場合は、次のプロトコル・オプションを任意で指定できます (これらのオプションでは、大文字と小文字は区別されます)。

HTTP オプション	説明
buffer_size=*number*	Mobile Link サーバから送信される HTTPS メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTPS メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65535 バイトです。
identity=*server-identity*	サーバ認証で使用される ID ファイルのパスとファイル名。HTTPS では、RSA 証明書でなければなりません。
identity_password=*password*	ID ファイルのパスワードを指定する省略可能なパラメータ。 「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。
fips={yes|no}	fips=yes を指定すると、HTTPS プロトコルと、FIPS 認定の暗号化アルゴリズムを使用して、接続を受け入れることができます。FIPS 接続では、別の FIPS 140-2 認定ソフトウェアを使用します。FIPS 認定でない RSA 暗号化を使用するサーバは、FIPS 認定の RSA 暗号化を使用するクライアントと互換性があり、FIPS 認定の RSA 暗号化を使用しているサーバは、FIPS 認定でない RSA 暗号化を使用しているクライアントと互換性があります。
host=*hostname*	Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
port=*portnumber*	Mobile Link サーバが受信に使用するソケット・ポート番号。ポート番号は、Mobile Link サーバがモニタするように設定されているポートと一致させます。デフォルトのポートは 443 です。

HTTP オプション	説明
tls_type={**rsa**|**ecc**}	tls として TCP/IP プロトコルを指定した場合、楕円曲線暗号方式 (ecc) または RSA 暗号化 (rsa) を指定できます。下位互換のために、ecc を certicom と指定することもできます。デフォルトの tls_type は rsa です。 トランスポート・レイヤ・セキュリティを使用する場合は、ID と ID パスワードを指定する必要があります。 ○ **identity=*identity-file*** サーバ認証で使用される ID ファイルのパスとファイル名を指定します。 ○ **identity_password=password** ID ファイルのパスワードを指定します。 「トランスポート・レイヤ・セキュリティを使用する Mobile Link サーバの起動」『SQL Anywhere サーバ - データベース管理』を参照してください。
version=*http-version*	Mobile Link サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルト・バージョンを指定する文字列です。1.0 または 1.1 を選択できます。デフォルト値は 1.1 です。
e2ee_type={**rsa**|**ecc**}	セッション・キーの交換に使用するキーのタイプ。rsa か ecc のどちらかにし、プライベート・キー・ファイルのキー・タイプに一致するタイプを指定してください (次のオプションを参照)。デフォルトの e2ee_type は rsa です。
e2ee_private_key=*file*	rsa プライベート・キーまたは ecc プライベート・キーを含む、PEM でコード化されたファイル。エンドツーエンド暗号化を有効にするためには、このオプションが必須です。 PEM でコード化されたファイルは、createkey ユーティリティを使用して作成します。「キー・ペア・ジェネレータ・ユーティリティ (createkey)」『SQL Anywhere サーバ - データベース管理』を参照してください。

HTTP オプション	説明
e2ee_private_key_password=*password*	プライベート・キー・ファイルのパスワード。エンドツーエンド暗号化を有効にするためには、このオプションが必須です。

例

次のコマンド・ラインはポートを 12345 に設定します。

```
mlsrv11 -c "dsn=SQL Anywhere 11 CustDB;uid=ml_server;pwd=sql" -x tcpip(port=12345)
```

次の例では、セキュリティのタイプ (RSA)、サーバ ID ファイル、サーバのプライベート・キーを保護する ID パスワードを指定します。

```
mlsrv11 -c "dsn=my_cons"
 -x tls(tls_type=rsa;identity=c:¥test¥serv_rsa1.crt;identity_password=pwd)
```

次の例は前の例と似ていますが、ID ファイル名にスペースが含まれる点だけが異なります。

```
mlsrv11 -c "dsn=my_cons"
 -x "tls(tls_type=rsa;identity=c:¥Program Files¥test¥serv_rsa1.crt;identity_password=pwd)"
```

次の例は、HTTPS を使用したエンドツーエンド暗号化の使い方を示します。

```
mlsrv11 -c "dsn=my_cons" -x https(tls_type=rsa;identity=my_identity.crt;
identity_password=my_id_pwd;e2ee_type=rsa;e2ee_private_key=my_pk.pem;
e2ee_private_key_password=my_pk_pwd)
```


-xo オプション

Mobile Link バージョン 8 と 9 のクライアントのネットワーク・プロトコルとプロトコル・オプションを設定します。

構文

mlsrv11 -c "*connection-string***"**
-xo *protocol*[*protocol-options*] ...

protocol-options : **(** *keyword***=***value*; ... **)**

備考

クライアント・アプリケーションとの通信に使用する通信プロトコルを指定します。デフォルトは TCPIP でポート 2439 を使用します。

許可されている protocol の値は次のとおりです。

- **tcpip** TCP/IP を介したアプリケーションからの接続を受け入れます。
- **http** 標準の Web プロトコルを使用した接続を受け入れます。クライアント・アプリケーションは HTTP のバージョンを選択でき、Mobile Link サーバは接続ごとにバージョンを調整します。
- **https** 安全なトランザクションを処理する HTTP の変形プロトコルを使用した接続を受け入れます。HTTPS プロトコルは、RSA 暗号化を使用して HTTP over SSL/TLS を実装します。また、他の HTTPS サーバと互換性があります。
- **https_fips** HTTPS プロトコルと、FIPS 認定の暗号化アルゴリズムを使用して、接続を受け入れます。HTTPS_FIPS では、別の FIPS 140-2 認定ソフトウェアを使用します。rsa_tls を使用しているサーバは、rsa_tls_fips を使用しているクライアントと互換性があり、rsa_tls_fips を使用しているサーバは、rsa_tls を使用しているクライアントと互換性があります。

必要に応じて、*option=value* の形式でネットワーク・プロトコル・オプションを指定することもできます。個々の複数のオプションは、セミコロンで区切ってください。指定できるオプションは、選択するプロトコルによって異なります。

- **TCP/IP オプション** tcpip プロトコルを指定する場合は、次のプロトコル・オプションを任意で指定できます。
  - **backlog=number-of-connections** リモート接続の最大数。この値を超えて新しい同期要求が行われると Mobile Link サーバによって拒否されるため、クライアント側で同期が失敗します。バックログのサイズを指定することで、サーバがビジーなときにクライアントが同期を待ち続けることを回避できます。バッグログのサイズを指定しないと、Mobile Link サーバはできるだけ多くの接続を受け入れるので、ネットワーク接続に対するオペレーティング・システムの制限に達したり超える可能性があります。これにより、処理速度が低下したり、誤動作が発生することがあります。

    クライアントは、リモート接続の最大数をすでに受け入れている Mobile Link サーバと同期しようとすると、エラー・コード -85 (SQLE_COMMUNICATIONS_ERROR) を受信しま

す。クライアント・アプリケーションはこのエラーを処理して、数分後に接続を再び試みます。

SQLE_COMMUNICATIONS_ERROR の詳細については、「通信エラーが発生しました。」『エラー・メッセージ』を参照してください。

数千の同時同期が可能な環境で Mobile Link を使用している場合は、backlog オプションを使用して、リモート接続の最大数をオペレーティング・システムの制限よりも少なく指定します。「オペレーティング・システムの制限の考慮」 176 ページを参照してください。

- **host=hostname**　Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
- **ignore=hostname**　接続を確立する場合に、Mobile Link サーバが無視するホスト名または IP アドレス。このオプションを使用すると、最も可能性が低いレベルにある負荷分散装置からの要求を無視することができ、Mobile Link サーバ・ログと Mobile Link モニタの出力ファイルへの過剰な出力を回避できます。無視するホストは複数指定できます。たとえば、-x tcpip(ignore=lb1;ignore=123.45.67.89) の形式で指定します。
- **liveness_timeout=n**　クライアントとの最後の通信から、Mobile Link が同期を終了するまでの時間 (秒単位)。値 0 はタイムアウトがないことを意味します。デフォルトは 120 秒です。
- **port=portnumber**　Mobile Link サーバが受信に使用するソケット・ポート番号。デフォルトのポートは 2439 です。これは、Mobile Link サーバの IANA 登録ポート番号です。

> **注意**
> mlsrv11 の -x と -xo の各オプションでは、同じデフォルトのポートが使用され、-x は指定しなくても開始されます。したがって、-x オプションでポートを変更しなかった場合は -xo のポートを変更する必要があります。

- **security=cipher(keyword=value;...)**　この接続を介して行われるすべての通信は、トランスポート・レイヤ・セキュリティを使用して暗号化および認証されます。cipher には次のいずれかを指定してください。

暗号化 (cipher)	説明
rsa_tls	RSA 暗号化。
rsa_tls_fips	FIPS 認定の RSA 暗号化。rsa_tls_fips は、FIPS 140-2 認定ソフトウェアという別のソフトウェアを使用しますが、https (SQL Anywhere バージョン 9.0.2 以降) を使用するクライアントと互換性があります。
ecc_tls	楕円曲線暗号方式。下位互換のために、ecc_tls を certicom_tls と指定することもできます。

セキュリティ・パラメータは、certificate (サーバ認証で使用される ID ファイルのパスとファイル名) と certificate_password です。選択した暗号パッケージ・プログラムと一致する証明書を使用してください。

詳細については、「トランスポート・レイヤ・セキュリティを使用する Mobile Link サーバの起動」『SQL Anywhere サーバ - データベース管理』を参照してください。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」『SQL Anywhere 11 - 紹介』を参照してください。

- **HTTP オプション** http プロトコルを指定する場合は、次のプロトコル・オプションを任意で指定できます。
  - **backlog=number-of-connections** リモート接続の最大数。この値を超えて新しい同期要求が行われると Mobile Link サーバによって拒否されるため、クライアント側で同期が失敗します。バックログのサイズを指定することで、サーバがビジーなときにクライアントが同期を待ち続けることを回避できます。バックログのサイズを指定しないと、Mobile Link サーバはできるだけ多くの接続を受け入れるので、ネットワーク接続に対するオペレーティング・システムの制限に達したり超える可能性があります。これにより、処理速度が低下したり、誤動作が発生することがあります。

    クライアントは、リモート接続の最大数をすでに受け入れている Mobile Link サーバと同期しようとすると、エラー・コード -85 (SQLE_COMMUNICATIONS_ERROR) を受信します。クライアント・アプリケーションはこのエラーを処理して、数分後に接続を再び試みます。

    SQLE_COMMUNICATIONS_ERROR の詳細については、「通信エラーが発生しました。」『エラー・メッセージ』を参照してください。

    数千の同時同期が可能な環境で Mobile Link を使用している場合は、backlog オプションを使用して、リモート接続の最大数をオペレーティング・システムの制限よりも少なく指定します。詳細については、「オペレーティング・システムの制限の考慮」 176 ページを参照してください。
  - **buffer_size=number** Mobile Link サーバから送信される HTTP メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTP メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65535 バイトです。
  - **contd_timeout=seconds** 同期を中止する前に、部分的に完了した同期の次の部分を受信するまで Mobile Link サーバが待機する秒数。クライアントが接続を続行できないことが待機時間で示されている場合は Mobile Link サーバのリソースを解放するように、このオプションを調整できます。デフォルト値は 30 秒です。
  - **host=hostname** Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
  - **port=portnumber** Mobile Link サーバが受信に使用するソケット・ポート番号。ポート番号は、Mobile Link サーバがモニタしているポートと一致させます。デフォルトのポートは 80 です。

注意
mlsrv11 の -x と -xo の各オプションでは、同じデフォルトのポートが使用され、-x は指定しなくても開始されます。したがって、-x オプションでポートを変更しなかった場合は -xo のポートを変更する必要があります。

- **session_key={ cookie | header }** 接続の追跡に使用する、JSESSIONID の代わりを作成します。ネットワークで JSESSIONID がすでに使用されている場合に必要になることがあります。
- **unknown_timeout=seconds** 同期を中止する前に、新しい接続の HTTP ヘッダを受信するまで Mobile Link サーバが待機する秒数。ネットワーク障害が発生したことが待機時間で示されている場合は Mobile Link サーバのリソースを解放するように、このオプションを調整できます。デフォルト値は 30 秒です。
- **version=http-version** Mobile Link サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルト・バージョンを指定する文字列です。1.0 または 1.1 を選択できます。デフォルト値は 1.1 です。

● **HTTPS または HTTPS_FIPS オプション** https プロトコルでは、トランスポート・レイヤ・セキュリティで RSA デジタル証明書を使用します。https_fips プロトコルでは、別の FIPS 140-2 認定ソフトウェアを使用しますが、https と互換性があります。

詳細については、「トランスポート・レイヤ・セキュリティを使用する Mobile Link サーバの起動」 『SQL Anywhere サーバ - データベース管理』を参照してください。

別途ライセンスが必要な必須コンポーネント
ECC 暗号化と FIPS 認定の暗号化には、別途ライセンスが必要です。強力な暗号化テクノロジはすべて、輸出規制対象品目です。

「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

https プロトコルを指定する場合は、次のプロトコル・オプションを任意で指定できます。

- **backlog=number-of-connections** リモート接続の最大数。この値を超えて新しい同期要求が行われると Mobile Link によって拒否されるため、クライアント側で同期が失敗します。バックログのサイズを指定することで、サーバがビジーなときにクライアントが同期を待ち続けることを回避できます。バックログのサイズが指定されていない場合、クライアントは、バックログのサイズにかかわらず同期を試行します。
- **buffer_size=number** Mobile Link サーバから送信される HTTPS メッセージの本文の最大サイズ (バイト単位)。このオプションを変更すると、HTTPS メッセージの送信に割り当てられるメモリ量が減少または増加します。デフォルトは 65535 バイトです。
- **contd_timeout=seconds** 同期を中止する前に、部分的に完了した同期の次の部分を受信するまで Mobile Link サーバが待機する秒数。クライアントが接続を続行できないことが待機時間で示されている場合は Mobile Link サーバのリソースを解放するように、このオプションを調整できます。デフォルト値は 30 秒です。

- **host=hostname**　Mobile Link サーバが受信に使用するホスト名または IP アドレス。デフォルト値は localhost です。
- **port=portnumber**　Mobile Link サーバが受信に使用するソケット・ポート番号。ポート番号は、Mobile Link サーバがモニタするように設定されているポートと一致させます。デフォルトのポートは 443 です。

  注意
  mlsrv11 の -x と -xo の各オプションでは、同じデフォルトのポートが使用され、-x は指定しなくても開始されるので、-x オプションでポートを変更しなかった場合は -xo のポートを変更する必要があります。

- **certificate**　サーバ認証で使用される証明書ファイルのパスとファイル名。これは RSA 証明書でなければなりません。
- **certificate_password**　証明書ファイルのパスワードを指定する省略可能なパラメータ。

  セキュリティの詳細については、「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。
- **session_key={ cookie | header }**　接続の追跡に使用する、JSESSIONID の代わりを作成します。ネットワークで JSESSIONID がすでに使用されている場合に必要になることがあります。
- **unknown_timeout=seconds**　同期を中止する前に、新しい接続の HTTP ヘッダを受信するまで Mobile Link サーバが待機する秒数。ネットワーク障害が発生したことが待機時間で示されている場合は Mobile Link サーバのリソースを解放するように、このオプションを調整できます。デフォルト値は 30 秒です。
- **version=http-version**　Mobile Link サーバは、クライアントが使用している HTTP のバージョンを自動的に検出します。このパラメータは、クライアントが使用しているバージョンをサーバが検出できない場合に使用される、HTTP のデフォルト・バージョンを指定する文字列です。1.0 または 1.1 を選択できます。デフォルト値は 1.1 です。

例

次のコマンド・ラインは、バックログのサイズを 10 接続に設定します。

```
mlsrv11 -c "dsn=SQL Anywhere 11 CustDB;uid=ml_server;pwd=sql" -xo http(backlog=10)
```

-zp オプション

競合検出にどのタイムスタンプ値を使用するかを調整します。

構文

mlsrv11 -c "*connection-string***" -zp**

備考

このオプションを使用すると、統合データベースとリモート・データベース間にタイムスタンプの競合がある場合に、最小精度よりも高い精度を持つタイムスタンプ値を競合検出に使用できます。リモート・データベース上の更新されたタイムスタンプによって、次の同期で競合が発生する場合があるため、このオプションは、統合データベースのタイムスタンプがリモート・データベースのタイムスタンプよりも精度が高い場合に有用です。このオプションを指定すると、Mobile Link はそのような競合を無視します。精度が不一致で -zp が使用されていない場合は、同期ごとにスキーマが異なるテーブル別の警告がログに書き込まれるので、-zp オプションの使用を推奨します。可能であればリモート・データベースのタイムスタンプの精度を調整するようユーザに通知する、同期ごとの警告もさらに追加されます。

-zs オプション

mlstop 用の Mobile Link サーバ名と、サーバ・ファームの共有サーバ・ステータスを指定します。

構文

mlsrv11 -c "*connection-string***" -zs** *name*

備考

指定する名前には、ASCII の英数字を使用できますが、その他の文字は使用できません。

-zs オプションを使用して起動した Mobile Link サーバの停止に mlstop を使用するときは、サーバ名を mlstop のコマンド・ラインで指定する必要があります。たとえば mlstop myMLserver のようになります。Mobile Link サーバがインストールされているコンピュータからしか、シャットダウンは開始できません。

Mobile Link がサーバ・ファームで実行されている場合は、サーバをユニークに識別するためにこの名前を指定してください。

参照

- 「Mobile Link 停止ユーティリティ (mlstop)」 727 ページ
- 「Mobile Link サーバをサーバ・ファームで実行する」 40 ページ
- 「-lsc オプション」 74 ページ
- 「-ss オプション」 97 ページ
- 「Mobile Link サーバ・ファームでの Notifier」 『Mobile Link - サーバ起動同期』

-zt オプション

Mobile Link サーバを実行するのに使用されるプロセッサの最大数を指定します。

構文

mlsrv11 -c "*connection-string***" -zt** *number*

備考

一部の ODBC ドライバでは、このオプションが必須です。また、プロセッサ・リソースを厳密に制御できます。

このオプションは、Windows オペレーティング・システムでしか使用できません。デフォルトは、コンピュータに搭載されているプロセッサの数です。

-zu オプション

authenticate_user スクリプトが未定義の場合に、ユーザの自動的な追加を制御します。

構文

mlsrv11 -c "*connection-string***" -zu{ + | - }** ...

備考

このオプションを -zu+ として指定すると、認識されなかった Mobile Link ユーザ名が最初の同期時に ml_user テーブルに追加されます。-zu- を引数に指定するか、まったく指定しない場合は、認識されないユーザ名を同期できません。

このオプションは、開発中にユーザを登録するのに使用すると便利です。配備されたアプリケーションでの使用はおすすめしません。

参照

- 「新しいユーザからの同期」 『Mobile Link - クライアント管理』
- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』
- 「Mobile Link ユーザ認証ユーティリティ (mluser)」 729 ページ
- 「authenticate_user 接続イベント」 377 ページ

-zus オプション

アップロードするローがテーブルにないときでも、Mobile Link サーバがテーブルのアップロード・スクリプトを呼び出すようにします。

構文

mlsrv11 -c "*connection-string***" -zus** ...

備考

デフォルトでは、テーブルのローがアップロードされない場合、Mobile Link サーバは、定義されていてもそのテーブルのアップロード・スクリプトを呼び出しません。このオプションはデフォルトの動作を無効にして、ローがアップロードされなくても、Mobile Link サーバがテーブルのアップロード・スクリプトを呼び出すようにします。

-zw オプション

表示する警告メッセージのレベルを制御します。

構文

mlsrv11 -c "*connection-string***" -zw** *levels*

備考

Mobile Link には、5 つのレベルの警告メッセージがあります。

レベル	説明
0	すべての警告メッセージを表示しない
1	サーバ・レベルと高い ODBC レベル：Mobile Link サーバが起動するときに警告メッセージを表示
2	同期レベルとユーザ・レベル：同期が開始するときに警告メッセージを表示
3	スキーマ・レベル：Mobile Link サーバがクライアント・スキーマを処理するときに警告メッセージを表示
4	スクリプト・レベルと低い ODBC レベル：Mobile Link サーバがスクリプトをフェッチ、準備、または実行するときに警告メッセージを表示
5	テーブル・レベルまたはロー・レベル：Mobile Link サーバがアップロードまたはダウンロードでテーブル操作を実行するときに警告メッセージを表示

レポートする警告メッセージのレベルを指定する場合は、複数のレベルをカンマで区切るか、2 つのドットで範囲を指定できます。たとえば、**-zw 1..3,5** は **-zw 1,2,3,5** と同じです。

メッセージのレポートはパフォーマンスにほとんど影響しません。レベル数が高いほど、多くのメッセージが生成される傾向があります。

同じコマンド・ラインで -zw を 2 回以上使用すると、Mobile Link は最後のインスタンスのみを認識します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

デフォルトは **1,2,3,4,5** です。この場合、すべてのレベルの警告メッセージが表示されます。

-zwd オプション

特定の警告コードを無効にします。

構文

mlsrv11 -c "*connection-string***" -zwd** *code*, ...

備考

特定の警告コードを無効にすると、同じレベルの他のコードがレポートされる場合でも、その警告コードはレポートされません。

警告メッセージ・コードの完全なリストについては、「Mobile Link サーバの警告メッセージ」『エラー・メッセージ』を参照してください。

同じコマンド・ラインで -zwd を 2 回以上使用すると、Mobile Link は設定を累積します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

-zwe オプション

特定の警告コードを有効にします。

構文

mlsrv11 -c "*connection-string***" -zwe** *code*, ...

備考

特定の警告コードを有効にすると、-zw で同じレベルの他のコードを無効にしてある場合でも、その警告コードがレポートされます。

警告メッセージ・コードの完全なリストについては、「Mobile Link サーバの警告メッセージ」『エラー・メッセージ』を参照してください。

同じコマンド・ラインで -zwe を 2 回以上使用すると、Mobile Link は設定を累積します。-zw、-zwd、-zwe の設定が競合する場合は、-zwe、-zwd、-zw の優先順位で処理されます。

同期の方法

目次

Mobile Link 開発のヒント

同期機能をアプリケーションに追加すると、複雑なアプリケーションを作成できます。以下のヒントを参考にしてください。

同期機能をプロトタイプ・アプリケーションに追加する場合、トラブルの原因となっているコンポーネントを推測するのは困難です。プロトタイプの開発時に、アプリケーションの内部に INSERT 文を一時的にハード・エンコードして、テストやデモンストレーションのためのデータを作成してください。プロトタイプが正常に動作するようになったら、同期を有効にして、一時的に使用した INSERT 文を削除します。

簡単な同期の方法から始めてください。簡単なアップロードまたはダウンロードを行う場合、スクリプトは 1 つか 2 つしか必要ありません。スクリプトが正しく動作していれば、タイムスタンプ、プライマリ・キー・プール、競合解決などのより高度な方法を導入できます。

Mobile Link とプライマリ・キー

同期システムでは、プライマリ・キーは、異なるデータベース (リモートと統合) 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。したがって、Mobile Link アプリケーションは以下の規則に従う必要があります。

- 同期される各テーブルには、プライマリ・キーが存在する必要がある。
- プライマリ・キーの値は更新しない。
- プライマリ・キーは、同期されるすべてのデータベース間でユニークでなければならない。

「ユニークなプライマリ・キーの管理」 143 ページ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

タイムスタンプによる方法は、効率よく同期するために最も便利な一般的な手法です。この方法では、各ユーザが最後に同期を行った時間が追跡され、それ以降に変更されたローだけがダウンロードされます。

Mobile Link は、各 Mobile Link ユーザが最後にデータをダウンロードした日時を示すタイムスタンプ値を管理します。この値は、「最終ダウンロード時刻」と呼ばれます。

「スクリプトでの最終ダウンロード時刻の使用」 134 ページを参照してください。

♦ **テーブル用にタイムスタンプベースの同期を実装するには、次の手順に従います。**

1. 統合データベースで、ローの最終修正時刻を保持するカラムを追加します。通常、このカラムは次のように宣言されます。

DBMS	最終変更カラム
Adaptive Server Enterprise	datetime
IBM DB2 LUW	timestamp
IBM DB2 メインフレーム	timestamp
Microsoft SQL Server	datetime
MySQL	timestamp
Oracle	date
SQL Anywhere	timestamp DEFAULT timestamp

2. download_cursor イベントと download_delete_cursor イベントのスクリプト内で、最初のパラメータを timestamp カラムの値と比較します。

例

次のテーブル宣言とスクリプトによって、Contact サンプルの Customer テーブルに対するタイムスタンプベースの同期が実装されます。

- テーブル定義

```
CREATE TABLE "DBA"."Customer"(
  "cust_id"  integer NOT NULL DEFAULT GLOBAL AUTOINCREMENT,
  "name"  char(40) NOT NULL,
  "rep_id"  integer NOT NULL,
  "last_modified" timestamp NULL DEFAULT timestamp,
  "active"  bit NOT NULL,
  PRIMARY KEY ("cust_id") )
```

- download_delete_cursor スクリプト

```
SELECT cust_id
FROM Customer JOIN SalesRep
ON Customer.rep_id = SalesRep.rep_id
WHERE Customer.last_modified >= {ml s.last_table_download}
   AND ( SalesRep.ml_username != {ml s.username}
        OR Customer.active = 0 )
```

- download_cursor スクリプト

```
SELECT cust_id, Customer.name, Customer.rep_id
FROM Customer KEY JOIN SalesRep
WHERE Customer.last_modified >= {ml s.last_table_download}
   AND SalesRep.ml_username = {ml s.username}
   AND Customer.active = 1
```

「同期論理のソース・コード」『Mobile Link - クイック・スタート』と「Contact サンプルの顧客窓口の同期」『Mobile Link - クイック・スタート』を参照してください。

スクリプトでの最終ダウンロード時刻の使用

最終ダウンロード・タイムスタンプは、多くの Mobile Link イベントにパラメータとして指定されます。ダウンロード・フェーズの直前に、最後に成功した同期中に統合データベースから取得された値です。現在の Mobile Link のユーザが同期を行ったことがない場合や同期に成功したことがない場合、この値は 1900-01-01 に設定されます。

「ダウンロード・タイムスタンプの生成および使用方法」 135 ページを参照してください。

複数のパブリケーションがあり、それらを異なる時間に同期させている場合には、2 つの異なる最終ダウンロード・タイムスタンプを持つことができます。このため、最終ダウンロード・タイムスタンプには次の 2 つのスクリプト・パラメータ名があります。

- **last_table_download**　テーブルの最終ダウンロード・タイムスタンプです。
- **last_download**　すべてのテーブルが同期されていた最後の時間です。どのテーブルでも、last_table_download の最も古い値になります。

Mobile Link スクリプトで名前付きパラメータの代わりに疑問符を使用すると、正しい値が常に使用されます。

警告
SQL Anywhere 統合データベースを使用していて、最終変更情報を保持しているカラムが DEFAULT TIMESTAMP 型の場合は、カラムを同期しないでください。リモート・データベースがこのようなカラムを要求する場合は、別のカラム名を使用してください。そうしないと、デフォルトのタイムスタンプ値が、アップロードされた値で上書きされ、統合データベースでローが最後に変更された時刻が保持されません。

参照

- 「スクリプトのパラメータ」 336 ページ

例

```
SELECT cust_id, Customer.name, Customer.rep_id
FROM Customer KEY JOIN SalesRep
WHERE Customer.last_modified >= {ml s.last_table_download}
   AND SalesRep.ml_username = {ml s.username}
   AND Customer.active = 1
```

ダウンロード・タイムスタンプの生成および使用方法

Mobile Link は、次のようにタイムスタンプベースのダウンロードのタイムスタンプを生成して使用します。

- アップロードのコミット後、prepare_for_download イベントを呼び出す直前に、Mobile Link サーバは統合データベースから現在の時刻をフェッチして、値を保存します。このタイムスタンプ値は現在のダウンロードの開始時刻を表します。次回の同期では、この時刻の後に変更されたデータのみをダウンロードします。
- Mobile Link サーバは、ダウンロードの一部としてこのタイムスタンプ値を送信し、クライアントはそれを保存します。
- クライアントは次回同期するとき、アップロードと一緒に送信する **last_download_timestamp** にこのタイムスタンプ値を使用します。
- Mobile Link サーバは、クライアントがアップロードしたばかりの last_download_timestamp を download_cursor と download_delete_cursor に渡します。すると、カーソルは、最後の last_download_timestamp 以降のタイムスタンプを持つ変更を選択できるので、新しい変更だけがダウンロードされるようになります。

最終ダウンロード時刻が格納される場所

最終ダウンロード時刻は、リモート・データベースに格納されます。この場所は、ダウンロードが正常に適用されたかどうかがわかっているのはリモートのみであるという理由で適切な場所です。

SQL Anywhere リモートでは、最終ダウンロード時刻はサブスクリプションごとに格納されます。「SYSSYNC システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

Ultra Light リモートでは、最終ダウンロード時刻はパブリケーションごとに格納されます。「syspublication システム・テーブル」『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。

最終ダウンロード時刻の変更

まれな状況として、last_download_timestamp の変更が必要な場合があります。たとえば、リモート・データベースのすべてのデータを誤って削除した場合には、最終ダウンロード・タイムスタンプの値をリセットする modify_last_download_timestamp 接続スクリプトを定義して、リモート・データベースを再同期できます。modify_next_last_download_timestamp という別のイベントもあ

ります。これを使用すると、現在の同期ではなく、次回の同期のタイムスタンプをリセットできます。次の項を参照してください。

- 「modify_last_download_timestamp 接続イベント」 486 ページ
- 「modify_next_last_download_timestamp 接続イベント」 489 ページ

Ultra Light にも、リモートから最終ダウンロード時刻を変更する機能があります。次の項を参照してください。

- C/C++ Embedded SQL：「ULResetLastDownloadTime 関数」 『Ultra Light - C/C++ プログラミング』
- C++：「ResetLastDownloadTime 関数」 『Ultra Light - C/C++ プログラミング』
- .NET 2.0：「ResetLastDownloadTime メソッド」 『Ultra Light - .NET プログラミング』

参照

- 「スクリプトでの最終ダウンロード時刻の使用」 134 ページ

夏時間への対応

分散データベース・システムでは、データの同期が夏時間への切り替え時に重なると問題が発生します。たとえば、データを消失する可能性があります。これは夏時間から元に戻った結果、あいまいになりかねない 1 時間がある秋にだけ問題になります。

夏時間に対応するには、次の 3 つの方法があります。

- 統合データベース・サーバで UTC (協定世界時) を使用する。
- 統合データベース・サーバで夏時間を無効にする。
- 時刻の切り替え時の 1 時間はシャットダウンする。

スナップショットを使った同期

ほとんどの同期には、タイムスタンプベースの同期が適しています。ただし、スナップショットによってデータの更新をしたい場合も考えられます。

スナップショットを使ってテーブルを同期する場合、テーブルのローのうちで関係するローすべてを完全にダウンロードします。すでにダウンロード済みのローもダウンロードされます。この方法が最も簡単ですが、不必要に大量のデータ・セットが交換されるため、パフォーマンスが悪くなります。

スナップショットを使った同期によって、テーブルのすべてのローをダウンロードできます。また、ローの分割方法と組み合わせて実行することもできます。「リモート・データベース間でのローの分割」 139 ページを参照してください。

スナップショットを使った同期をいつ行うか

通常、次の特徴を両方満たすテーブルに対してスナップショットを使用すると最も有効です。

- **ロー数が比較的少ない**　ローの数が少ない場合は、ローを全部ダウンロードしても大きなオーバーヘッドにはなりません。
- **頻繁にローが変更される**　テーブルのほとんどのローが頻繁に変更される場合は、前回の同期の後で変更されていないローを明示的に除外してもあまり効果はありません。

テーブルの内容が為替レートのリストになっている場合は、通貨の種類はそれほど多くないので、この方法が適しています。また、ほとんどのレートは頻繁に更新されます。ビジネスの性質によって、価格、利率のリスト、または最新ニュース項目といった内容を含むテーブルが考えられます。

♦ スナップショットベースの同期を実装するには、次の手順に従います。

1. リモート・ユーザが値を更新しない場合は、アップロード・スクリプトを未定義のままにしておきます。
2. テーブルのローを削除する場合は、リモート・テーブルのローをすべて削除する download_delete_cursor スクリプトを作成するか、少なくともすべてのローがもう必要ないことを定義します。統合データベースからローを削除しないで、削除のマークを付けてください。ローの値を知らないと、リモート・データベースからローを削除できません。

   「download_delete_cursor スクリプトの作成」 353 ページを参照してください。
3. download_cursor スクリプトを作成し、リモート・テーブルに登録するローをすべて選択します。

ローの削除

統合データベースからローを削除しないで、削除のマークを付けてください。ローの値を知らないと、リモート・データベースからローを削除できません。download_cursor スクリプトの場合はマークなしのローだけを、download_delete_cursor スクリプトの場合はマーク付きのローだけを選択します。

download_delete_cursor スクリプトは、download_cursor スクリプトより先に実行されます。ダウンロードにローが含まれる場合は、同じプライマリ・キーを持つローを削除リスト内に含める必要はありません。ダウンロードしたローをリモート・ロケーションで取得するときに、同じプライマリ・キーを持つ既存のローは置き換えられます。

「ローをダウンロードするスクリプトの作成」 351 ページを参照してください。

別の削除方法

リモート・データベースからローを削除する場合、download_cursor スクリプトを使わなくても、リモート・アプリケーションを使ってローを削除できます。たとえば、同期のすぐ後に、アプリケーションで SQL 文を実行して不要なローを削除できます。

アプリケーションによって削除されたローは、通常は次回の同期で Mobile Link サーバにアップロードされますが、STOP SYNCHRONIZATION DELETE 文を使って、アップロードされないようにできます。次に例を示します。

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM table-name
 WHERE expiry_date < CURRENT TIMESTAMP;
COMMIT;
START SYNCHRONIZATION DELETE;
```

「download_delete_cursor スクリプトの作成」 353 ページを参照してください。

スナップショットの例

サンプル・アプリケーションの ULProduct テーブルは、スナップショットを使った同期によって管理されます。テーブルに入っているローの数は比較的少ないため、スナップショットを使った同期でのオーバヘッドがわずかです。

1. アップロード・スクリプトがありません。これは、リモート・データベースでは製品情報を追加できないという業務意思を反映しています。
2. download_delete_cursor スクリプトがないため、リストから製品を削除しないものと見なします。
3. download_cursor スクリプトによって、現在のすべての製品に関して、prod_id、price、prod_name が選択されます。既存の製品の場合は、リモート・テーブル内のその製品の価格が更新されます。新しい製品の場合は、リモート・テーブルにローが挿入されます。

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

ローの数が極めて少ないテーブルでのスナップショットを使った同期の別の例については、「Contact サンプルの営業担当者の同期」『Mobile Link - クイック・スタート』を参照してください。

リモート・データベース間でのローの分割

各 Mobile Link リモート・データベースは、異なるデータのサブセットを統合データベース内に持つことができます。つまり、自分専用の同期スクリプトを作成して、リモート・データベース間でデータを「分割」できます。

共通部分がないように切断分割にすることも、重複を持たせて分割することもできます。たとえば、従業員ごとに独自の顧客セットを持っていて、かつ顧客を共有していない場合は、「切断」分割になります。複数のリモート・データベースに存在するように顧客を共有している場合、分割は「重複」を含みます。

分割は、テーブル用のスクリプトである download_cursor と download_delete_cursor で実行されます。これらのスクリプトによって、リモート・データベースにローをダウンロードするように定義します。各スクリプトは、パラメータとして Mobile Link ユーザ名を使用します。スクリプトでこのパラメータを WHERE 句に指定して、ユーザごとに適切なローを取得します。

切断分割

download_cursor スクリプトと download_delete_cursor スクリプトによって、同期で使用するテーブルごとに分割を制御します。このスクリプトでは、最後のダウンロードのタイムスタンプと、同期を呼び出すときに指定する Mobile Link ユーザ名という 2 つのパラメータを使用します。

♦ **リモート・データベース間でテーブルを分割するには、次の手順に従います。**

1. テーブル定義でカラムを指定し、そこに統合データベースの同期ユーザ名を持たせます。このカラムをリモート・データベースにダウンロードする必要はありません。
2. このカラムがスクリプトのパラメータと一致するように、download_cursor スクリプトと downloade_delete_cursor スクリプトの WHERE 句に条件文を指定します。

   スクリプト・パラメータは、スクリプト内で疑問符または名前付きパラメータによって表すことができます。次の例では、download_cursor スクリプトによって、テーブル Contact を emp_id で分割します。

   ```
   SELECT id, contact_name
   FROM Contact
   WHERE last_modified >= {ml s.last_table_download}
   AND emp_id = {ml s.username}
   ```

   「download_cursor テーブル・イベント」 418 ページと「download_delete_cursor テーブル・イベント」 422 ページを参照してください。

例

CustDB サンプル・アプリケーション内のプライマリ・キー・プール・テーブルを使って、リモート・データベースごとに独自のプライマリ・キー値のセットを指定します。この方法は、プライマリ・キーの重複を避けるために使用します。詳細については、「プライマリ・キー・プールの使用」 147 ページを参照してください。

この方法では、プライマリ・キー・プール・テーブルがリモート・データベース間で切断分割されるようにしてください。

キー・プール・テーブル ULCustomerIDPool にあるプライマリ・キー値は、各ユーザが顧客を追加するときに使用します。ULCustomerIDPool テーブルには次の 3 つのカラムがあります。

- **pool_cust_id**　ULCustomer テーブルで使用するプライマリ・キー値。このカラムだけがリモート・データベースにダウンロードされます。
- **pool_emp_id**　このプライマリ・キーの所有者である従業員。
- **last_modified**　このテーブルは、last_modified カラムに基づいたタイムスタンプを使って管理されます。

  タイムスタンプの同期については、「タイムスタンプベースのダウンロード」 133 ページを参照してください。

このテーブルの download_cursor スクリプトを次に示します。

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

変数または名前付きパラメータを使用しない場合は、プレースホルダの "**?**" を含んだ、ジョインまたはサブ選択を使用できます。

「Contact サンプルの顧客の同期」 『Mobile Link - クイック・スタート』と「Contact サンプルの顧客窓口の同期」 『Mobile Link - クイック・スタート』を参照してください。

重複のある分割

統合データベースの一部のテーブルは、複数のリモート・データベースに属するローを持ちます。各リモート・データベースは統合データベース内にローのサブセットを持ち、さらにそのサブセットは他のリモート・データベースと重複しています。顧客テーブルの場合は、こうしたことがよく起こります。この場合、顧客テーブルと複数のリモート・データベース間で多対多の関係があり、通常、この関係を表すテーブルが存在します。download_cursor イベントと download_delete_cursor イベントのスクリプトでは、関係を表すテーブルにダウンロードされるテーブルをジョインする必要があります。

例

CustDB サンプル・アプリケーションでは、この方法を ULOrder テーブルに使用します。ULEmpCust テーブル上での ULCustomer と ULEmployee の関係は、多対多の関係です。

各リモート・データベースは、ULOrder テーブルから、emp_id カラムの値が Mobile Link ユーザ名と一致するローのみを受信します。

CustDB アプリケーションの ULOrder テーブルで、SQL Anywhere バージョンの download_cursor スクリプトを使用した例を次に示します。

```
SELECT o.order_id, o.cust_id, o.prod_id,
  o.emp_id, o.disc, o.quant, o.notes, o.status
```



```
FROM ULOrder o , ULEmpCust ec
WHERE o.cust_id = ec.cust_id
 AND ec.emp_id = {ml s.username}
 AND ( o.last_modified >= {ml s.last_table_download}
  OR ec.last_modified >= {ml s.last_table_download})
 AND  ( o.status IS NULL
  OR o.status != 'Approved' )
 AND ( ec.action IS NULL )
```

このスクリプトは非常に複雑です。スクリプトを見ると、リモート・データベースのテーブルを定義するクエリには、統合データベースのテーブルを複数指定できることがわかります。このスクリプトは、次のすべての条件に一致する ULOrder のローをすべてダウンロードします。

- ULOrder の cust_id カラムと ULEmpCust の cust_id カラムが一致する。
- ULEmpCust の emp_id カラムが同期ユーザ名と一致する。
- 注文、または従業員と顧客の関係に対する最終更新日がどちらも、同期ユーザ用の最新の同期時間よりも新しい。
- ステータスは "**Approved**" 以外である。

ULEmpCust のアクション・カラムを使用して、削除するカラムにマークを付けます。その目的は、現在説明している内容には関係ありません。

download_delete_cursor スクリプトを次に示します。

```
SELECT o.order_id
FROM ULOrder o, ULEmpCust ec
WHERE o.cust_id = ec.cust_id
  AND ec.emp_id = {ml s.username}
  AND ( o.last_modified >= {ml s.last_table_download} OR
        c.last_modified >= {ml s.last_table_download} )
  AND ( o.status IS NULL OR
        o.status != 'Approved' )
  AND ( ec.action IS NULL )
```

リモート・データベースから "Approved" のローがすべて削除されます。

子テーブルの分割

前の項の例では、他のテーブルの基準に従ってテーブルを分割する方法を説明しています。「重複のある分割」 140 ページを参照してください。

リモート・データベースの一部のテーブルには、切断のサブセットか重複したサブセットがありますが、サブセットを決定するカラムはありません。子テーブルには、通常、別のテーブルを参照する 1 つの外部キー (または一連の外部キー) があります。参照先のテーブルにはカラムがあり、これによって適切なサブセットが決定されます。

この場合、download_cursor スクリプトと download_delete_cursor スクリプトでは、参照先のテーブルをジョインしたり、WHERE 句を使用してローを適切なサブセットに制限したりする必要があります。

例については、「Contact サンプルの顧客窓口の同期」 『Mobile Link - クイック・スタート』を参照してください。

アップロード専用の同期とダウンロード専用の同期

デフォルトでは、同期は双方向です。つまり、データはアップロードおよびダウンロードされます。しかし、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を選択できます。

同期モデルに関するメモ
このトピックでは、Mobile Link 同期システムをデータベースに作成するときにアップロード専用およびダウンロード専用の同期を設定する方法について説明します。アップロード専用またはダウンロード専用は、Sybase Central で同期モデルを作成する場合にも指定できます。

SQL Anywhere リモート・データベース

- **アップロード**　アップロード専用の同期を実行するには、dbmlsync オプション -uo または拡張オプション UploadOnly を使用します。次の項を参照してください。

  ○「-uo オプション」『Mobile Link - クライアント管理』
  ○「UploadOnly (uo) 拡張オプション」『Mobile Link - クライアント管理』

- **ダウンロード**　ダウンロード専用の同期を実行するには、dbmlsync オプション -ds または拡張オプション DownloadOnly を使用します。次の項を参照してください。

  ○「-ds オプション」『Mobile Link - クライアント管理』
  ○「DownloadOnly (ds) 拡張オプション」『Mobile Link - クライアント管理』

  SQL Anywhere リモート・データベースはダウンロード専用のパブリケーションを使用することもできます。このダウンロード方法はダウンロード専用同期とは異なります。「ダウンロード専用のパブリケーション」『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

- **アップロード**　アップロード専用の同期を実行するには、Upload Only 同期パラメータを使用します。

  「Upload Only 同期パラメータ」『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。

- **ダウンロード**　ダウンロード専用の同期を実行するには、Download Only 同期パラメータを使用します。

  「Download Only 同期パラメータ」『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。

ユニークなプライマリ・キーの管理

同期される各テーブルには、プライマリ・キーが必要です。また、プライマリ・キーは同期対象のすべてのデータベース間でユニークでなければなりません。プライマリ・キーの値は更新しないようにしてください。

多くの場合、テーブルのプライマリ・キーとして単一のカラムを使用すると便利です。たとえば、顧客にはそれぞれユニークな ID 値を割り当ててください。営業担当者全員がデータベース接続を直接管理できる環境で作業する場合は、ID 番号の割り当ては簡単に実施できます。顧客テーブルに新しい顧客が挿入されると、テーブルに最後に追加された値よりも大きな値を持つ新規のプライマリ・キーが自動的に追加されます。

接続が切断された環境では、新しいローの挿入時に、プライマリ・キーにユニークな値を割り当てるのは簡単ではありません。営業担当者が新しい顧客を追加する場合は、顧客テーブルのリモート・コピーにも同様な操作を行います。自分以外の営業担当者が、顧客テーブルの自分以外のコピーに対して操作を行っている場合、同じ顧客 ID 値を使わせないようにします。

この項では、ユニークなプライマリ・キーの生成に関する問題を解決する次の方法について説明します。

- 「複合キーの使用」 143 ページ
- 「UUID の使用」 143 ページ
- 「グローバル・オートインクリメントの使用」 144 ページ
- 「プライマリ・キー・プールの使用」 147 ページ

複合キーの使用

Mobile Link リモート ID は、同期システム内のリモート・データベースをユニークに定義します。したがって、ユニークなプライマリ・キーを簡単に作成するには、値の一部として Mobile Link リモート ID を含む複合プライマリ・キーを作成します。ユニークな Mobile Link ユーザ名を保持している場合には、リモート ID の代わりにユーザ名を使用できます。

「リモート ID」 『Mobile Link - クライアント管理』を参照してください。

UUID の使用

newid( ) 関数を使用してプライマリ・キーに対して完全にユニークな値を作成することによって、プライマリ・キーをユニークにすることができます。作成された UUID は、uuidtostr( ) 関数を使用して文字列に変換できます。また、strtouuid( ) 関数を使用してバイナリに戻すことができます。

UUID は GUID とも呼ばれ、すべてのコンピュータを通じてユニークです。ただし、この値は完全にランダムのため、値が追加された日時や値の順序を判別することはできません。また、UUID の値は他の方法 (グローバル・オートインクリメントを含む) で必要な値よりかなり大きく、プライマリ・キー・テーブルと外部キー・テーブルの両方でより多くのテーブル領域を必要とします。UUID を使用するテーブルのインデックスも効率性に劣ります。

参照

SQL Anywhere データベース：

- 「NEWID デフォルト」『SQL Anywhere サーバ - SQL の使用法』
- 「NEWID 関数 [その他]」『SQL Anywhere サーバ - SQL リファレンス』
- 「UNIQUEIDENTIFIER データ型」『SQL Anywhere サーバ - SQL リファレンス』

Ultra Light データベース：

- 「Ultra Light でのプライマリ・キーの一意性」『Ultra Light データベース管理とリファレンス』
- 「NEWID 関数 [その他]」『Ultra Light データベース管理とリファレンス』

例

次の SQL Anywhere の CREATE TABLE 文は、完全にユニークなプライマリ・キーを作成します。

```
CREATE TABLE customer (
  cust_key UNIQUEIDENTIFIER NOT NULL
        DEFAULT NEWID( ),
  rep_key VARCHAR(5),
  PRIMARY KEY(cust_key))
```

グローバル・オートインクリメントの使用

SQL Anywhere と Ultra Light のデータベースでは、デフォルトのカラム値を GLOBAL AUTOINCREMENT に設定できます。このデフォルト設定は、ユニークな値を管理するカラムのすべてに適用できますが、特にプライマリ・キーの場合に役立ちます。

♦ **グローバル・オートインクリメント・カラムを使用するには、次の手順に従います。**

1. カラムをグローバル・オートインクリメント・カラムとして宣言します。

   このデフォルトのグローバル・オートインクリメントを指定すると、そのカラムの値のドメインが分割されます。各分割には同じ数の値が含まれます。たとえば、データベース内の整数カラムの分割サイズを 1000 に設定した場合、1 つの分割が 1001 から 2000 まで拡大します。また、2 つ目の分割は 2001 から 3000 まで拡大し、以降、同じように拡大していきます。

   「デフォルトのグローバル・オートインクリメントの宣言」 145 ページを参照してください。

2. global_database_id 値を設定します。

   SQL Anywhere では、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割からのみ設定されます。たとえば、データベースに ID 番号 10 を割り当て、分割サイズが 1000 の場合、このデータベースのデフォルト値は 10001 ～ 11000 の範囲から選択されます。このデータベースの別のコピーで、ID 番号 11 が割り当てられたデータベースからは、11001 ～ 12000 の範囲にある同一カラムのデフォルト値が指定されます。

   「グローバル・データベース ID の設定」 145 ページを参照してください。

デフォルトのグローバル・オートインクリメントの宣言

Sybase Central でカラムのプロパティを選択するか、CREATE TABLE 文または ALTER TABLE 文に DEFAULT GLOBAL AUTOINCREMENT フレーズを組みこむことで、作業データベースにデフォルト値を設定できます。

オプションで、AUTOINCREMENT キーワードの直後にカッコで分割サイズを指定できます。この分割サイズには任意の正の整数を設定できますが、通常、分割サイズは、サイズの値がすべての分割で不足しないように選択されます。

INT または UNSIGNED INT 型のカラムの場合、デフォルトの分割サイズは $2^{16} = 65536$ です。他の型のカラムの場合、デフォルトの分割サイズは $2^{32} = 4294967296$ です。特にカラムが INT または BIGINT 型以外のときは、これらのデフォルトが適切ではない場合があるため、分割サイズを明示的に指定してください。

たとえば、次の SQL 文では 2 つのカラム (顧客 ID 番号を保持する整数カラム、顧客名を保持する文字列カラム) を持つ簡単なテーブルが作成されます。分割サイズは 5000 に設定されています。

```
CREATE TABLE customer (
 id   INT         DEFAULT GLOBAL AUTOINCREMENT (5000),
 name VARCHAR(128) NOT NULL,
 PRIMARY KEY (id)
)
```

参照

- SQL Anywhere：「CREATE TABLE 文」『SQL Anywhere サーバ - SQL リファレンス』
- Ultra Light：「Ultra Light CREATE TABLE 文」『Ultra Light データベース管理とリファレンス』

グローバル・データベース ID の設定

アプリケーションを配備するときには、各データベースに対して必ず異なる ID 番号を割り当てます。ID 番号はさまざまな方法で作成して配布できます。テーブルに値を設定し、リモート ID など、ユニークなプロパティに基づいて、各データベースに適切なローをダウンロードするのも 1 つの方法です。

♦ **グローバル・データベース ID 番号を設定するには、次の手順に従います。**

- SQL Anywhere では、パブリック・オプション global_database_id の値を設定して、データベースのグローバル ID を設定します。ID 番号は正の整数にします。「global_database_id オプション [データベース]」『SQL Anywhere サーバ - データベース管理』を参照してください。

  Ultra Light では、global_id オプションを設定して、データベースのグローバル ID を設定します。「Ultra Light global_database_id オプション」『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。

デフォルト値の選択方法

グローバル・データベース ID は、SQL Anywhere ではパブリック・オプション global_database_id、Ultra Light では global_id オプションを使用して設定します。

各データベース内のグローバル・データベース ID オプションは、ユニークな正の整数に設定してください。特定のデータベースのデフォルト値の範囲は、$pn + 1$ ～ $p(n + 1)$ です。ここで、p は分割サイズ、n はグローバル・データベース ID の値を表します。たとえば、分割サイズを 1000、グローバル・データベース ID を 3 に設定すると、範囲は 3001 ～ 4000 になります。

SQL Anywhere と Ultra Light では、次の規則を適用してデフォルト値を選択します。

- カラムに現在の分割の値が含まれていない場合、最初のデフォルト値は $pn + 1$ である。ここで、p は分割サイズ、n はグローバル・データベース ID の値を表します。
- カラムに現在の分割の値が含まれていても、そのすべてが $p(n + 1)$ 未満であれば、この範囲内でこれまで使用した最大値より 1 大きい値が次のデフォルト値になる。
- デフォルトのカラム値は、現在の分割以外のカラムの値の影響を受けない。つまり、$pn + 1$ より小さいか $p(n + 1)$ より大きい数には影響されない。Mobile Link 同期を介して別のデータベースからレプリケートされた場合に、このような値が存在する可能性があります。

グローバル・データベース ID がデフォルト値の 2147483647 に設定されると、カラムには NULL 値が挿入されます。NULL 値が許可されていない場合に、ローの挿入を試みるとエラーが発生します。たとえば、テーブルのプライマリ・キーにカラムが含まれている場合に、この状況が発生します。

グローバル・データベース ID には負の値は設定できないので、正の値が常に選択されます。ID 番号の最大値を制限するのは、カラムのデータ型と分割サイズだけです。

デフォルトの NULL 値は、分割で値が不足したときにも生成されます。この場合には、データベースに新しいグローバル・データベース ID 値を割り当てて、別の分割からデフォルト値を選択できるようにしてください。カラムで NULL が許可されていない場合、NULL 値を挿入しようとするとエラーが発生します。未使用の値が残り少ないことを検出し、このような状態を処理するには、GlobalAutoincrement タイプのイベントを作成できます。

特定の分割で値が不足する場合は、新しいグローバル・データベース ID をそのデータベースに割り当てることができます。方法が適切なものであれば、新しいデータベース ID 番号を割り当てることができます。未使用のデータベース ID 値のプールを管理する方法も、その 1 つです。このプールは、プライマリ・キー・プールと同じ方法で管理されます。

分割で値が不足しそうな場合に、自動的にデータベース管理者へ通知する (またはその他のアクションを実行する) ようにイベント・ハンドラを設定できます。SQL Anywhere データベースについては、「イベントのトリガ条件の定義」『SQL Anywhere サーバ - データベース管理』を参照してください。

参照

- 「グローバル・データベース ID の設定」 145 ページ
- SQL Anywhere：「global_database_id オプション [データベース]」 『SQL Anywhere サーバ - データベース管理』
- Ultra Light：「Ultra Light global_database_id オプション」 『Ultra Light データベース管理とリファレンス』

例

SQL Anywhere データベースでは、次の文はデータベース ID 番号を 20 に設定します。

```
SET OPTION PUBLIC.global_database_id = 20
```

特定カラムの分割サイズが 5000 の場合、このデータベースのデフォルト値は 100001 ～ 105000 の範囲から選択されます。

プライマリ・キー・プールの使用

ユニークなプライマリ・キーのこの問題を解決する効果的な方法の 1 つは、データベースの各ユーザに、必要に応じて使用できるプライマリ・キー値のプールを割り当てることです。たとえば、営業担当者ごとに 100 個の新しい ID 値を割り当てます。各営業担当者は、自分のプール内の値を、新しい顧客に自由に割り当てることができます。

♦ **プライマリ・キー・プールを実装するには、次の手順に従います。**

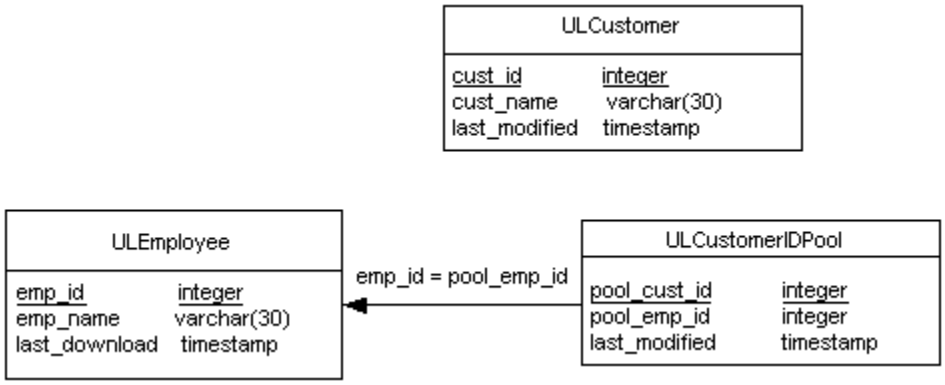
1. 統合データベースと各リモート・データベースに新しいテーブルを追加して、新しいプライマリ・キー・プールを格納します。ユニークな値を格納するカラムとは別に、これらのテーブルにはユーザ名を格納するカラムが必要です。このユーザ名のカラムによって、値を割り当てる権限を持つ者を識別できます。
2. ストアド・プロシージャを作成し、十分な数の新しい ID 値が各ユーザに確実に割り当てられるようにします。新しいエントリを多数挿入する、または同期をあまり行わないリモート・ユーザには、特に多く、新しい値を割り当てます。
3. download_cursor スクリプトを作成して、各ユーザに割り当てられた新しい値を選択し、それをリモート・データベースにダウンロードします。
4. リモート・データベースを使用するアプリケーションを変更し、ユーザが新しいローを挿入するときに、プールに入っている値をアプリケーションが使用するようにします。アプリケーションは、その値をプールから削除して、値の再使用を防ぎます。
5. アップロード・スクリプトを作成します。ユーザがリモート・データベースの自分専用の値プールから削除した値と対応するローが、Mobile Link サーバによって、統合データベースの値のプールから削除されます。
6. end_upload スクリプトを作成し、値のプールを管理するストアド・プロシージャを呼び出します。これで、ユーザのプールに対してさらに多くの値が追加され、削除済みの値がアップロード中に置き換わります。

例

リモート・ユーザは、サンプル・アプリケーションを使って顧客を追加できます。新しいローにはそれぞれユニークなプライマリ・キー値があることが必要です。ただし、データ・エントリ中は、まだ各リモート・データベースは切断された状態です。

ULCustomerIDPool にはプライマリ・キー値のリストがあり、この値を各リモート・データベースで使用できます。また、値を使い切ってしまうと、ULCustomerIDPool_maintain ストアド・プロシージャによってプール内の値が完全に補充されます。管理プロシージャは、テーブルレベルの end_upload スクリプトによって呼び出されます。各リモート・データベースのプールは、upload_insert スクリプトと download_cursor スクリプトによって管理されます。

1. 統合データベースの ULCustomerIDPool テーブルには、新しい顧客 ID 番号のプールが格納されます。ULCustomer テーブルとは直接のリンク関係はありません。



2. ULCustomerIDPool_maintain プロシージャによって、統合データベースの ULCustomerIDPool テーブルが更新されます。SQL Anywhere 統合データベース用のサンプル・コードを次に示します。

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
   DECLARE pool_count INTEGER;

   -- Determine how may 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
```

このプロシージャは、現在のユーザに現在割り当てられている番号をカウントします。また、新しいローを挿入して、このユーザが十分な数の顧客 ID 番号を使用できるようにします。

このプロシージャは、ULCustomerIDPool テーブル用の end_upload テーブル・スクリプトによって、アップロードの最後に呼び出されます。スクリプトを次に示します。


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

3. ULCustomerIDPool テーブル用の download_cursor スクリプトは、リモート・データベースに新しい番号をダウンロードします。

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

4. 新しい顧客を挿入するには、リモート・データベースを使用しているアプリケーションで、プール中の未使用の ID 番号を選択して、その番号をプールから削除してから、この ID 番号による新しい顧客情報を挿入します。次に示す Ultra Light アプリケーション用の Embedded SQL 関数は、プールから新しい顧客番号を取り出します。

```
bool CDemoDB::GetNextCustomerID( void )
/************************************/
{
    short ind;

    EXEC SQL SELECT min( pool_cust_id )
    INTO :m_CustID:ind FROM ULCustomerIDPool;
    if( ind < 0 ) {
        return false;
    }
    EXEC SQL DELETE FROM ULCustomerIDPool
    WHERE pool_cust_id = :m_CustID;
    return true;
}
```

競合の解決

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する可能性があります。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 つ目のローが到着したときに競合が検出されます。

デフォルトでは次のように処理されます。

- ローを挿入しようとしたときに、そのローが挿入済みであることが検出されると、エラーが生成されます。
- ローを削除しようとしたときに、そのローが削除済みであることが検出されると、2 回目の削除の試行は無視されます。

異なる動作が必要な場合は、この項で説明するアップロード・イベントを 1 つ以上定義して動作を実装できます。

競合について

警告
同期テーブルのプライマリ・キーは更新しないでください。プライマリ・キーは、異なるデータベース (リモートと統合) 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なので、プライマリ・キーを更新すると、プライマリ・キーの目的が無効になります。

競合はエラーとは異なります。競合が起こる可能性がある場合は、適切な値を計算するプロセスを定義するか、最低でも競合のログを取ってください。優れたアプリケーションを設計するには、競合の解決は不可欠です。

同期のダウンロード処理中は、リモート・データベースでは競合は発生しません。ダウンロードしたローに新しいプライマリ・キーが含まれている場合は、その値は新しいローに挿入されます。新しいプライマリ・キーが既存のローのプライマリ・キーと一致する場合は、そのローの値が更新されます。

例

User1 が最初に 10 個の在庫を売り出し、そのうち 3 個を販売して、Remote1 にある在庫の値を 7 個に更新します。User2 は 4 個販売し、Remote2 にある在庫を 6 に更新します。Remote1 が同期を実行すると、統合データベースは 7 に更新されます。Remote2 が同期を実行すると、在庫の値が 10 ではなくなっているため、競合が検出されます。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。
2. Remote2 がアップロードした新しいローの値。
3. Remote2 が最後の同期中に取得した古いローの値。

この場合、ビジネス論理は新しい在庫数を計算し、競合を解決するために次の方法を使用できます。


```
current consolidated - (old remote - new remote)
-> 7 - (10-6) = 3
```

競合の処理方法に関するその他の例については、次の項を参照してください。

- 「Contact サンプルの製品の同期」 『Mobile Link - クイック・スタート』

競合の検出

Mobile Link クライアントが更新済みのローを Mobile Link サーバに送信するときは、更新された新しい値 (更新後イメージ) だけでなく、最後のダウンロード、またはこのローの最初のアップロード以前に存在していたローの値から取得された古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

競合を検出するスクリプトがいくつか用意されています。Mobile Link サーバは、次のいずれかのスクリプトが適用された場合のみ競合を検出します。

- upload_fetch または upload_fetch_column_conflict スクリプト
- WHERE 句に指定されたカラムがすべて非プライマリ・キー・カラムである upload_update スクリプト

upload_fetch スクリプトによる競合の検出

テーブルに対して upload_fetch または upload_fetch_column_conflict スクリプトを定義すると、Mobile Link サーバは、アップロードの更新前イメージを、upload_fetch スクリプトから返される、同じプライマリ・キー値を持つローの値と比較します。更新前イメージの値が統合データベースの現在の値と一致しない場合、Mobile Link サーバは競合を検出します。

upload_fetch スクリプトは、更新対象となるローに対応する統合データベースのテーブルから、データの単一のローを選択します。通常の upload_fetch スクリプトの構文は次のとおりです。

```
SELECT pk1, pk2, ...col1, col2, ...
FROM table-name
WHERE pk1 = {ml r.pk1} AND pk2 = {ml r.pk2} ...
  AND col1 = {ml r.col1} AND col2 = {ml r.col2} ...
```

「upload_fetch テーブル・イベント」 525 ページを参照してください。

upload_fetch_column_conflict イベントは upload_fetch に似ていますが、2 人のユーザが同じカラムを更新した場合のみ競合を検出します。異なるユーザは、同じカラムを更新しないかぎり、同じローを更新することができ、競合は発生しません。

「upload_fetch_column_conflict テーブル・イベント」 528 ページを参照してください。

リモート・データベースのテーブルごとに、upload_fetch または upload_fetch_column_conflict スクリプトを 1 つのみ指定できます。

統合データベースのローのロック

upload_fetch スクリプトによって競合が検出されてから、競合解決が完了するまでに、統合データベースのローが変更される可能性があります。データが正しくなくなる可能性があるこの問題を回避するには、ロー・ロックを使用して upload_fetch スクリプトまたは upload_fetch_column_conflict スクリプトを実装できます。

SQL Anywhere 統合データベースでは、UPDLOCK キーワードと HOLDLOCK キーワードのどちらも使用できますが、同時実行性については UPDLOCK の方が適しています。次に例を示します。

```
SELECT column-names from table-name WITH (UPDLOCK)
  WHERE where-clause
```

Oracle、DB2 LUW、DB2 メインフレームの場合は、FOR UPDATE を使用します。次に例を示します。

```
SELECT column-names FROM table-name
  WHERE where-clause
    FOR UPDATE OF column_name1, column_name3, column_name6
```

注意
更新するカラム名を上の例のように指定すると、コンピュータのリソースが保持され、パフォーマンスが向上します。

Microsoft SQL Server の場合は、HOLDLOCK を使用します。次に例を示します。

```
SELECT column-names FROM table-name WITH (HOLDLOCK)
  WHERE where-clause
```

Adaptive Server Enterprise の場合は、HOLDLOCK を使用します。次に例を示します。

```
SELECT column-names FROM table-name
  HOLDLOCK
  WHERE where-clause
```

例

upload_fetch スクリプトを定義します。Mobile Link サーバはこのスクリプトを使用して、統合データベースの現在のローを取り出し、更新されたローの更新前イメージとこのローを比較します。2 つのローに同じ値が含まれている場合、競合はありません。2 つのローが異なる場合には、競合が検出され、Mobile Link は upload_old_row_insert と upload_new_row_insert スクリプトを呼び出し、その後に resolve_conflict スクリプトを呼び出します。

「resolve_conflict スクリプトによる競合の解決」 154 ページを参照してください。

upload_update スクリプトによる競合の検出

upload_update スクリプトを使用して競合を検出するには、以下のようにすべてのカラムを WHERE 句に含めます。

UPDATE *table-name*
SET *col1* **= {ml r.***col1***},** *col2* **= {ml r.** *col2* **}** ...

WHERE *pk1* **= {ml o.***pk1***} AND** *pk2* **= {ml o.***pk2***}** ...
AND *col1* **= {ml o.***col1***} AND** *col2* **= {ml o.***col2***}** ...

この文では、col1 や col2 はプライマリ・キー・カラムではありませんが、pk1 や pk2 はプライマリ・キー・カラムです。非プライマリ・キー・カラム (o.) の 2 番目のセットに渡される値は、更新ローの更新前イメージ (古い値) です。WHERE 句は、リモート・データベースから更新された古い値と、統合データベースの現在の値を比較します。これらの値が一致しないと更新は無視されるので、すでに統合データベースにあった値は保持されます。

「upload_update テーブル・イベント」 548 ページを参照してください。

upload_fetch または upload_fetch_column_conflict によって競合が検出されない場合のみ、upload_update スクリプトを競合検出に使用します。

シナリオ 1

upload_update、upload_old_row_insert、upload_new_row_insert、resolve_conflict の各イベントのスクリプトを定義します。

次の upload_update スクリプトを定義します。

```
UPDATE product
SET name={ml r.name}, description={ml r.description}
WHERE id={ml r.id}
  AND name={ml o.name}
  AND description={ml o.description}
```

Mobile Link は更新を実行して、修正されたローの数をチェックします。1 つのローも修正されていない場合、Mobile Link は、統合データベースのどのローも更新前イメージのローと一致しない競合を検出します。Mobile Link は upload_old_row_insert と upload_new_row_insert スクリプトを呼び出し、その後に resolve_conflict スクリプトを呼び出します。

「resolve_conflict スクリプトによる競合の解決」 154 ページを参照してください。

シナリオ 2

upload_old_row_insert、upload_new_row_insert、resolve_conflict のスクリプトを定義しません。代わりに、競合の検出と解決を処理するストアド・プロシージャを作成して、upload_update スクリプトで呼び出します。

「upload_update スクリプトによる競合の解決」 155 ページを参照してください。

競合の解決

競合を解決するために、いくつかのオプションがあります。

- 競合が発生した場合に、テンポラリ・テーブルまたは永久テーブルと resolve_conflict スクリプトを使用して解決する。

  「resolve_conflict スクリプトによる競合の解決」 154 ページを参照してください。

- 競合が発生した場合に、upload_update スクリプトを使用して解決する。

  「upload_update スクリプトによる競合の解決」 155 ページを参照してください。

- テーブル用の end_upload スクリプトを使用して、すべての競合を一度に解決する。

  「end_upload テーブル・イベント」 456 ページを参照してください。

resolve_conflict スクリプトによる競合の解決

Mobile Link サーバが upload_fetch スクリプトを使用して競合を検出すると、次のイベントが発生します。

- Mobile Link サーバは、リモート・データベースからアップロードされた古いロー値を upload_old_row_insert スクリプトの定義に従って挿入します。通常、古い値はテンポラリ・テーブルに挿入されます。

  「upload_old_row_insert テーブル・イベント」 535 ページを参照してください。

- Mobile Link サーバは、リモート・データベースからアップロードされた新しいロー値を upload_new_row_insert スクリプトの定義に従って挿入します。通常、新しい値はテンポラリ・テーブルに挿入されます。

  「upload_new_row_insert テーブル・イベント」 532 ページを参照してください。

- Mobile Link サーバは、resolve_conflict スクリプトを実行します。このスクリプトでは、ストアド・プロシージャを呼び出したり、実行手順の順序を定義したりすることによって、新しいロー値と古いロー値を使用して競合を解決できます。

詳細については、「resolve_conflict テーブル・イベント」 508 ページを参照してください。

例

次の例では、6 つのイベントに対してスクリプトを作成し、次にストアド・プロシージャを作成します。

- begin_synchronization スクリプトでは、contact_new と contact_old という 2 つのテンポラリ・テーブルを作成します。(begin_connection スクリプトでこれを行うこともできます)。
- upload_fetch スクリプトは、競合を検出します。
- 競合がある場合、upload_old_row_insert スクリプトと upload_new_row_insert スクリプトは、リモート・データベースからアップロードした新しいデータと古いデータを使用して、2 つのテンポラリ・テーブルを設定します。
- resolve_conflict スクリプトは、MLResolveContactConflict ストアド・プロシージャを呼び出して、競合を解決します。

イベント	スクリプト
begin_synchronization	CREATE TABLE #contact_new(id INTEGER, location CHAR(36), contact_date DATE); CREATE TABLE #contact_old(id INTEGER, location CHAR(36), contact_date DATE)
upload_fetch	SELECT id, location, contact_date FROM contact WHERE id = {ml r.id}
upload_old_row_insert	INSERT INTO #contact_new(id, location, contact_date) VALUES ({ml r.id}, {ml r.location}, {ml r.contact_date})
upload_new_row_insert	INSERT INTO #contact_old(id, location, contact_date) VALUES ({ml r.id}, {ml r.location}, {ml r.contact_date})
resolve_conflict	CALL MLResolveContactConflict( )
end_synchronization	DROP TABLE #contact_new; DROP TABLE #contact_old

MLResolveContactConflict ストアド・プロシージャは次のとおりです。

```
CREATE PROCEDURE MLResolveContactConflict( )
BEGIN
  --update the consolidated database only if the new contact date
  --is later than the existing contact date
  UPDATE contact c
    SET c.contact_date = cn.contact_date
      FROM #contact_new cn
      WHERE c.id = cn.id
        AND cn.contact_date > c.contact_date;
  --cleanup
  DELETE FROM #contact_new;
  DELETE FROM #contact_old;
END
```

upload_update スクリプトによる競合の解決

競合を解決するのに resolve_conflict スクリプトを使用する代わりに、upload_update スクリプトでストアド・プロシージャを呼び出すこともできます。この方法では、プログラムで競合の検出と解決の両方を行う必要があります。

ストアド・プロシージャでは、すべてのカラムを含んでいるが、更新前イメージの (古い) 値を使用する WHERE 句のある upload_update スクリプトのフォーマットを使用してください。

以下は、upload_update スクリプトの例です。

```
{CALL UpdateProduct(
   {ml o.id}, {ml o.name}, {ml o.desc}, {ml r.name}, {ml r.desc}
)
}
```

以下は、UpdateProduct ストアド・プロシージャの例です。

```
CREATE PROCEDURE UpdateProduct(
  @id INTEGER,
  @preName VARCHAR(20),
  @preDesc VARCHAR(200),
  @postName VARCHAR(20),
  @postDesc VARCHAR(200) )
BEGIN
    UPDATE product
     SET name = @postName, description = @postDesc
     WHERE id = @id
       AND name = @preName
       AND description = @preDesc
    IF @@rowcount=0 THEN
       // A conflict occurred: handle resolution here.
    END IF
END
```

この方法は、resolve_conflict スクリプトによる競合の解決よりも管理が簡単です。それは、管理するスクリプトが 1 つだけで、すべての論理が 1 つのストアド・プロシージャに含まれているからです。ただし、テーブル・カラムが NULL 入力可の場合、または BLOB や CLOB が含まれている場合には、ストアド・プロシージャのコードが複雑になる可能性があります。また、Mobile Link 統合データベースとしてサポートされている RDBMS の一部には、ストアド・プロシージャに渡すことができる値のサイズに制限があります。

次の項を参照してください。

- 「upload_update スクリプトによる競合の検出」 152 ページ
- 「upload_update テーブル・イベント」 548 ページ
- 「resolve_conflict スクリプトによる競合の解決」 154 ページ

例

次のストアド・プロシージャ sp_update_my_customer には、競合を検出し解決するための論理が定義されています。このストアド・プロシージャは古いカラム値と新しいカラム値を受け入れます。この例は SQL Anywhere の機能を使用します。スクリプトは次のように実装できます。

```
{CALL sp_update_my_customer(
   {ml o.cust_1st_pk},
   {ml o.cust_2nd_pk},
   {ml o.first_name},
   {ml o.last_name},
   {ml o.nullable_col},
   {ml o.last_modified},
   {ml r.first_name},
   {ml r.last_name},
   {ml r.nullable_col},
   {ml r.last_modified}
)}
CREATE PROCEDURE sp_update_my_customer(
    @cust_1st_pk        INTEGER,
    @cust_2nd_pk        INTEGER,
    @old_first_name     VARCHAR(100),
    @old_last_name      VARCHAR(100),
    @old_nullable_col   VARCHAR(20),
    @old_last_modified  DATETIME,
    @new_first_name     VARCHAR(100),
    @new_last_name      VARCHAR(100),
```



```
    @new_nullable_col   VARCHAR(20),
    @new_last_modified  DATETIME
)
BEGIN
DECLARE @current_last_modified DATETIME;
// Detect a conflict by checking the number of rows that are
// affected by the following update. The WHERE clause compares
// old values uploaded from the remote to current values in
// the consolidated database. If the values match, there is
// no conflict. The COALESCE function returns the first non-
// NULL expression from a list, and is used in this case to
// compare values for a nullable column.


UPDATE my_customer
SET first_name              = @new_first_name,
 last_name                  = @new_last_name,
 nullable_col               = @new_nullable_col,
 last_modified              = @new_last_modified

WHERE cust_1st_pk           = @cust_1st_pk
 AND cust_2nd_pk            = @cust_2nd_pk
 AND first_name             = @old_first_name
 AND last_name              = @old_last_name
 AND COALESCE(nullable_col, '') = COALESCE(@old_nullable_col, '')
 AND last_modified          = @old_last_modified;
...
// Use the @@rowcount global variable to determine
// the number of rows affected by the update. If @@rowcount=0,
// a conflict has occurred. In this example, the database with
// the most recent update wins the conflict. If the consolidated
// database wins the conflict, it retains its current values
// and no action is taken.

IF( @@rowcount = 0 ) THEN
// A conflict has been detected. To resolve it, use business
// logic to determine which values to use, and update the
// consolidated database with the final values.

  SELECT last_modified INTO @current_last_modified
    FROM my_customer WITH( HOLDLOCK )
    WHERE cust_1st_pk=@cust_1st_pk
      AND cust_2nd_pk=@cust_2nd_pk;

    IF( @new_last_modified > @current_last_modified ) THEN
    // The remote has won the conflict: use the values it
    // uploaded.

      UPDATE my_customer
      SET first_name    = @new_first_name,
        last_name     = @new_last_name,
        nullable_col  = @new_nullable_col,
        last_modified = @new_last_modified
      WHERE cust_1st_pk  = @cust_1st_pk
        AND cust_2nd_pk   = @cust_2nd_pk;

    END IF;
END IF;
END;
```

次の項を参照してください。

- 「COALESCE 関数 [その他]」『SQL Anywhere サーバ - SQL リファレンス』
- 「グローバル変数」『SQL Anywhere サーバ - SQL リファレンス』の @@rowcount

強制的な競合解決

強制的な競合解決は、アップロードされたすべてのローに競合があるものとして強制的に処理する特別な方法です。

upload_insert、upload_update、upload_delete スクリプトがすべて未定義の場合、Mobile Link サーバは強制的な競合解決を使用します。この操作モードでは、Mobile Link サーバは、そのテーブルからアップロードされたすべてのローを、upload_old_row_insert スクリプトと upload_new_row_insert スクリプトによって定義された文を使って挿入しようとします。基本的には、アップロードされたすべてのローは競合として処理されます。ストアド・プロシージャまたはスクリプトを作成し、アップロードした値を目的に応じて処理できます。

upload_insert、upload_update、または upload_delete スクリプトがない場合は、通常の競合解決処理は実行されません。この方法の主な使い方は、次の 2 通りです。

- **任意の競合の検出と解決**　自動メカニズムでは、ローの更新時にエラーを検出するだけです。また、それは古い値が統合データベースの現在の値と一致しない場合にだけ行われます。

  upload_old_row_insert スクリプトと upload_new_row_insert スクリプトを使って、アップロードした未加工データを取得し、そのデータが最適になるように処理できます。

- **パフォーマンス**　upload_insert、upload_update、upload_delete が定義されていない場合、Mobile Link サーバが通常行う競合検出タスクは実行されません。このタスクには、1 回につき 1 つのローについての統合データベースへの問い合わせが含まれます。これらのスクリプトを定義しない場合は、upload_old_row_insert スクリプトと upload_new_row_insert スクリプトで定義した文を使って、アップロードしたばかりの情報の挿入だけは行ってください。Mobile Link サーバはネットワークを介してローをフェッチしていないので、パフォーマンスが向上します。

参照

- 「強制的な競合に関する統計情報」 203 ページ

データ・エントリ

一部のデータベースには、データ・エントリ専用のテーブルがあります。この種類のテーブルを処理するには、同期時に挿入されたローをすべてアップロードし、そのローをダウンロードでリモート・データベースから削除する方法があります。同期後に、リモート・テーブルは再び空になり、別のデータ・バッチに使用できます。

このモデルを使用するには、end_upload テーブル・スクリプトを使用して、ローをテンポラリ・テーブルにアップロードしてから、ベース・テーブルに挿入してください。テンポラリ・テーブルを download_delete_cursor スクリプトの中で使用し、同期が正常に完了した後で、リモート・データベースからローを削除できます。

また、STOP SYNCHRONIZATION DELETE 文 (削除内容を次回の同期中にアップロードしないようにする) を使って、クライアント・アプリケーションでもローを削除できます。

「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

削除の処理

ローを統合データベースから削除する場合、そのローを持つすべてのリモート・データベースからも削除できるように、ローの記録が必要です。これを行う 2 つの方法として、論理削除を使用する方法と、シャドー・テーブルを使用する方法があります。

- **論理削除**　この方法では、ローは削除されません。不要になったデータについては、ステータス・カラムで非アクティブのマークが付けられます。download_cursor と download_delete_cursor の WHERE 句では、ローのステータスを参照できます。

  この方法は、CustDB サンプル・アプリケーションの ULEmpCust テーブルで使用されており、テーブルのアクション・カラムには削除を示す "D" が入っています。スクリプトでは、この値を使用してリモート・データベースからレコードを削除し、さらに、同期処理の最後に統合データベースからもレコードを削除します。CustDB ではこの方法を ULOrder テーブルに使用し、Contact サンプルではこの方法を Customer、Contact、Product テーブルに使用しています。

  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. **同期モデル作成ウィザード**の **[削除のダウンロード]** ページで、**[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。

- **シャドー・テーブル**　この方法では、シャドー・テーブルを作成し、削除したローのプライマリ・キー値をそこに格納します。ローが削除されると、1 つのトリガによってシャドー・テーブルが設定されます。download_delete_cursor スクリプトは、シャドー・テーブルを使用して、リモート・データベースからローを削除します。シャドー・テーブルには、実際のテーブルのプライマリ・キー・カラムだけが必要です。

「download_delete_cursor スクリプトの作成」 353 ページを参照してください。

削除同期の一時停止

通常、SQL Anywhere は同期サブスクリプションのあるパブリケーションに属するテーブルまたはカラムへの変更を自動的に記録します。これらの変更は次の同期時に統合データベースにアップロードされます。

しかし、同期対象のデータからローを削除しても、変更がアップロードされないようにする必要がある場合もあります。これを行うには STOP SYNCHRONIZATION DELETE 文を使用します。この機能は、特別な修正のために使用できますが、自動同期機能の一部が無効化されるので、注意して使用してください。この方法は、download_delete_cursor スクリプトを使用して必要なローを削除する処理に代わる、実用的な代替手段です。

STOP SYNCHRONIZATION DELETE 文を実行すると、その接続に対してそれ以降に実施された削除操作は同期されません。この効果は、START SYNCHRONIZATION DELETE 文が実行されるまで継続します。この効果はネストしません。つまり、最初の STOP SYNCHRONIZATION DELETE 文の後に同じ文を実行してもさらなる効果はないということです。

♦ **接続を介して実行された削除のアップロードを一時停止するには、次の手順に従います。**

1. 次の文を発行して削除の自動ロギングを停止します。

   ```
   STOP SYNCHRONIZATION DELETE
   ```

2. 必要に応じて、DELETE 文を使用して同期対象のデータからローを削除します。これまでの変更内容をコミットします。
3. 次の文を使用して削除のログを再開します。

   ```
   START SYNCHRONIZATION DELETE
   ```

削除されたローは Mobile Link サーバに送られないため、統合データベースからは削除されません。

参照

- SQL Anywhere クライアント：「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- Ultra Light クライアント：「Ultra Light STOP SYNCHRONIZATION DELETE 文」『Ultra Light データベース管理とリファレンス』

失敗したダウンロードの処理

ブロッキング・ダウンロード確認の使用

ブロッキング・ダウンロード確認は推奨されなくなりました。可能な場合は常に非ブロッキング・ダウンロード確認を使用してください。

ダウンロード・トランザクションで、ダウンロード内容の書き換え情報を保持します。この情報は、リモート・データベースに適用されたダウンロードの内容を基に、自動的に更新されます。

エラーが発生してダウンロード全体をリモート・データベースに適用できない場合、SendDownloadAck を ON に変更しても、Mobile Link サーバはダウンロードの確認を取得できないため、ダウンロード・トランザクションをロールバックします。トランケーション・ポイントの書き換え情報は、ダウンロード・トランザクションの一部であるため、ロールバックされます。次回ダウンロードが作成される場合、元のトランケーション・ポイントの書き換え情報を使用します。

「SendDownloadACK (sa) 拡張オプション」『Mobile Link - クライアント管理』と「Send Download Acknowledgement 同期パラメータ」『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。

同期スクリプトをテストした時にエラーが発生した場合は、end_download スクリプトを論理に追加してください。これで、失敗に終わったダウンロードを、スクリプトを使って確実に処理できます。

非ブロッキング・ダウンロード確認の使用

非ブロッキング・ダウンロード確認トランザクションで、ダウンロード内容の書き換え情報を保持します。この情報は、リモート・データベースによってダウンロードが正常に適用された後に呼び出される publication_nonblocking_download_ack スクリプトまたは nonblocking_download_ack スクリプトで更新してください。

エラーが発生するか SendDownloadAck が OFF の場合、これらの非ブロッキング・ダウンロード確認スクリプトは呼び出されず、ダウンロード・タイムスタンプは更新されません。同期スクリプトをテストしたときにエラーが発生した場合は、publication_nonblocking_download_ack スクリプトまたは nonblocking_download_ack スクリプトに論理を追加してください。これで、失敗に終わったダウンロードを、スクリプトを使って確実に処理できます。

失敗したダウンロードの再開

ダウンロードの失敗は、ダウンロード中の通信エラーまたはユーザによるダウンロードのキャンセルによって発生します。Mobile Link サーバは、再起動可能なダウンロードで使用するために、クライアントに受信されなかったダウンロード・データを保持します。-ds オプションを使用して、再起動可能なダウンロードに割り当てるデータの最大量を小さくすることで、ダウンロードの失敗が発生する確率を低くすることができます。サーバがダウンロード・データを破棄するのは、次のいずれかの場合です。

- ユーザがダウンロードを正常に完了した場合。

- 再開を有効にしていない状況で、ユーザが新しい同期要求で復帰した場合。
- 受信要求のキャッシュが必要な場合。正常にダウンロードできなかった最も古いデータが、最初にクリアされます。

Mobile Link は、ダウンロードの失敗からのリカバリを支援する機能を備えています。この機能を使用すると、ダウンロード全体の再送を防ぐこともできます。この機能は、SQL Anywhere と Ultra Light リモート・データベースでそれぞれ別に実装されています。「-ds オプション」 64 ページを参照してください。

SQL Anywhere リモート・データベース

ダウンロード中に同期が失敗すると、ダウンロードされたデータはリモート・データベースには適用されません。ただし、正常に送信されたダウンロードの部分は、リモート・デバイスのテンポラリ・ファイルに格納されます。dbmlsync は、このファイルを使用して長時間のデータ再送を防ぎ、ダウンロードの失敗からリカバリします。

この機能を実装するには 3 つの方法があります。どの方法の場合も、dbmlsync はアボートされ、アップロードする新しいデータが存在する場合は再開されたダウンロードは失敗します。

- **-dc**　ダウンロードが失敗した後、次回 dbmlsync を起動するときに、-dc を使用してダウンロードを再開します。失敗したダウンロードの一部が送信されている場合、Mobile Link サーバは、ダウンロードの残りだけを送信します。

  詳細については、「-dc オプション」 『Mobile Link - クライアント管理』を参照してください。

- **ContinueDownload (cd) 拡張オプション**　dbmlsync コマンド・ラインで使用した場合、cd 拡張オプションは -dc オプションと同じように動作します。このオプションは、データベース内に格納したり、sp_hook_dbmlsync_set_extended_options を使用して 1 つの同期内に設定することもできます。

  「ContinueDownload (cd) 拡張オプション」 『Mobile Link - クライアント管理』と「sp_hook_dbmlsync_set_extended_options」 『Mobile Link - クライアント管理』を参照してください。

- **sp_hook_dbmlsync_end hook**　restart パラメータを使用して、ダウンロードを再開できます。restartable download パラメータが true に設定されている場合は、ダウンロードを再開できます。ダウンロード・ファイルが存在し、一定のサイズの場合は、フック内に論理を作成してダウンロードを再開することもできます。

  「sp_hook_dbmlsync_end」 『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

次に示すように、ダウンロードが失敗した後の Ultra Light アプリケーションの動作を制御できます。

- 同期時に Keep Partial Download 同期パラメータを true に設定している場合、ダウンロードが完了する前に失敗すると、Ultra Light はダウンロードされた変更の部分を適用します。また Ultra Light は、Partial Download Retained パラメータを true に設定します。

この時点では、Ultra Light データベースは一貫性のない状態になります。アプリケーションの仕様に応じて、同期を正常に完了するか、データの変更を許可する前にロールバックしてください。「Keep Partial Download 同期パラメータ」『Ultra Light データベース管理とリファレンス』と「Partial Download Retained 同期パラメータ」『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。

- ダウンロードを再開するには、Resume Partial Download 同期パラメータを true に設定し、再度同期を実行します。「Resume Partial Download 同期パラメータ」『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。

  再開された同期はアップロードを実行せず、失敗したダウンロードによってダウンロードされた変更のみをダウンロードします。つまり、失敗したダウンロードは完了しますが、前回の実行以降に行われた変更は同期しません。これらの変更を取得するには、一度失敗したダウンロードが完了してから、再度同期を実行する必要があります。または、Rollback Partial Download を呼び出し、Resume Partial Download を false に設定した状態で同期します。

  ダウンロードを再開すると、失敗した同期から多数の同期パラメータがもう一度自動的に使用されます。たとえば、パブリケーション・パラメータは無視されます。同期は、最初のダウンロードで要求されたパブリケーションをダウンロードします。唯一設定が必要なパラメータは、Resume Partial Download パラメータ (true に設定) と User Name パラメータです。また、以下のパラメータ (設定されている場合) の設定も有効です。

  - Keep Partial Download (今後の中断に備える)
  - DisableConcurrency
  - Observer
  - User Data

- 失敗したダウンロードからの変更を、同期を再開せずにロールバックするには、変更をロールバックする関数を呼び出します。この関数は、embedded SQL の ULRollbackPartialDownload 関数です。Ultra Light コンポーネントの場合は、Connection オブジェクトのメソッドです。

  - **Ultra Light.NET** 「RollbackPartialDownload メソッド」『Ultra Light - .NET プログラミング』を参照してください。
  - **Embedded SQL** 「ULRollbackPartialDownload 関数」『Ultra Light - C/C++ プログラミング』を参照してください。

  サーバやネットワークが使用できないなどの理由で同期が完了できない場合や、エンド・ユーザに作業を続行させながらデータの一貫性を維持したい場合に、失敗したダウンロードから変更をロールバックできます。

通信エラーの詳細については、エラー・メッセージを参照してください。

注意
send_download_ack 同期パラメータが true に設定されている場合、再開されたダウンロードでは設定は無視されます。

ダウンロード確認

リモートでデータが受信されたことを保証する目的では、ダウンロード確認は必要ありません。

ダウンロード確認には、推奨されなくなったブロッキングと、非ブロッキングの 2 つのモードがあります。非ブロッキングがデフォルトです。ブロッキング・ダウンロード確認を使用するとパフォーマンスの低下が発生するため、非ブロッキング・モードをおすすめします。

ダウンロード確認を使用するには、クライアントとサーバの両方に設定があります。

クライアントでは、dbmlsync の拡張オプション SendDownloadACK または Ultra Light の同期パラメータ Send Download Acknowledgement を使用してダウンロード確認を指定します。サーバで変更した設定がない場合、デフォルトでは、非ブロッキング・ダウンロード確認になります。

サーバでは、mlsrv11 の -nba オプションを使用してブロッキング・ダウンロード確認を指定できます。非ブロッキング・ダウンロード確認を使用している場合、統合データベースに最後の正常なダウンロードの時刻を記録するために使用できる接続イベントは 2 つあります。

参照

- 「publication_nonblocking_download_ack 接続イベント」 500 ページ
- 「nonblocking_download_ack 接続イベント」 495 ページ
- 「-nba オプション」 76 ページ
- dbmlsync：「SendDownloadACK (sa) 拡張オプション」 『Mobile Link - クライアント管理』
- Ultra Light：「Send Download Acknowledgement 同期パラメータ」 『Ultra Light データベース管理とリファレンス』

ストアド・プロシージャ・コールからの結果セットのダウンロード

ストアド・プロシージャ・コールから結果セットをダウンロードできます。たとえば、次のテーブルに対する download_cursor があるとします。

```
CREATE TABLE MyTable (
   pk INTEGER PRIMARY KEY NOT NULL,
   col1 VARCHAR(100) NOT NULL,
   col2 VARCHAR(20) NOT NULL
)
```

download_cursor テーブル・スクリプトは次のようになります。

```
SELECT pk, col1, col2
  FROM MyTable
    WHERE last_modified >= {ml s.last_table_download}
    AND employee = {ml s.username}
```

MyTable へのダウンロードに、より高度なビジネス論理を使用する場合は、次のようにスクリプトを作成できます。DownloadMyTable は、2 つのパラメータ (最終ダウンロード・タイムスタンプと Mobile Link ユーザ名) を取り、結果セットを返すストアド・プロシージャです(この例では、移植性のため ODBC 呼び出し規則を使用しています)。

```
{call DownloadMyTable( {ml s.last_table_download}, {ml s.username} )}
```

次に、サポートされる統合データベースごとの簡単な例を示します。詳細については、使用している統合データベースのマニュアルを参照してください。

次の例は、SQL Anywhere、Adaptive Server Enterprise、Microsoft SQL Server に適用されます。

```
CREATE PROCEDURE SPDownload
   @last_dl_ts DATETIME,
   @u_name VARCHAR( 128 )
AS
BEGIN
    SELECT pk, col1, col2
      FROM MyTable
        WHERE last_modified >= @last_dl_ts
        AND employee = @u_name
END
```

次の例は、Oracle に適用されます。Oracle では、パッケージが定義されている必要があります。定義するパッケージには、結果セットのレコード・タイプと、レコード・タイプを返すカーソル・タイプが含まれていなければなりません。

```
Create or replace package SPInfo as
Type SPRec is record (
   pk     integer,
   col1   varchar(100),
   col2   varchar(20)
);
Type SPCursor is ref cursor return SPRec;
End SPInfo;
```

次に、Oracle では、パッケージのカーソル・タイプが最初のパラメータに指定されているストアド・プロシージャが必要です。download_cursor スクリプトは、3 つではなく 2 つのパラメータの

みを渡すことに注意してください。Oracle で結果セットを返すストアド・プロシージャの場合、ストアド・プロシージャ定義でパラメータとして宣言されているカーソル・タイプによって結果セットの構造が定義されますが、真のパラメータ自体は定義されません。この例では、ストアド・プロシージャは Mobile Link システム・テーブルへのスクリプトの追加も行います。

```
Create or replace procedure
   DownloadMyTable( v_spcursor IN OUT SPInfo.SPCursor,
                    v_last_dl_ts IN DATE,
                    v_user_name IN VARCHAR ) As
Begin
   Open v_spcursor For
     select pk, col1, col2
      from MyTable
        where last_modified >= v_last_dl_ts
        and employee = v_user_name;
End;

CALL ml_add_table_script(
 'v1',
 'MyTable',
 'download_cursor',
 '{CALL DownloadMyTable(
    {ml s.last_table_download},{ml s.username} )}'
);
```

次の例は、IBM DB2 LUW に適用されます。

```
CREATE PROCEDURE DownloadMyTable(
  IN last_dl_ts TIMESTAMP,
  IN u_name VARCHAR( 128 ) )
      LANGUAGE SQL
      MODIFIES SQL DATA
      COMMIT ON RETURN NO
      DYNAMIC RESULT SETS 1

      BEGIN
         DECLARE C1, cursor WITH RETURN FOR
            SELECT pk, col1, col2 FROM MyTable
            WHERE last_modified >= last_dl_ts AND employee = u_name;
         OPEN C1;
      END;
```

次の例は、IBM DB2 メインフレームに適用されます。

```
CREATE PROCEDURE DownloadMyTable(
  IN last_dl_ts TIMESTAMP,
  IN u_name VARCHAR( 128 ) )
      LANGUAGE SQL
      MODIFIES SQL DATA
      EXTERNAL NAME MYDMT
      WLM ENVIRONMENT MYWLM
      COMMIT ON RETURN NO
      DYNAMIC RESULT SETS 1
      BEGIN
         DECLARE C1, cursor WITH RETURN FOR
            SELECT pk, col1, col2 FROM MyTable
            WHERE last_modified >= last_dl_ts AND employee = u_name;
         OPEN C1;
      END;
```


自己参照テーブルからのデータのアップロード

一部のテーブルは自己参照します。たとえば、employee テーブルに、従業員をリストするカラムと各従業員のマネージャをリストするカラムが含まれていて、マネージャを管理するマネージャの階層がある場合があります。これらのテーブルを同期するのは、困難である可能性があります。それは、Mobile Link のデフォルト動作ではリモート・データベースでのすべての変更を結合するので、効率的ではあっても、トランザクションの順序が失われるためです。

この状況を処理するには、次の 2 つの方法があります。

- SQL Anywhere リモート・データベースを使用している場合は、dbmlsync -tu オプションを使用して、リモートの各トランザクションが別のトランザクションとして送信されるように指定できます。

  「-tu オプション」 『Mobile Link - クライアント管理』を参照してください。

- マッピング・テーブルを追加して、トランザクションの順序が問題にならないようにします。

Mobile Link 独立性レベル

Mobile Link は、RDBMS で独立性レベルが有効になっている場合、最適な独立性レベルで統合データベースに接続します。データの一貫性を維持しながら最高のパフォーマンスを提供するデフォルトの独立性レベルが選択されます。

通常、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。

スナップショット・アイソレーションを使用すると、重複データがダウンロードされる可能性があります (たとえば、実行時間の長いトランザクションによって同じスナップショットが長時間使用される場合)。しかし、Mobile Link クライアントはこの問題を自動的に処理するので、低下するのはリモート側での転送時間と処理作業量だけです。

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。

独立性レベルは、データベースへの接続が行われた直後に設定されます。さらに他の接続設定が行われてから、トランザクションがコミットされます。独立性レベルと、場合によってはその他の設定を有効にするために、ほとんどの RDBMS で COMMIT が必要です。

SQL Anywhere バージョン 10

SQL Anywhere バージョン 10 はスナップショット・アイソレーションをサポートしています。デフォルトでは、Mobile Link はアップロードには SQL_TXN_READ_COMMITTED 独立性レベルを、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。

SQL Anywhere 統合データベースでスナップショット・アイソレーションを有効にした場合だけ、Mobile Link はスナップショット・アイソレーションを使用できます。スナップショット・アイソレーションが有効でない場合、Mobile Link はデフォルトの SQL_TXN_READ_COMMITTED を使用します。

データベースがスナップショット・アイソレーションを使用できるようにすると、パフォーマンスに影響を与える可能性があります。これは、スナップショット・アイソレーションを使用するトランザクションの数に関係なく、修正されたすべてのローのコピーを保持する必要があるからです。「スナップショット・アイソレーションの有効化」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。

アップロードに対してスナップショット・アイソレーションを有効にするには mlsrv11 -esu オプションを使用し、スナップショット・アイソレーションを無効にするには mlsrv11 -dsd オプションを使用します。接続スクリプトで Mobile Link のデフォルトの独立性レベルを変更する必要がある場合は、begin_upload スクリプトまたは begin_download スクリプトで変更してください。begin_connection スクリプトでデフォルトの独立性レベルを変更すると、アップロードとダウンロード・トランザクションの開始時に設定が上書きされます。

「-esu オプション」 68 ページと「-dsd オプション」 65 ページを参照してください。

バージョン 10 より前の SQL Anywhere

バージョン 10 より前の SQL Anywhere を使用している場合、デフォルトの Mobile Link 独立性レベルは SQL_TXN_READ_COMMITTED です。begin_connection スクリプトで Mobile Link セッション全体のデフォルト値を変更するか、begin_upload スクリプトと begin_download スクリプトでアップロードとダウンロードのデフォルト値をそれぞれ変更できます。

Adaptive Server Enterprise

Adaptive Server Enterprise では、デフォルトの Mobile Link 独立性レベルは SQL_TXN_READ_COMMITTED です。begin_connection スクリプトで Mobile Link セッション全体のデフォルト値を変更するか、begin_upload スクリプトと begin_download スクリプトでアップロードとダウンロードのデフォルト値をそれぞれ変更できます。

Oracle

Oracle はスナップショット・アイソレーションをサポートしていますが、READ COMMITTED と呼ばれています。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。

begin_connection スクリプトで Mobile Link セッション全体のデフォルト値を変更するか、begin_upload スクリプトと begin_download スクリプトでアップロードとダウンロードのデフォルト値をそれぞれ変更できます。

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 に対するパーミッションを持っている必要があります。持っていない場合には、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。SYS だけがこのアクセス権を付与できます。このアクセス権を付与する Oracle の構文は次のとおりです。

```
grant select on SYS.V_$TRANSACTION to user-name
```

Microsoft SQL Server 2005 以降

Microsoft SQL Server 2005 はスナップショット・アイソレーションをサポートしています。デフォルトでは、Mobile Link はアップロードには SQL_TXN_READ_COMMITTED 独立性レベルを、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。

SQL Server 統合データベースでスナップショット・アイソレーションを有効にした場合だけ、Mobile Link はスナップショット・アイソレーションを使用できます。スナップショット・アイソレーションが有効でない場合、Mobile Link はデフォルトの SQL_TXN_READ_COMMITTED を使用します。詳細については、SQL Server のマニュアルを参照してください。

アップロードに対してスナップショット・アイソレーションを有効にするには mlsrv11 -esu オプションを使用し、スナップショット・アイソレーションを無効にするには mlsrv11 -dsd オプションを使用します。接続スクリプトで Mobile Link のデフォルトの独立性レベルを変更する必要がある場合は、begin_upload スクリプトまたは begin_download スクリプトで変更してください。begin_connection スクリプトでデフォルトの独立性レベルを変更すると、アップロードとダウンロード・トランザクションの開始時に設定が上書きされます。

「-esu オプション」 68 ページと「-dsd オプション」 65 ページを参照してください。

SQL Server でスナップショット・アイソレーションを使用するには、Mobile Link サーバをデータベースに接続するのに使用するユーザ ID が、SQL Server システム・テーブル

SYS.DM_TRAN_ACTIVE_TRANSACTIONS にアクセスするパーミッションを持っている必要があります。このパーミッションが与えられていない場合、Mobile Link はデフォルト・レベルの SQL_TXN_READ_COMMITTED を使用します。

統合データベースが、他のデータベースも実行している Microsoft SQL Server で実行している場合、アップロードまたはダウンロードにスナップショット・アイソレーションを使用している場合、およびアップロードまたはダウンロード・スクリプトがサーバ上の他のデータベースにアクセスしていない場合は、Mobile Link サーバの -dt オプションを指定してください。このオプションにより、Mobile Link は、現在のデータベース内のトランザクションを除く、すべてのトランザクションを無視します。これにより、スループットが向上し、ダウンロードされるローの重複が減少します。

「-dt オプション」 66 ページを参照してください。

バージョン 2005 より前の Microsoft SQL Server

バージョン 2005 より前の Microsoft SQL Server を使用している場合、デフォルトの Mobile Link 独立性レベルは SQL_TXN_READ_COMMITTED です。begin_connection スクリプトで Mobile Link セッション全体のデフォルト値を変更するか、begin_upload スクリプトと begin_download スクリプトでアップロードとダウンロードのデフォルト値をそれぞれ変更できます。

参照

- 「-dsd オプション」 65 ページ
- 「-esu オプション」 68 ページ
- 「-dt オプション」 66 ページ
- 「同期処理」 『Mobile Link - クイック・スタート』
- 「独立性レベルと一貫性」 『SQL Anywhere サーバ - SQL の使用法』
- 「スナップショット・アイソレーション」 『SQL Anywhere サーバ - SQL の使用法』

Mobile Link のパフォーマンス

目次

パフォーマンスに関するヒント

次に、Mobile Link の最高のパフォーマンスを引き出すために推奨される事項のリストを示します。

テスト

同期システムのスループットに総合的に影響を与える要素には、さまざまなものがあります。それは、リモート・データベースを実行するデバイスのタイプ、リモート・データベースのスキーマ、リモートのデータ量と同期頻度、ネットワーク特性 (HTTP、プロキシ、Web サーバ、リダイレクタなどの特性)、Mobile Link サーバを実行するハードウェア、同期スクリプト、同時に同期するデータ量、使用する統合データベースのタイプ、統合データベースを実行するハードウェア、統合データベースのスキーマです。

テストは非常に重要です。配備する前に、運用環境で使用するのと同じハードウェアとネットワークを使用してテストを実行してください。また、リモートの数、同期頻度、データ量も同じにしてテストを実行します。テストでは、以下のパフォーマンスのヒントを参考にしてください。

競合の回避

同期スクリプトでは、競合を回避し、同時実行性を最大化します。

たとえば、begin_download スクリプトが、テーブル内のカラムを増分してダウンロードの合計数をカウントするとします。複数のユーザが同時に同期を行う場合、このスクリプトによって効果的にダウンロードが直列化されます。begin_synchronization、end_synchronization、prepare_for_download の各スクリプトでも同じカウンタが有効です。これらのスクリプトはコミットの直前に呼び出され、データベースのロックは短時間ですむからです。

「競合」 179 ページを参照してください。

同期のトランザクション構造については、「同期処理のトランザクション」 『Mobile Link - クイック・スタート』を参照してください。

最適な数のデータベース・ワーカ・スレッドの使用

Mobile Link の -w オプションを使用して、Mobile Link データベース・ワーカ・スレッド数を、最適なスループットが得られる最小値に設定します。使用状況に合った最適な数を見つけるには、実験が必要です。

データベース・ワーカ・スレッドの数が多いほど、統合データベースに同時にアクセスできる同期の数が多くなり、スループットが向上します。

データベース・ワーカ・スレッドの数を小さい値にしておくと、統合データベースでの競合発生回数、統合データベースへの接続数、最適なキャッシュに必要なメモリを少なくすることができます。

次の項を参照してください。

- 「データベース・ワーカ・スレッド数」 179 ページ
- 「-w オプション」 108 ページ
- 「-wu オプション」 109 ページ

SQL Anywhere クライアントの場合、クライアント側のダウンロード・バッファを有効にする

SQL Anywhere クライアントの場合、ダウンロード・バッファにより、クライアントがダウンロードを適用するのを待たずに、Mobile Link データベース・ワーカ・スレッドがダウンロードを転送できます。ダウンロード・バッファはデフォルトで有効になっています。ただし、ダウンロード確認が有効になっている場合は、ダウンロード・バッファを使用できません (次の項目を参照)。

「DownloadBufferSize (dbs) 拡張オプション」『Mobile Link - クライアント管理』を参照してください。

非ブロッキング・ダウンロード確認の使用

デフォルトでは、ダウンロード確認は無効になっています。ただし、ダウンロード確認は、統合データベースでリモート・プログレスを記録するときに便利です。ブロッキング・ダウンロード確認は、リモートがダウンロードを適用しているときにデータベース接続を占有します。ダウンロード確認を使用する場合は、非ブロッキング・ダウンロード確認を使用してください。非ブロッキング・ダウンロード確認がデフォルトです。

「SendDownloadACK (sa) 拡張オプション」『Mobile Link - クライアント管理』と「-nba オプション」 76 ページを参照してください。

不要な BLOB の同期の回避

頻繁に同期されるローに BLOB を含めるのは非効率的です。これを避けるには、BLOB と BLOB ID を含むテーブルを作成し、このテーブル内で同期が必要な ID を参照します。

データベース接続の最大数の設定

Mobile Link データベース接続の最大数を、同期スクリプト・バージョンの数と Mobile Link データベース・ワーカ・スレッド数の積に 1 を加えた数に設定します。これにより、Mobile Link がデータベース接続を閉じたり作成したりする必要が少なくなります。接続の最大数は mlsrv11 -cn オプションで設定します。

「Mobile Link データベース接続」 181 ページと「-cn オプション」 58 ページを参照してください。

十分な物理メモリの確保

Mobile Link サーバを実行しているコンピュータに、他のメモリ要件に加えて、キャッシュを確保する十分な物理メモリがあることを確認してください。

アクティブに処理される同期の数は、データベース・ワーカ・スレッドの数によって制限されません。Mobile Link サーバでは、多数の同期のアップロードのアンパックとダウンロードの送信を同時に行うことができます。最高のパフォーマンスを得るには、同期をディスクにページングしないで処理するために十分なメモリ・キャッシュが Mobile Link サーバにあることが重要です。Mobile Link サーバのメモリ・キャッシュを設定するには、-cm オプションを使用します。

「-cm オプション」 57 ページを参照してください。

十分な処理能力の使用

十分な処理能力を Mobile Link に確保して、Mobile Link サーバの処理がボトルネックにならないようにしてください。通常、Mobile Link サーバで必要とされる CPU 処理能力は、統合データベースの場合に比べて非常に小さくてすみます。ただし、Java または .NET のロー・ハンドリングを使用する場合は、Mobile Link サーバの処理要件が増えます。実際には、ボトルネックの要因になることが多いのは、ネットワークの制限やデータベースの競合です。

最小の冗長ロギングの使用

ビジネス・ニーズに合う最小の冗長ロギングを使用してください。デフォルトでは、冗長ロギングは設定されておらず、Mobile Link はディスクにログを書き込みません。ロギングの冗長性を制御するには、-v オプションを使用します。また、-o オプションか -ot オプションを使用すると、ファイルへのロギングを有効にできます。

冗長ログ・ファイルの代わりに、Mobile Link モニタで同期をモニタできます。Mobile Link モニタは Mobile Link サーバと同じコンピュータ上になくてもかまいません。また、モニタ接続が Mobile Link サーバのパフォーマンスに与える影響はほとんどありません。「Mobile Link モニタ」 183 ページを参照してください。

オペレーティング・システムの制限の考慮

TCP/IP を介してサーバがサポートできる同時接続数は、オペレーティング・システムによって制限されます。1000 を超えるクライアントが同時に同期しようとしたときにこの制限に達する可能性があります。この制限に達すると、オペレーティング・システムで予期しない動作が発生する可能性があります。たとえば、接続が予期せずに終了したり、接続しようとするクライアントが拒否されたりします。この動作を防ぐには、-sm オプションを使用して、オペレーティング・システムの制限よりも少ない数をリモート接続の最大数として指定します。

クライアントは、-sm オプションで指定された同時同期の最大数をすでに受け入れている Mobile Link サーバと同期しようとすると、エラー・コード -85 (SQLE_COMMUNICATIONS_ERROR) を受信します。クライアント・アプリケーションはこのエラーを処理して、数分後に接続を再び試みます。

-sm オプションの詳細については、「-sm オプション」 96 ページを参照してください。

SQLE_COMMUNICATIONS_ERROR の詳細については、「通信エラーが発生しました。」 『エラー・メッセージ』を参照してください。

Java または .NET の同期論理と SQL 同期論理

Java または .NET の同期論理と、SQL 同期論理では、スループットに著しい差は見られません。ただし、Java と .NET の同期論理では、同期ごとに余分なオーバーヘッドが発生し、より多くのメモリが必要です。

さらに、SQL 同期論理は統合データベースが稼働しているコンピュータで実行されますが、Java または .NET の同期論理は Mobile Link サーバが稼働しているコンピュータで実行されます。このため、統合データベースの負荷が高い場合は、Java または .NET の同期論理が適していることがあります。

ダイレクト・ロー・ハンドリングを使用して同期すると、Mobile Link サーバの負荷が高くなるので、ダイレクト・ロー・ハンドリングの実装方法によっては、必要な RAM、ディスク領域、CPU 処理能力が増える可能性があります。

優先同期

一部のテーブルを他のテーブルより高い頻度で同期する必要がある場合は、それらのテーブル用に別のパブリケーションとサブスクリプションを作成します。Sybase Central で同期モデルを使用する場合、これを行うには、複数のモデルを作成します。この優先パブリケーションを他のパブリケーションより頻繁に同期し、他のパブリケーションをピーク時以外の時間に同期できます。

必要なローだけのダウンロード

スナップショットではなくタイムスタンプ同期を使用するなどして、必要なローだけをダウンロードするようにしてください。不要なローのダウンロードは無駄であり、同期のパフォーマンスに悪影響を及ぼします。

スクリプト実行時の最適化

統合データベースのスクリプトを実行するときのパフォーマンスは重要です。テーブルに対してインデックスを作成し、アップロード・カーソル・スクリプトとダウンロード・カーソル・スクリプトによって必要なローだけが効率的に特定されるようにすると便利です。ただし、インデックスが多すぎるとアップロードに時間がかかります。

Sybase Central で**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場合、モデルを展開すると、インデックスがダウンロード・カーソルごとに自動的に定義されます。

大規模なアップロードのロー数の推定

SQL Anywhere クライアントの場合、多数のローをアップロードするときは、アップロードするロー数の推定値を dbmlsync に指定すると、アップロード速度を大幅に改善できます。この操作には、dbmlsync -urc オプションを使用します。

「-urc オプション」『Mobile Link - クライアント管理』を参照してください。

Mobile Link のパフォーマンスに影響を与える主要な要因

Mobile Link 同期のためのスループットなど、システム全体のパフォーマンスは、通常、そのシステムのある時点のボトルネックによって制限されます。Mobile Link 同期では、同期スループットを制限するボトルネックとして次のものが考えられます。

- **統合データベースのパフォーマンス**　Mobile Link で特に重要なのは、統合データベースによる Mobile Link スクリプトの実行速度です。複数のデータベース・ワーカ・スレッドがスクリプトを同時に実行するので、最高のスループットを得るには、同期スクリプトでのデータベース競合を避ける必要があります。
- **Mobile Link のデータベース・ワーカ・スレッド数**　スレッド数が小さければ小さいほど、データベース接続数は少なくなるので、統合データベースでの競合発生率は低くなり、オペレーティング・システムのオーバヘッドは少なくなります。ただし、データベース・ワーカ・スレッドの数があまりに少ないと、ワーカ・スレッドが開放されるまでクライアントが待機したままになったり、統合データベースへの接続数が少ないために、効率的な重複ができなくなったりします。
- **クライアントから Mobile Link への通信の帯域幅**　ダイヤルアップ接続、広域ネットワーク (WAN) や無線ネットワークでの接続など、通信速度の遅い接続では、クライアントと Mobile Link サーバがデータ転送を待たなければならないことがあります。
- **クライアントの処理速度**　ダウンロードには、ローやインデックスの書き込みなど、より多くのクライアント処理が含まれるため、クライアントの処理速度の遅さは、アップロードよりもダウンロードでボトルネックになる可能性が高くなります。
- **Mobile Link から統合データベースへの通信の帯域幅**　Mobile Link と統合データベースが同じコンピュータで稼働している場合や、別々のコンピュータで稼働していても高速ネットワークで接続されている場合には、帯域幅がボトルネックになる可能性は低くなります。
- **Mobile Link サーバを実行しているコンピュータの処理速度**　Mobile Link を実行しているコンピュータの処理能力が低い場合や、Mobile Link データベース・ワーカ・スレッドとバッファに十分なメモリがない場合には、Mobile Link の実行速度が同期のボトルネックになることがあります。バッファとデータベース・ワーカ・スレッドが物理メモリに収まる場合は、Mobile Link サーバのパフォーマンスがディスク速度に左右されることはほとんどありません。

Mobile Link のパフォーマンスのチューニング

Mobile Link 同期のスループットを最大限にするには、複数の同期が同時に発生し、効率よく実行することが重要になります。複数の同時同期を有効にするため、Mobile Link ではさまざまなタスク用にデータベース・ワーカ・スレッドのプールを用意しています。そのうちの 1 つは、ネットワークからのアップロード・データの読み込みとそのアンパック専用です。「データベース・ワーカ・スレッド」と呼ばれる別のスレッド・プールは、統合データベースへのアップロードの適用と統合データベースからダウンロードするデータのフェッチに使用されます。また別のデータベース・ワーカ・スレッドのプールは、ダウンロード・データのパックとリモート・デー

タベースへの送信専用です。各データベース・ワーカ・スレッドは、同期スクリプトを使い、統合データベースへの単一の接続を使用して、変更の適用とフェッチを行います。

競合

最も重要な要因は、同期スクリプトでのデータベース競合を避けることです。複数のクライアントがデータベースを使用する他の場合と同様、クライアントがデータベースに同時にアクセスするときには、データベース競合を最小限にします。同期のたびに修正が必要なデータベースのローがあると、競合の発生率が高くなります。たとえば、あるローのカウンタを増分するスクリプトがある場合、カウンタを更新することがボトルネックになる可能性があります。

同期要求は (-sm オプションで指定された制限数まで) 受け付けられ、アップロードされたデータの読み込みとアンパックが行われて、データベース・ワーカ・スレッドで使用できるようになります。データベース・ワーカ・スレッドの数より同期の数が多い場合は、超過分はキューに追加されて、データベース・ワーカ・スレッドに空きが出るのを待機します。

データベース・ワーカ・スレッド数と接続数は制御できますが、Mobile Link は、1 つのデータベース・ワーカ・スレッドに最低 1 つの接続があることを常に確認します。データベース・ワーカ・スレッドよりも多くの接続がある場合、余分な接続はアイドル状態です。スクリプト・バージョンが複数ある場合には、後述するように、余分な接続が役に立つこともあります。

データベース・ワーカ・スレッド数

同期スクリプトで発生する競合以外に、同期スループットの最も重要な要因になるのがデータベース・ワーカ・スレッド数です。データベース・ワーカ・スレッド数は、統合データベースで同時に進行する同期の数を制御します。

最適なデータベース・ワーカ・スレッド数を判別するにはテストが不可欠です。

データベース・ワーカ・スレッドの数を増やすと、統合データベースにアクセスできる同期の重複を多くしたり、スループットを向上させたりすることができます。その反面、重複する同期の間でリソースやデータベースの競合が増えて、1 つの同期にかかる時間も長くなります。データベース・ワーカ・スレッドの数が増えると、1 つの同期の時間が長くなるというデメリットがあるので、同時同期をより多く実行することが重要になります。また、さらにデータベース・ワーカ・スレッドを追加すると、スループットが低下します。使用する環境に最も適したデータベース・ワーカ・スレッド数を決定するには実験が必要ですが、次の情報も参考になります。

パフォーマンス・テストで、最高のアップロード・スループットは、データベース・ワーカ・スレッド数が比較的少ないときに発生することがわかりました。ほとんどの場合、3 〜 10 のデータベース・ワーカ・スレッドが最適でした。統合データベース、データ容量、データベース・スキーマ、同期スクリプトの複雑さ、使用したハードウェアなどの要因によって結果は変動します。ボトルネックの一般的な原因は、統合データベース内でアップロード・スクリプトの SQL を同時に実行するデータベース・ワーカ・スレッド間の競合です。

ブロッキング・ダウンロード確認を使用するダウンロードに最適なデータベース・ワーカ・スレッド数は、クライアントから Mobile Link への帯域幅とクライアントの処理速度によって異なります。処理速度の遅いクライアントで、最適なダウンロード・パフォーマンスを得るには、より多くのデータベース・ワーカ・スレッドが必要です。これは、アップロードと比較すると、ダウンロードでは、より多くのクライアント処理が必要になる一方、それほど多くの統合データベース処理を必要としないからです。ブロッキング・ダウンロード確認を使用している場合、データベース・ワーカ・スレッドは、リモート・データベースでダウンロードの適用が完了する

までブロックします。非ブロッキング・ダウンロード確認では、ワーカをより多くする必要はありません。

ダウンロード確認を使用していない場合 (デフォルト)、別のスレッドでダウンロードを送信している間に特定のデータベース・ワーカ・スレッドで同期を処理できるので、クライアントから Mobile Link への帯域幅の影響が小さくなります。したがって、データベース・ワーカ・スレッド数は重要ではありません。

データベース・ワーカ・スレッド数よりも多くのダウンロードを同時に送信できます。ダウンロードのパフォーマンスを最適化するには、これらのダウンロードのバッファ用の RAM が Mobile Link サーバに十分にあることが重要です。RAM が十分になかった場合、ダウンロードはディスクにページングされるので、ダウンロードのパフォーマンスが低下する可能性があります。Mobile Link サーバのメモリ・キャッシュ・サイズを指定するには、-cm オプションを使用します。

「-cm オプション」 57 ページを参照してください。

Mobile Link サーバでディスクへのページングが開始されたら (同時に処理されるダウンロードが多すぎる場合など)、-sm オプションを使用して、データベース・ワーカ・スレッドの数を減らすか、アクティブに処理される同期の合計数を制限することを検討してください。

「-sm オプション」 96 ページを参照してください。

ダウンロード確認をオフ (デフォルト) のままにしておくと、ダウンロードに使用できるデータベース・ワーカ・スレッドの最適数が少なくなります。これは、データベース・ワーカ・スレッドがクライアントによるダウンロードの適用を待つ必要がないからです。

「SendDownloadACK (sa) 拡張オプション」『Mobile Link - クライアント管理』を参照してください。

ダウンロード確認を使用する場合、パフォーマンスは、(ブロッキング・ダウンロード確認ではなく) 非ブロッキング・ダウンロード確認を使用した方が優れています。非ブロッキング確認モードでは、リモート・データベースがダウンロードを適用しているときに、サーバはデータベース・ワーカ・スレッドを再利用します。つまり、データベース・ワーカ・スレッドの数を増やす必要がないため、より高いパフォーマンスを得ることができます。

ダウンロードとアップロードの最高のスループットを得るために、Mobile Link には 2 つのオプションが用意されています。1 つは、ダウンロードを最適化するデータベース・ワーカ・スレッドの総数を指定するオプションです。もう 1 つは、アップロードを同時に適用できる数を制限して、アップロードのスループットを最適にするオプションです。

-w オプションは、データベース・ワーカ・スレッドの総数を制御します。デフォルトは 5 です。

-wu オプションは、アップロードを同時に統合データベースに適用できるデータベース・ワーカ・スレッド数を制限します。デフォルトでは、すべてのデータベース・ワーカ・スレッドがアップロードを同時に適用できますが、この場合、統合データベースで重大な競合が発生します。-wu オプションを使用すると、この競合を軽減すると同時に、多数のデータベース・ワーカ・スレッドでダウンロード・データのフェッチを最適化できます。-wu オプションは、その数値がデータベース・ワーカ・スレッドの合計数未満の場合にだけ有効です。

「-w オプション」 108 ページと「-wu オプション」 109 ページを参照してください。

Mobile Link データベース接続

Mobile Link では、データベースの接続はデータベース・ワーカ・スレッドごとに作成されます。-cn オプションを使用すると、Mobile Link に対してデータベース接続のより大きなプールを作成するように指定できますが、Mobile Link が接続を閉じるか、異なるスクリプト・バージョンを使用する必要がないかぎり、余分な接続はアイドル状態です。

Mobile Link がデータベース接続を閉じ、新しい接続を開く場合が 2 つあります。1 つはエラーが発生した場合です。もう 1 つは、クライアントがある同期スクリプト・バージョンを要求し、現在使用されている接続の中にその同期バージョンを利用できる接続がない場合です。

注意
それぞれのデータベース接続はスクリプト・バージョンと関連付けられています。バージョンを変更するには、接続を閉じて再度開いてください。

定常的に複数のスクリプト・バージョンを使用するのであれば、接続数を増やすことで Mobile Link での接続開閉要求を減らすことができます。同期に使用する接続数が、データベース・ワーカ・スレッド数とスクリプト・バージョン数の積になる場合は、接続数を増やす必要はありません。

次のコマンド・ラインには、2 つのスクリプト・バージョンに対して Mobile Link を調整する例が示されています。

```
mlsrv11 -c "dsn=SQL Anywhere 11 Demo" -w 5 -cn 10
```

同期に使用するデータベース接続の最大数は、スクリプト・バージョン数とデータベース・ワーカ・スレッド数の積です。したがって、-cn を 10 に設定することで、データベース接続が不必要に閉じられたり開かれたりすることを排除できます。

「-cn オプション」 58 ページを参照してください。

Mobile Link のパフォーマンスのモニタ

さまざまなツールを使用して、同期のパフォーマンスをモニタできます。

Mobile Link モニタは、同期をモニタするためのグラフィカル・ツールです。同期時のさまざまな処理に要した時間を確認できます。

「Mobile Link モニタ」 183 ページを参照してください。

また、同期をモニタするための Mobile Link スクリプトがいくつかあります。これらのスクリプトにより、ビジネス論理でパフォーマンス統計を使用できます。たとえば、パフォーマンス情報を格納して後で分析したり、同期の実行時間が長すぎる場合に DBA に警告できます。詳細については、次の項を参照してください。

- 「download_statistics 接続イベント」 425 ページ
- 「download_statistics テーブル・イベント」 428 ページ
- 「synchronization_statistics 接続イベント」 511 ページ
- 「synchronization_statistics テーブル・イベント」 514 ページ
- 「time_statistics 接続イベント」 517 ページ
- 「time_statistics テーブル・イベント」 520 ページ
- 「upload_statistics 接続イベント」 538 ページ
- 「upload_statistics テーブル・イベント」 543 ページ

Mobile Link モニタ

目次

Mobile Link モニタの概要

Mobile Link モニタは、同期のパフォーマンスに関する詳細情報を提供する Mobile Link 管理ツールです。

Mobile Link モニタを起動して Mobile Link サーバに接続すると、そのモニタリング・セッションで発生するすべての同期に関する統計情報の収集が開始されます。接続を切断するか、Mobile Link サーバを停止するまで、Mobile Link モニタはデータを収集し続けます。

Mobile Link モニタのインタフェースでは、表形式またはグラフィカル形式でデータを表示できます。データをバイナリ形式で保存して後から Mobile Link モニタに表示したり、*.csv* 形式で保存して Microsoft Excel などの別のツールで開いたりすることもできます。または、Mobile Link がサポートしているリレーショナル・データベースなどの ODBC データ・ソースにエクスポートすることも可能です。

同期に関するさまざまな情報を、Mobile Link モニタの出力で確認できます。たとえば、エラーが発生した同期や、指定したその他の条件に一致する同期をすばやく特定できます。期間の異なる同期においてほぼ同時に終了したフェーズがあるかどうかをチェックすると、(同期は前のフェーズが完了するのを待ってから処理を続行するため) 可能性のある同期スクリプトの競合を特定できます。

特に Mobile Link サーバとは別のコンピュータで Mobile Link モニタを実行する場合は、モニタリングによってパフォーマンスが低下することはないため、Mobile Link モニタは開発環境でも運用環境でも日常的に使用できます。

SQL Anywhere モニタ

SQL Anywhere モニタは、SQL Anywhere データベースや Mobile Link サーバの正常性や可用性に関する情報を示す、ブラウザベースの管理ツールです。システム全体の正常性や可用性を評価するときや、全体的な同期統計情報を分析するときに便利です。SQL Anywhere モニタは個々の同期に関する情報は提供しません。タイミングなどの同期別の統計を含む、個々の同期に関する詳細情報を取得するには、Mobile Link モニタを使用します。

SQL Anywhere モニタの詳細については、「SQL Anywhere モニタ」 『SQL Anywhere サーバ - データベース管理』を参照してください。

Mobile Link モニタの起動

Mobile Link サーバごとに、Mobile Link モニタのインスタンスを複数実行できます。

♦ **データのモニタリングを開始するには、次の手順に従います。**

1. 統合データベースと Mobile Link サーバを起動します (起動していない場合)。
2. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Mobile Link モニタ]** を選択します。

   または、コマンド・プロンプトで **mlmon** と入力することもできます。詳細については、以下を参照してください。
3. Mobile Link モニタ接続は、Mobile Link サーバへの同期接続と同じように開始されます。たとえば、-zu+ を指定して Mobile Link サーバを起動した場合は、ここで指定するユーザ ID が何であっても構いません。すべての Mobile Link モニタ・セッションに対して、スクリプト・バージョンが for_MLMonitor_only に設定されます。

   [Mobile Link サーバへの接続] ウィンドウで次の情報を入力してください。

   - **[ホスト]** Mobile Link サーバが稼働しているコンピュータのネットワーク名または IP アドレス。デフォルトでは、Mobile Link モニタが稼働しているコンピュータです。Mobile Link サーバが Mobile Link モニタと同じコンピュータで稼働している場合は、**localhost** を使用できます。
   - **[プロトコル]** Mobile Link サーバが同期要求に使用しているのと同じネットワーク・プロトコルとポートに設定してください。
   - **[ポート]** Mobile Link サーバが同期要求に使用しているのと同じネットワーク・ポートに設定してください。
   - **[暗号化]** プロトコルとして HTTPS または TLS を選択すると、このボックスが有効になります。暗号化のタイプをドロップダウン・リストから選択します。
   - **[追加のプロトコルオプション]** オプションのパラメータを指定します。次のパラメータを設定できます。複数のパラメータを指定する必要がある場合は、セミコロンで区切ります。
     - **buffer_size**=*number*
     - **client_port**=*nnnn*
     - **client_port**=*nnnn-mmmmm*
     - **persistent**={**0**|**1**} (HTTP と HTTPS のみ)
     - **proxy_host**=*proxy_hostname* (HTTP と HTTPS のみ)
     - **proxy_port**=*proxy_portnumber* (HTTP と HTTPS のみ)
     - **url_suffix**=*suffix* (HTTP と HTTPS のみ)
     - **version**=*HTTP-version-number* (HTTP と HTTPS のみ)

   「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』を参照してください。

4. 同期を開始します。

   収集されたデータが Mobile Link モニタに表示されます。

コマンド・ラインでの mlmon の起動

コマンド・ライン・オプションを使用すると、Mobile Link モニタでファイルを開いたり、起動時に Mobile Link サーバへ接続することができます。次の構文を使用します。

mlmon [*connect-options* | *inputfile*.{ **mlm** | **csv** }]

文中の各項目を次に説明します。

connect-options　これは、次のうちの 1 つまたは複数です。

- **-u** *ml_username* (Mobile Link サーバへの接続に必要)
- **-p** *password*
- **-x** { **tcpip** | **tls** | **http** | **https** } [**(** *keyword=value*;... **)**]

  上記で説明している keyword=value のペアは、ホスト、プロトコル、追加のネットワーク・パラメータです。-x オプションは、Mobile Link サーバへの接続に必要です。
- **-o** *outputfile*.{ **mlm** | **csv** }

  -o オプションは、接続の最後で Mobile Link モニタを閉じ、指定したファイルにセッションを保存します。

mlmon -? と入力すると、mlmon の構文が表示されます。

UNIX での Mobile Link モニタの起動

Linux のデスクトップ・アイコンをサポートする Linux バージョンを使用していて、SQL Anywhere 11 のインストール時にこれらのアイコンをインストールするように選択した場合は、次の手順を使用できます。

♦ Mobile Link モニタを起動するには、次の手順に従います (Linux デスクトップ・アイコンの場合)。

1. **[アプリケーション] - [SQL Anywhere 11] - [Mobile Link モニタ]** を選択します。
2. 「データのモニタリングを開始するには、次の手順に従います。」に示した要領で、Mobile Link サーバに接続するための情報を入力します。

> **注意**
> 以降の手順は、SQL Anywhere ユーティリティのソースを指定済みであることを前提としています。「UNIX と Mac OS X での環境変数の設定」『SQL Anywhere サーバ - データベース管理』を参照してください。

♦ Mobile Link モニタを起動するには、次の手順に従います (UNIX コマンド・ラインの場合)。

1. ターミナル・セッションで次のコマンドを入力します。


```
mlmon
```

2. Mobile Link サーバに接続するための情報を前述の要領で入力します。

Mobile Link モニタの停止

♦ **Mobile Link モニタを停止するには、次の手順に従います。**

1. Mobile Link モニタで、**[モニタ] - [Mobile Link サーバからの切断]** を選択します。データの収集が停止します。

   Mobile Link サーバをシャットダウンするか、Mobile Link モニタを終了して、データの収集を停止することもできます。

   Mobile Link モニタを終了する前に、そのセッションのデータを保存できます。「Mobile Link モニタのデータの保存」 197 ページを参照してください。

2. Mobile Link モニタを終了する準備ができたら、**[ファイル] - [閉じる]** を選択します。

Mobile Link モニタの使用

Mobile Link モニタには次のウィンドウ枠があります。

- **[詳細テーブル]** **[詳細テーブル]** は一番上のウィンドウ枠です。これは、各同期の合計所要時間と、同期の各部分に要した時間を示すスプレッドシートです。

  「[詳細テーブル] ウィンドウ枠」 188 ページを参照してください。

- **[使用率グラフ]** **[使用率グラフ]** は 2 番目のウィンドウ枠です。ここには、Mobile Link サーバのさまざまなキューの長さがグラフィック表示されます。**[使用率グラフ]** ウィンドウ枠と **[チャート]** ウィンドウ枠で同じ目盛りが使用されます。**[チャート]** ウィンドウ枠の下の目盛りは時間を表します。下の **[概要]** ウィンドウ枠でデータをドラッグして選択するか、**[ビュー]** - **[移動]** を選択して、[使用率グラフ] ウィンドウ枠に表示されているデータを選択できます。

  「[使用率グラフ] ウィンドウ枠」 190 ページを参照してください。

- **[チャート]** **[チャート]** は 3 番目のウィンドウ枠です。ここには、同期がグラフィック表示されます。このウィンドウ枠の下の目盛りは時間を表します。下の **[概要]** ウィンドウ枠でデータをドラッグして選択するか、**[ビュー]** - **[移動]** を選択して、[チャート] ウィンドウ枠に表示されているデータを選択できます。

  「[チャート] ウィンドウ枠」 192 ページを参照してください。

- **[概要]** **[概要]** は一番下のウィンドウ枠です。ここには、セッション内のすべての同期の概要が表示されます。このウィンドウ枠には**マーキー・ツール**と呼ばれるボックスの枠があり、**[チャート]** および **[使用率グラフ]** ウィンドウ枠に表示されるデータを選択できます。

  「[概要] ウィンドウ枠」 193 ページを参照してください。

また、表示をカスタマイズするための **[オプション]** ウィンドウと、詳細な情報を確認するための [プロパティ] ウィンドウがあります。次の項を参照してください。

- 「[オプション] ウィンドウ」 194 ページ
- 「セッション・プロパティ」 195 ページ
- 「サンプル・プロパティ」 195 ページ
- 「同期プロパティ」 196 ページ

[詳細テーブル] ウィンドウ枠

[詳細テーブル] には、同期の各部分の所要時間に関する情報が表示されます。時間はすべて Mobile Link サーバによって測定されます。対応するスクリプトが定義されていなくても、所要時間が 0 ではない場合があります。

[ツール] - **[オプション]** を開き、次に **[テーブル]** タブを開くと、**[詳細テーブル]** ウィンドウ枠に表示されるカラムを選択できます。使用可能な統計の説明については、「Mobile Link の統計のプロパティ」 200 ページを参照してください。

デフォルトでは、次のカラムが表示されます。

- **sync**　各同期を識別します。この ID は、Mobile Link モニタではなく Mobile Link サーバによって割り当てられるので、どの Mobile Link モニタ・セッションでも、1 から始まるとは限らず、続き番号になるとも限りません。同じ ID を **[同期プロパティ]** ウィンドウで見ることができます。「同期プロパティ」 196 ページを参照してください。
- **remote_id**　リモート・データベースの ID。
- **user**　同期ユーザ。
- **version**　同期スクリプトのバージョン。

  「スクリプト・バージョン」 341 ページを参照してください。
- **download_ack**　ダウンロード確認のタイプ (なし、ブロッキング、または非ブロッキング)。
- **start_time**　Mobile Link サーバが同期を開始した日時(これは、クライアントが同期を要求した時点よりも遅い場合があります)。
- **duration**　同期の合計時間 (秒)。
- **sync_request**　Mobile Link がクライアントからアップロードされたデータを受信するのに要する時間 (秒)。これは、リモートからの接続で開始し、認証の直前に終了する同期の部分です。
- **receive_upload**　リモート・データベースと Mobile Link サーバの間のネットワーク接続を確立してから、アップロード・ストリームの最初のバイトを受信するまでの時間。-sm を -nc よりも小さい値に設定した場合以外はこの時間はわずかです。-sm を -nc よりも小さい値に設定した場合は、-sm で指定したアクティブな同期の最大数よりも同期の数が多かった場合に同期が一時停止した時間が、この時間に含まれる可能性があります。
- **get_db_worker**　空いているデータベース・ワーカ・スレッドを取得するために必要な時間。
- **connect**　新しいデータベース接続が必要な場合に、データベース・ワーカ・スレッドでデータベース接続を確立するために必要な時間。たとえば、エラーの後や、スクリプトのバージョンが変更された場合に新しい接続が必要になります。
- **authenticate_user**　認証を必要とする同期設定の場合に、Mobile Link が同期要求、ユーザ名、パスワードを検証するのに要する時間 (秒)。これは、ユーザ認証トランザクション (認証の開始から begin_synchronization イベントの直前まで) の長さです。
- **begin_sync**　begin_synchronization スクリプトが実行されている場合、実行に要する時間 (秒)。
- **apply_upload**　アップロードを統合データベースに適用するのに要する時間 (秒)。これは、begin_upload スクリプトから end_upload スクリプトまでの時間です。
- **prepare_for_download**　prepare_for_download スクリプトが実行されている場合、実行に要する時間 (秒)。
- **fetch_download**　データのダウンロードに要する時間 (秒)。これは、begin_download スクリプトから end_download スクリプトまでの時間です。ダウンロード確認が有効になっている場合は、ダウンロードをリモート・データベースに適用し、確認を返す時間が含まれます。
- **end_sync**　end_synchronization スクリプトが実行されている場合、実行に要する時間 (秒)。

テーブルを特定のカラムでソートするには、カラムの見出しをクリックします。Mobile Link モニタに表示される新しいデータがある場合は、ソートされて追加されます。

[ビュー] メニューで **[詳細テーブル]** オプションのチェックを外すことで、**[詳細テーブル]** ウィンドウ枠を閉じることができます。

[使用率グラフ] ウィンドウ枠

[使用率グラフ] は、上から 2 番目のウィンドウ枠です。ここには、いくつかのタイプのワーク・キューに関するサーバ統計情報が表示されます。

このウィンドウ枠に表示されるデータの詳細については、「使用率グラフの使用」 190 ページを参照してください。

[使用率グラフ] ウィンドウ枠では、**[チャート]** ウィンドウ枠と同じ水平スクロールバー、垂直時間ラベル、ズーム・レベルが使用されます。つまり、**[使用率グラフ]** ウィンドウ枠と **[チャート]** ウィンドウ枠で同じ時間帯が上下に並びます。

グラフに表示されているデータを選択する方法は複数あります。

- **[ビュー]** - **[移動]** を選択します。
- **[概要]** ウィンドウ枠で、**マーキー・ツール**を動かします。**マーキー・ツール**とは、**[概要]** ウィンドウ枠に表示される小さなボックスです。「[概要] ウィンドウ枠」 194 ページを参照してください。

[使用率グラフ] の領域内をダブルクリックすると、**[サンプル・プロパティ]** ウィンドウが開き、表示されているサンプル間隔の詳細が表示されます。サンプル間隔は約 1 秒です。「サンプル・プロパティ」 195 ページを参照してください。

注意
プロパティのリストは、Mobile Link サーバまたは開いた *.mlm* ファイルから取得され、Mobile Link サーバの言語が使用されています。Mobile Link モニタで別の言語を使用している場合は、一部の文字が正常に表示されない可能性があります。

使用率グラフの使用

[使用率グラフ] の値を確認する場合、また出力をカスタマイズする場合は、**[ツール]** - **[オプション]** を選択して、**[グラフ]** タブを開きます。このタブには、**[使用率グラフ]** のキューが色分けして表示されます。また、ここでグラフをカスタマイズすることもできます。

プロパティ

- **[TCP/IP ワーク・キュー]** Mobile Link サーバの低レベル・ネットワーク・レイヤで実行される処理を表します。このレイヤでは、ネットワークに対するパケットの読み込みと書き込みを行います。このキューには、読み込み/書き込み要求が大量に追加されます。ネットワークから読み込む必要がある受信データが通知されたり、ストリーム・ワーカによりネットワークへの書き込みを指示されたりすると、このキューが大きくなります。

このキューが滞るのは、通常は読み込みまたは書き込みの未処理が原因です。読み込みと書き込みの両方の場合もありますが、通常はいずれか一方です。読み込みが滞るのは、サーバで大量の RAM が使用されていて、メモリ・ページのスワップ・インとスワップ・アウトが頻繁に行われている場合です。この場合は、RAM を増やすことを検討してください。書き込みが滞るのは、クライアントとサーバ間のネットワーク接続の速度が遅い場合です。このキューだけが滞る場合は、CPU の使用率を確認してください。CPU の使用率が高い場合は、読み込みまたはメモリに問題がないかどうかを確認します。CPU の使用率が低い場合は、書き込み速度が遅い可能性があります。より高速のネットワークを使用することを検討してください。

- **[ストリーム・ワーク・キュー]** バージョン 10 クライアント専用です。Mobile Link サーバの高レベル・ネットワーク・レイヤで実行される処理を表します。このレイヤでは、HTTP、暗号化、圧縮など、高レベル・ネットワーク・プロトコル処理を行います。TCP/IP からの読み込みが多かったり、コマンド・プロセッサ・レイヤからの書き込み要求が多かったりすると、このキューが大きくなります。このキューだけが滞る場合は、HTTP や圧縮など、一部のネットワーク・プロトコルを削除することを検討してください。削除できない場合は、-sm オプションを使用して、許可する同時同期数を減らすことを検討してください。

  「-sm オプション」 96 ページを参照してください。

- **[ハートビート・ワーク・キュー]** サーバ内でパルス・イベントを送信する Mobile Link サーバのレイヤを表します。このレイヤでは、たとえば、接続している Mobile Link モニタへの 1 秒に 1 回のサンプル・パルスをトリガします。

  このキューは、滞る可能性がほとんどないので、デフォルトで非表示になっています。

- **[コマンド・プロセッサ・ワーク・キュー]** 内部 Mobile Link プロトコル・コマンドを解釈し、さらにそのコマンドを統合データベースに適用するために、Mobile Link サーバによって実行される処理を表します。このキューは、多数の要求を受け取ったときに大きくなります。要求のタイプには、同期要求、リスナ要求、mlfiletransfer 要求などがあります。このキューは、統合データベースで同期の処理が行われているときに、さらに同期の要求を受け取ったときにも大きくなります。

  このキューだけが滞る場合は、CPU の使用率を確認してください。CPU の使用率が高い場合は、要求数が多すぎる可能性があります。-sm オプションを使用して、許可する同時同期数を減らすことを検討してください。CPU の使用率が低い場合は、統合データベースのパフォーマンスに向上の余地がないかどうかを検討してください。

  「-sm オプション」 96 ページを参照してください。

- **[ビジーなデータベース・ワーカ・スレッド]** この値は、Mobile Link サーバによる統合データベースの使用量を示します。この値の 1 単位は、データベースで何らかの処理を実行しているデータベース・ワーカ・スレッドを表します。挿入、更新、削除、選択の区別はありません。この値が 0 の場合、Mobile Link サーバは統合データベースで処理を実行していません。

  このカウントが大きい場合 (mlsrv11 -w オプションで設定した最大値に近い場合)、Mobile Link サーバは統合データベースに高い負荷をかけています。そのような場合でも、スループットが悪くなければ、何もする必要はありません。スループットに満足できない場合は、-w オプションを使用してデータベース・ワーカ・スレッドの数を増やすことを検討してください。その際は、-w 値を大きくすると、接続間の競合が増大することに注意してください。

すべての接続でアップロードが実行されると状況が特に悪化するので、mlsrv11 -wu オプションを使用して、アップロードを実行するデータベース・ワーカに低めの制限を設ける必要が生じます。適切なスループットを実現する -w と -wu の設定値が見つからない場合は、同期スクリプトを検証して、競合の原因がないかどうかを検討してください。最終的には、RDBMS のマニュアルを参照して、統合データベースの全体的なパフォーマンスを向上させる方法がないかどうかを検討してください。

位取り

このカラムは、各プロパティの現在の位取りを示します。

[使用率グラフ] の縦軸の位取りは常に 0 ～ 100 です。これは全体の 0 ～ 100 パーセントを表します。位取りは値ごとに設定します。デフォルトでは、すべての位取りは 5 です。これは、値が 0 ～ 20 の範囲にあり、拡大 (5 倍) すると 0 ～ 100 の範囲になることを示しています。値が 20 を超えると、最大値が 100 になるように位取りが自動的に調整されます。

表示の最大値は、この位取りで 100 を割ることで確認できます。たとえば、TCP/IP ワークキューの位取りが 2.381 の場合、最大値は (100 / 2.381) = 42 になります。最大値がいくつであるかは通常は重要ではありません。重要なのは、現在のモニタリング・セッションで見られるように、グラフの上部に向かっているその値がそのプロパティの現在の最大値、つまりそのプロパティの最大負荷に近づいているということです。

グラフが定常的に上部に表示されており、同期スループットが低下している場合、調査が必要な問題がパフォーマンスに関して発生している可能性があります。同様に、1 つ以上の値がグラフ上部に長期間にわたって停滞している場合も、パフォーマンスに問題がある可能性があります。一方、Mobile Link サーバのパフォーマンスが良好なときも、グラフは頻繁に表示上部へ向かいます。これは、単に Mobile Link サーバがビジー状態であり、処理が正常に行われていることを示します。

アンチエイリアス処理

カスタマイズできるものの 1 つは、アンチエイリアス処理です。アンチエイリアス処理により、グラフの見栄えがよくなりますが、描画に時間がかかることもあります。

[チャート] ウィンドウ枠

[チャート] ウィンドウ枠には、**[詳細テーブル]** と同じ情報がグラフィックで表示されます。**[チャート]** 内のバーは、各同期で要した時間を表し、バーの各部分は同期のフェーズを表します。

データの表示

[詳細テーブル] で同期をクリックすると、その同期が選択されます。

同期をダブルクリックすると、**[同期セッション・プロパティ]** ウィンドウが開きます。「同期プロパティ」 196 ページを参照してください。

リモート ID 別またはコンパクトにデータをグループ化

データをユーザ別にグループ化できます。**[ビュー]** - **[リモート ID]** を選択します。

また、データをコンパクト・モードで表示すると、すべてのアクティブな同期ができるだけ少ない行数で表示されます。**[ビュー]** - **[コンパクト・ビュー]** を選択します。**[コンパクト・ビュー]** では、ロー番号は無意味になります。

データの拡大

[チャート] ウィンドウ枠に表示されているデータを選択するには、次の 4 つの方法があります。

- **ズーム・オプション**　**[ビュー]** メニューのズーム・オプションとツールバーのズーム・ボタンを使用して、ズーム・インやズーム・アウトができます。同期によって使用可能な領域を埋めるには、**[選択範囲にズーム]** を使用します。
- **スクロールバー**　**[チャート]** ウィンドウ枠の下のスクロールバーをクリックしてスライドさせます。
- **[移動] ウィンドウ**　このウィンドウを開くには、**[ビュー]** - **[移動]** を選択します。

  [開始日時] では、**[チャート]** ウィンドウ枠に表示するデータの開始時刻を指定できます。この設定を変更する場合は、少なくとも日付／時刻の年月日を指定してください。

  [チャートの範囲] では、表示する期間を指定できます。チャート範囲は、ミリ秒、秒、分、時間、または日数で指定できます。チャート範囲によって、データの細分性が決定されます。時間の長さが短いほど、より詳細に表示されます。
- **マーキー・ツール**　**[概要]** ウィンドウ枠で、**マーキー・ツール**を動かします。**マーキー・ツール**とは、**[概要]** ウィンドウ枠に表示される小さなボックスです。「[概要] ウィンドウ枠」 194 ページを参照してください。

時間軸

[チャート] ウィンドウ枠の下には、期間を示す目盛りがあります。時間のフォーマットは、表示される時間の間隔に応じて自動的に再調整されます。目盛り上にカーソルを置くと、いつでも完全な日付／時刻を表示できます。

デフォルトの色スキーム

[オプション] ウィンドウ (**[ツール]** メニューから開く) では、**[チャート]** ウィンドウ枠の色を表示または設定できます。**[チャート]** ウィンドウ枠の各フェーズには、アップロードにライム・グリーン、ダウンロードにコーラル・レッド、開始と終了に青の色スキームがデフォルトで使用されます。各フェーズの以前の部分は暗い影で表します。

色の設定については、「[オプション] ウィンドウ」 194 ページを参照してください。

[概要] ウィンドウ枠

[概要] ウィンドウ枠には、Mobile Link モニタのセッション全体の概要が表示されます。セッション間は**マーキー・ツール**を使用してナビゲートできます。マーキー・ツールは、**[概要]** ウィンドウ枠内のボックスです。

アクティブ同期、完了同期、および失敗同期が色分けされて表示されます。色を設定するには、Mobile Link モニタを開き、**[ツール]** - **[オプション]** を選択し、**[概要]** タブをクリックします。

「[オプション] ウィンドウ」 194 ページを参照してください。

[概要] ウィンドウ枠を閉じるには、**[ビュー]** メニューで **[概要]** のチェックをオフにします。

[概要] ウィンドウ枠は、[Mobile Link モニタ] ウィンドウの他の部分から切り離すこともできます。**[オプション]** ウィンドウで、**[概要]** タブを開き、**[概要のウィンドウをメイン・ウィンドウに組み込む]** チェックボックスをオフにします。

マーキー・ツール

マーキー・ツールとは、**[概要]** ウィンドウ枠に表示される小さなボックスです。**マーキー・ツール**を使用すると、さまざまなデータを表示したり、データをさまざまな詳細度で表示できます。このボックス内の領域は、[チャート] ウィンドウ枠と [使用率グラフ] ウィンドウ枠に表示されます。**マーキー・ツール**は次のように使用できます。

- **[概要]** ウィンドウ枠をクリックして、**マーキー・ツール**を動かします。これにより、[チャート] および [使用率グラフ] に表示されるデータの開始時刻が変わります。
- **[概要]** ウィンドウ枠内でドラッグして**マーキー・ツール**を描画し直すことで、**マーキー・ツール**の場所とサイズを変更します。これにより、データの開始時刻と範囲が変わります。マーキー・ボックスを小さくすると、[チャート] に表示されるデータの間隔が短くなり、より詳細なデータが表示されます。

♦ **マーキー・ツールの色を変更するには、次の手順に従います。**

1. **[ツール]** - **[オプション]** を選択します。
2. **[概要]** タブをクリックします。
3. **[マーキー]** フィールドで新しい色を選択します。
4. **[OK]** をクリックします。

[オプション] ウィンドウ

[オプション] ウィンドウでは、**[チャート]** ウィンドウ枠と **[概要]** ウィンドウ枠のグラフィック表示の色とパターンなど、多くの設定を指定できます。

[オプション] ウィンドウを開くには、Mobile Link モニタを開き、**[ツール]** - **[オプション]** を選択します。

デフォルトのリストア

デフォルト設定をリストアするには、ファイル *mlMonitorSettings11* を削除します。このファイルは、ユーザ・プロファイル・ディレクトリにあります。

セッション・プロパティ

[セッション・プロパティ] ウィンドウには、セッションに関する統計が表示されます。ここには、Mobile Link モニタが開始されてからの期間全体についてのプロパティ値が表示されます。**[セッション・プロパティ]** ウィンドウを開くには、Mobile Link モニタを開き、**[ファイル]** - **[プロパティ]** を選択します。

[一般] タブには、セッションに関する基本情報が表示されます。

[統計情報] タブには、**[サンプル・プロパティ]** と同じ統計情報が表示されます。「サンプル・プロパティ」 195 ページを参照してください。

サンプル・プロパティ

[サンプル・プロパティ] ウィンドウには、時間間隔に関する詳細情報が表示されます。各時間間隔は約 1 秒間です。サンプルには、Mobile Link モニタによって、受け取った順序で番号が付けられます。

グラフの外観をカスタマイズして、プロパティを非表示にすることはできますが、**[サンプル・プロパティ]** ウィンドウにはすべてのプロパティが表示されます。非表示にしたプロパティは、**[サンプル・プロパティ]** ウィンドウでは **[非表示]** と示されます。そうでない場合は、色が表示されます。

[サンプル・プロパティ] ウィンドウを開くには、**[グラフ使用率]** ウィンドウ枠内で検証する期間の部分をクリックします。

[サンプル] タブには、指定した 1 秒間に関する情報が表示されます。表示されるプロパティは、指定した時間間隔の終了時の値です。

[範囲] タブには、プロパティ・ウィンドウを開いたときに表示されていたサンプルの範囲 (**[概要]** に表示される水平範囲) 全体の平均が表示されます。範囲の統計情報は、**[範囲]** タブの **[計算]** をクリックしたときに計算されます。

[サンプル・プロパティ] に表示される情報は次のとおりです。

- **[サンプル]**　サンプルには、Mobile Link モニタによって、受け取った順序で番号が付けられます。**[サンプル]** タブの場合は、サンプル番号が表示されます。**[範囲]** タブの場合は、サンプルの範囲が表示されます。
- **[起動時刻] と [終了時刻]**　**[サンプル]** タブの場合は、約 1 秒間のサンプル期間が表示されます。
- **[統計情報] - [色] カラム**　そのプロパティの表示用にグラフで使用されている色が示されます。
- **[統計情報] - [プロパティ] カラム**　サンプルのキューの長さ、または該当範囲に対する平均キューの長さが表示されます。このカラムに表示されるプロパティのタイプは次のとおりです。
  - **[TCP/IP ワーク・キュー]**　ここには、ネットワークからの読み込みによって追加されるのを待機しているバッファ数と、ネットワークに書き込まれるのを待機しているバッファ数

との和がリストされます。具体的な値はあまり重要ではありませんが、値が大きい場合はネットワーク関連のボトルネックを示していることがあります。

- **[ストリーム・ワーク・キュー]** バージョン 10 クライアント専用です。Mobile Link サーバの高レベル・ネットワーク・レイヤで実行される処理を表します。このレイヤでは、HTTP、暗号化、圧縮など、高レベル・ネットワーク・プロトコル処理を行います。
- **[ハートビート・ワーク・キュー]** 同期以外の定期的な内部 Mobile Link サーバ・タスクに関するキューの長さです。
- **[コマンド・プロセッサ・ワーク・キュー]** データベース・タスクの実行と Mobile Link クライアントとの通信の解釈または準備に関するキューの長さです。具体的な値はあまり重要ではありませんが、値が大きい場合はデータベース関連のボトルネックを示していることがあります。
- **[ビジーなデータベース・ワーカ・スレッド]** この値は、Mobile Link サーバによる統合データベースの使用量を示します。この値の 1 単位は、データベースで何らかの処理を実行しているデータベース・ワーカ・スレッドを表します。挿入、更新、削除、選択の区別はありません。この値が 0 の場合、Mobile Link サーバは統合データベースで処理を実行していません。

注意：プロパティ名は Mobile Link サーバまたは .mlm ファイルから取得され、Mobile Link サーバの言語が使用されています。Mobile Link モニタで別の言語を使用している場合は、一部の文字が正常に表示されない可能性があります。

- **[統計情報] - [値] カラム** プロパティの値です。
- **[統計情報] - [制限] カラム** プロパティに許可されている最大値です。これにより、すべてのプロパティについてグラフを 0 ～ 100% で使用できて、便利です。ページ・フォールトなどのプロパティには本質的に限りがないので、この制限は無視されます。

同期プロパティ

[詳細テーブル] ウィンドウ枠または **[チャート]** ウィンドウ枠内で同期をダブルクリックすると、その同期のプロパティが表示されます。

すべてのテーブルの統計 (その同期のすべてのテーブルの合計) か、個々のテーブルに関する統計を表示できます。ドロップダウン・リストには、その同期に関係したテーブルのリストが表示されます。

[同期プロパティ] ウィンドウのいずれかのページに表示される数量の詳細については、**[ヘルプ]** をクリックしてください。

[同期プロパティ] ウィンドウの統計値の説明については、「Mobile Link の統計のプロパティ」 200 ページを参照してください。

Mobile Link モニタのデータの保存

Mobile Link モニタのセッションのデータは、バイナリ・ファイル (.mlm) または値をカンマで区切ったテキスト・ファイル (.csv) として保存するか、リレーショナル・データベースのテーブルまたは Microsoft Excel ファイルとして保存できます。

ファイルへの保存

データをファイルとして保存するには、**[ファイル] - [名前を付けて保存]** を選択します。

- 保存したデータを Mobile Link モニタで表示する場合は、データをバイナリ (.mlm) ファイルとして保存します。ファイルを再び開くには、**[ファイル] - [開く]** を選択します。モニタされた情報をすべて保持できるのは、バイナリ・ファイル形式だけです。
- 保存したデータを Microsoft Excel などの別のツールで表示する場合は、カンマで区切ったファイル (*.csv*) として保存します。この場合に保存されるのは、テーブルごとの情報とセッションの終了時刻を除いた、同期のプロパティ・ウィンドウの情報だけです。.csv ファイルを Mobile Link モニタで開くこともできます。

  .csv ファイル・フォーマットでは、時間がミリ秒数で格納されます。

データが自動的にファイルに保存されるように指定できます。これを行うには、**[ツール] - [オプション]** を選択し、**[一般]** タブで出力ファイル名を入力します。出力ファイルが新しいデータで上書きされます。

リレーショナル・データベースまたは Excel へのエクスポート

Mobile Link モニタのデータは、ODBC 接続を使用してエクスポートすることもできます。Mobile Link によってサポートされている任意のリレーショナル・データベース、または Microsoft Excel スプレッドシートにエクスポートできます。

データをエクスポートすると、Mobile Link モニタ・セッションのすべてのカラムがエクスポートされます。さらに、エクスポートを実行した時間を識別する export_time というカラムもエクスポートされます。グラフのデータはエクスポートされません。

カラムのいくつかは予約語のため、データ・ソースは引用符付き識別子が有効になっている必要があります。Mobile Link モニタは、SQL Anywhere、Adaptive Server Enterprise、Microsoft SQL Server の各データベースに対する引用符付き識別子を、自動的に有効にします。引用符付き識別子オプションが有効になっていないと、エクスポートは失敗します。

♦ データベースまたは Excel にデータをエクスポートするには、次の手順に従います。

1. Mobile Link モニタ情報を収集した後、Mobile Link サーバから切断します。
2. Mobile Link モニタで、**[ファイル] - [データベースへのエクスポート]** を選択します。
3. 出力のオプションを選択します。
   - データを保存するために作成するテーブルの名前を 2 つ指定します。指定しない場合は、デフォルトが使用されます。そのテーブルが存在しない場合は、Mobile Link モニタによって作成されます。Excel 出力では、2 つのテーブル名は作成する 2 つのワークシートを識別します。

- 既存のテーブルにあるデータを上書きするかどうかを選択します。データを上書きしない場合は、新しいデータが既存のデータに追加されます。

4. **[OK]** をクリックして **[接続]** ウィンドウを開き、データベースに接続するか、ODBC を使用して Excel スプレッドシートに接続します。

統計のカスタマイズ

ウォッチ・マネージャを使用すると、指定した基準を満たす同期を視覚的に区別できます。たとえば、大規模な同期、長時間の同期、長時間を要する小規模な同期、または警告を受信した同期を強調表示できます。

ウォッチ・マネージャを開くには、Mobile Link モニタを開き、**[ツール]** - **[ウォッチ・マネージャ]** をクリックします。

ウォッチ・マネージャの左ウィンドウ枠には、使用可能なすべてのウォッチのリストが表示されます。右ウィンドウ枠には、アクティブなウォッチのリストが表示されます。ウォッチをアクティブ・リストに追加したり、アクティブ・リストから削除するには、左ウィンドウ枠でウォッチを選択し、該当のボタンをクリックします。

事前に定義されたウォッチが 3 つ (**[アクティブ]**、**[完了]**、**[失敗]**) あります。事前に定義されたウォッチを編集して、表示方法を変更できます。また、右ウィンドウ枠からこれらのウォッチを削除すると、非アクティブにできます。

[チャート] には、ウォッチの条件を満たしていない同期は表示されません。すべてのウォッチを無効に (**[現在のウォッチ]** リストから削除) すると、**[チャート]** ウィンドウ枠または **[概要]** ウィンドウ枠に同期が表示されなくなります。

右ウィンドウ枠でのウォッチの順序は重要です。リストの上にあるウォッチから先に処理されます。**[上へ移動]** ボタンと **[下へ移動]** ボタンを使用して、右ウィンドウ枠でのウォッチの順序を編成できます。

事前に定義されたウォッチを使用して、別のウォッチを作成できます。ウォッチの条件を編集するには、古い条件を削除してから新しいウォッチ条件を追加します。

新しい Mobile Link モニタが同じ Mobile Link サーバに接続すると、すでに接続されている Mobile Link モニタには短時間同期として示されます。Mobile Link モニタ同期のバージョン名は for_ML_Monitor_only になります。このウォッチ付き Mobile Link モニタ同期は非表示にできます。

♦ **新しいウォッチを作成するには、次の手順に従います。**

1. **ウォッチ・マネージャ**で **[新規]** をクリックします。
2. **[名前]** ボックスにウォッチの名前を入力します。
3. **プロパティ**、**比較演算子**、**値**を選択します。

   プロパティの完全なリストについては、「Mobile Link の統計のプロパティ」 200 ページを参照してください。
4. **[追加]** をクリックします (設定を保存するために **[追加]** をクリックしてください)。
5. 必要に応じて、別の**プロパティ**、**演算子**、**値**を選択して、**[追加]** をクリックします。
6. **[チャート]** ウィンドウ枠で、ウォッチの**チャート・パターン**を選択します
7. **[概要]** ウィンドウ枠で、ウォッチの**概要の色**を選択します

Mobile Link の統計のプロパティ

次に、Mobile Link モニタで使用できるプロパティのリストを示します。これらの統計は、**[新規ウォッチ]** ウィンドウ、**[詳細テーブル]** ウィンドウ枠、**[同期プロパティ]** ウィンドウで表示できます。**[同期プロパティ]** では、プロパティ名にアンダースコアが付いていません。

[新規ウォッチ] ウィンドウの詳細については、「統計のカスタマイズ」 199 ページを参照してください。

[詳細テーブル] の詳細については、「[詳細テーブル] ウィンドウ枠」 188 ページを参照してください。

[同期プロパティ] ウィンドウの詳細については、「同期プロパティ」 196 ページを参照してください。

同期の統計情報

強制的な競合モードを使用していない場合、Mobile Link の統計情報に関するプロパティには次の同期情報を返します。

強制的な競合モードの詳細については、「強制的な競合に関する統計情報」 203 ページを参照してください。

プロパティ	説明
active	同期が進行中の場合は true。
apply_upload	アップロードしたデータを統合データベースに適用するために必要な時間。
authenticate_user	ユーザ認証の実行にかかった時間の合計。authenticate_* イベントの実行時間も含まれます。
begin_sync	begin_synchronization イベント時間の合計。
completed	同期が正常に完了した場合は true。
conflicted_deletes	常に 0 (ゼロ) です。
conflicted_inserts	常に 0 (ゼロ) です。
conflicted_updates	競合を引き起こした更新ローの数。対処する競合解決スクリプトが正常に呼び出された場合のみ、ローは含まれます。
connect	新しいデータベース接続が必要な場合に、データベース・ワーカ・スレッドでデータベース接続を確立するために必要な時間。たとえば、エラーの後や、スクリプトのバージョンが変更された場合に新しい接続が必要になります。

プロパティ	説明
connect_for_download_ack	新しいデータベース接続が必要な場合に、データベース・ワーカ・スレッドでデータベース接続を確立するために必要な時間。
connection_retries	Mobile Link サーバが統合データベースへの接続をリトライした回数。
download_bytes	ダウンロードを保存するために Mobile Link サーバ内で使用されるメモリの容量。同期のサーバ・メモリへの影響が反映されます。
download_deleted_rows	Mobile Link サーバによって (download_delete_cursor スクリプトを使用して) 統合データベースからフェッチされたロー削除の数。
download_errors	ダウンロード中に発生したエラーの数。
download_fetched_rows	Mobile Link サーバによって (download_cursor スクリプトを使用して) 統合データベースからフェッチされたローの数。
download_filtered_rows	クライアントがアップロードしたローと一致したため、Mobile Link クライアントにダウンロードされなかったフェッチ済みローの数。
download_warnings	ダウンロード中に発生した警告の数。
duration	Mobile Link サーバが測定した同期時間の合計。
end_sync	end_synchronization イベント時間の合計。
fetch_download	統合データベースからダウンロードするローをフェッチしてダウンロード・ストリームを作成するために必要な時間。
get_db_worker	空いているデータベース・ワーカ・スレッドを取得するために必要な時間。
get_db_worker_for_download_ack	ダウンロード確認を受信してから、データベース・ワーカ・スレッドが空くのを待つ時間。
ignored_deletes	handle_error または handle_odbc_error が定義されており、1000 が返された場合、または指定のテーブルに対して upload_delete スクリプトが定義されていない場合において、upload_delete スクリプトを呼び出したときにエラーを発生させたアップロード削除ローの数。

プロパティ	説明
ignored_inserts	無視されたアップロード挿入ローの合計数。無視される理由は次のとおりです。1) 通常モードで upload_insert スクリプトないか、または強制的な競合モードで upload_new_row_insert スクリプトがない。2) Mobile Link サーバが対応するスクリプトを呼び出し中にエラーが発生し、handle_error または handle_odbc_error イベントが 1000 を返した。
ignored_updates	競合を引き起こしたが、競合解決スクリプトが正常に呼び出されていなかったり、upload_update スクリプトが定義されていなかったりした、アップロード更新ローの数。
nonblocking_download_ack	publication_nonblocking_download_ack 接続と nonblocking_download_ack 接続のイベントに必要な時間。
prepare_for_download	prepare_for_download イベント時間の合計。
remote_id	リモート・データベースをユニークに識別するリモート ID。
send_download	ダウンロード・ストリームをリモート・データベースに送信するために必要な時間。時間は、ダウンロード・ストリームのサイズと、転送用のネットワーク帯域幅によって異なります。アップロード専用同期の場合、ダウンロード・ストリームは単なるアップロード確認です。
start_time	同期開始の日付／時刻 (ISO-8601 拡張フォーマット)。
sync	Mobile Link モニタ・セッション内で同期をユニークに識別する数。
sync_deadlocks	同期で検出された統合データベース内のデッドロックの数。
sync_errors	同期で発生したエラーの合計数。
sync_request	リモート・データベースと Mobile Link サーバの間のネットワーク接続を確立してから、アップロード・ストリームの最初のバイトを受信するまでの時間。
sync_tables	同期に関係したクライアント・テーブルの数。
sync_warnings	同期で発生した警告の数。

プロパティ	説明
upload_bytes	アップロードを保存するために Mobile Link サーバ内で使用されるメモリの容量。同期のサーバ・メモリへの影響が反映されます。
upload_deadlocks	アップロード中に検出された統合データベース内のデッドロック数。
upload_deleted_rows	統合データベースから正常に削除されたローの数。
upload_errors	アップロード中に発生したエラーの数。
upload_inserted_rows	統合データベースに正常に挿入されたローの数。
upload_updated_rows	統合データベースで正常に更新されたローの数。
upload_warnings	アップロード中に発生した警告の数。
user	Mobile Link ユーザ名。
version	同期バージョンの名前。
wait_for_download_ack	ダウンロードがリモート・データベースに適用され、リモート・データベースからダウンロード確認が送信されるのを待つ時間。

強制的な競合に関する統計情報

強制的な競合モードの場合、Mobile Link 統計情報プロパティには次の情報が返されます。

統計プロパティ	説明
conflicted_deletes	upload_old_row_insert スクリプトを使用して統合データベースに正常に挿入されたアップロード削除ローの数。
conflicted_inserts	upload_new_row_insert スクリプトを使用して統合データベースに正常に挿入されたアップロード挿入ローの数。
conflicted_updates	upload_new_row_insert または upload_old_row_insert スクリプトを使用して正常に適用されたアップロード更新ローの数。
ignored_deletes	handle_error または handle_odbc_error が定義されており、1000 が返された場合、または指定のテーブルに対して upload_old_row_insert スクリプトが定義されていない場合において、upload_old_row_insert スクリプトを呼び出したときにエラーを発生させたアップロード削除ローの数。

統計プロパティ	説明
ignored_inserts	handle_error または handle_odbc_error が定義されており、1000 が返された場合、または指定のテーブルに対して upload_new_row_insert スクリプトが定義されていない場合において、upload_new_row_insert スクリプトを呼び出したときにエラーを発生させたアップロード挿入ローの数。
ignored_updates	handle_error または handle_odbc_error が定義されており、1000 が返された場合、または指定のテーブルに対して upload_new_row_insert スクリプトと upload_old_row_insert スクリプトが定義されていない場合において、upload_new_row_insert スクリプトまたは upload_old_row_insert スクリプトを呼び出したときにエラーを発生させたアップロード更新ローの数。
upload_deleted_rows	常に 0 (ゼロ) です。
upload_inserted_rows	常に 0 (ゼロ) です。
upload_updated_rows	常に 0 (ゼロ) です。

Mobile Link 用 SQL Anywhere モニタ

目次

SQL Anywhere モニタの概要

SQL Anywhere モニタは、SQL Anywhere データベースや Mobile Link サーバの正常性や可用性に関する情報を示す Web ブラウザベースの管理ツールで、単に「モニタ」とも呼ばれます。

この章では、モニタを使用して Mobile Link サーバに関するメトリックを収集する方法について説明します。SQL Anywhere データベースにモニタを使用する方法については、「SQL Anywhere モニタ」『SQL Anywhere サーバ - データベース管理』を参照してください。

モニタには次の機能があります。

- **データの常時収集**　SQL Anywhere 11 に対応している他の多くの管理ツールとは異なり、モニタはユーザが Web ブラウザでログインしていない時も含めてメトリックを常時収集します。また、シャットダウンされるまでメトリックの収集を続けます。
- **電子メールによる警告の通知**　メトリックを収集すると、収集したメトリックを検証し、Mobile Link サーバの異常を示す状況を検出した場合に、警告の電子メールを送信します。
- **ブラウザベースのインタフェース**　モニタは Web ブラウザを使用していつでも接続可能で、収集された警告やメトリックを表示できます。
- **複数のデータベースや Mobile Link サーバのモニタリング**　1 つのツールで、同一または異なるコンピュータで実行されている SQL Anywhere データベースと Mobile Link サーバを同時にモニタリングできます。

  SQL Anywhere データベースのモニタリングについては、「SQL Anywhere モニタ」『SQL Anywhere サーバ - データベース管理』を参照してください。
- **パフォーマンスへの影響を最小化**　モニタリングによってパフォーマンスが低下することはないため、開発環境や運用環境でモニタを日常的に使用できます。

前提条件

- オペレーティング・システムに対応している Adobe Flash Player の最新バージョンがインストールされていること (推奨)。モニタには、Adobe Flash Player のバージョン 9 との下位互換性があります。適切なバージョンを確認するには、http://www.adobe.com/jp/products/flashplayer/systemreqs/ にアクセスしてください。
- Web ブラウザで JavaScript が有効になっていること。
- SQL Anywhere 11.0.1 がインストールされていること。

運用環境でのモニタの実行

モニタは、モニタリング対象のリソースとは別のコンピュータにインストールして実行することができます。このようにすると、その後の SQL Anywhere のアップグレードやアップデートのときにモニタのリソースと構成が上書きされません。モニタを運用環境で使用する場合は、別のコンピュータにインストールすることをおすすめします。「別のコンピュータへの SQL Anywhere モニタのインストール」『SQL Anywhere サーバ - データベース管理』を参照してください。

制限事項

- モニタを使用してメトリックを収集できる SQL Anywhere データベースと Mobile Link サーバは次のとおりです。
  - SQL Anywhere 9.0.2、10.0.0、10.0.1、11.0.0、11.0.1
  - 最初の EBF 以降が適用された Mobile Link 11.0.0 と 11.0.1
- 1 台のコンピュータで実行できるモニタは 1 つだけです。
- 個々の同期に関する情報は提供しません。タイミングなどの同期別の統計を含む、個々の同期に関する詳細情報を取得するには、Mobile Link モニタを使用します。「Mobile Link モニタの概要」 184 ページを参照してください。

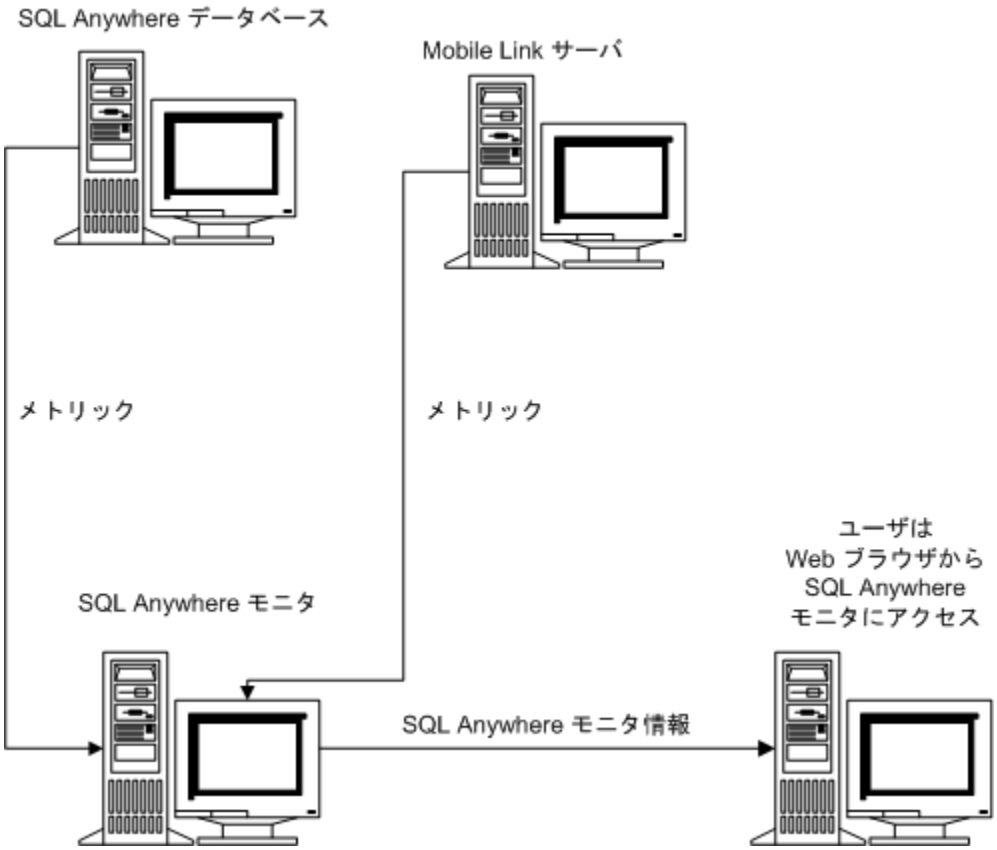
参照

Mobile Link サーバに対して使用できる他の管理ツールやパフォーマンス・ツールの詳細については、次の項を参照してください。

- 「Mobile Link モニタ」 183 ページ

モニタのアーキテクチャ

モニタは、他のコンピュータ上で実行されている SQL Anywhere データベースや Mobile Link サーバからメトリックやパフォーマンス・データを収集します。その間、別のコンピュータが Web ブラウザ経由でモニタにアクセスします。



モニタは、次のタスクを担当するユーザを、そのユーザが DBA かどうかに関係なく支援するよう設計されています。

- Mobile Link サーバがネットワークに接続されていることを確認する。
- Mobile Link サーバに十分なディスク領域またはメモリがあることを確認する。
- 所定の期間内に Mobile Link サーバが実行した同期の回数を確認する。

参照

- 「モニタのクイック・スタート」 209 ページ

モニタのクイック・スタート

Mobile Link サーバ・モニタリングの設定手順は次のとおりです。

1. SQL Anywhere 11.0.1 をネットワークに常時接続されるコンピュータにインストールします。モニタは、SQL Anywhere を使用して Mobile Link サーバをモニタリングします。

   モニタはモニタリング対象のリソースと同じコンピュータ上で実行できますが、特に運用環境では、Mobile Link サーバまたはその他のアプリケーションへの影響を最小限に抑えるために、モニタを別のコンピュータ上で実行することをおすすめします。

2. Web ブラウザに Adobe Flash Player の適切なバージョンがインストールされていること、および JavaScript が有効になっていることを確認します。「前提条件」 206 ページを参照してください。

3. Mobile Link サーバを起動します (実行されていない場合)。

4. モニタを起動し、Web ブラウザで開きます。「モニタの起動」 216 ページを参照してください。

   モニタへのアクセスに Web ブラウザを使用するコンピュータは、モニタが実行されているネットワークに接続されている必要があります。

5. 管理者としてログインします。デフォルトのユーザ名は **admin**、デフォルトのパスワードも **admin** です。

6. **[管理]** タブをクリックし、Mobile Link サーバをモニタリング対象のリソースとして追加します。「リソースの追加」 228 ページを参照してください。

7. 新規ユーザを追加し、admin ユーザのパスワードを変更します。「モニタ・ユーザの作成」『SQL Anywhere サーバ - データベース管理』を参照してください。

8. モニタリングする Mobile Link サーバ用の警告を設定します。「警告」 『SQL Anywhere サーバ - データベース管理』を参照してください。

9. **[モニタ]** タブをクリックし、目的の Mobile Link サーバについて収集されたメトリックを表示します。「リソースのモニタリング」 220 ページを参照してください。

チュートリアル：モニタの使用

このチュートリアルを参考にして、Mobile Link 同期サーバのサンプルのモニタリングを設定してください。

レッスン 1：モニタの起動

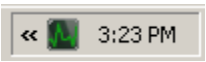
♦ **モニタを起動して開くには、次の手順に従います。**

1. モニタを起動します。**[スタート] - [プログラム] - [SQL Anywhere 11] - [SQL Anywhere モニタ] - [SQL Anywhere モニタ]** を選択します。

   モニタを別のコンピュータにインストールした場合、この手順を実行する必要はありません。モニタが SQL Anywhere とは別のコンピュータにインストールされている場合、モニタはサービスとして実行され、コンピュータの起動時に自動的に起動されます。

2. データを参照します。この手順は、モニタが別のコンピュータにインストールされているかどうかによって異なります。

   システム・トレイで [SQL Anywhere モニタ] アイコンをクリックし、**[データの参照]** を選択します。

   モニタが別のコンピュータにインストールされている場合は、**[スタート] - [プログラム] - [SQL Anywhere モニタ 11] - [データの参照]** を選択します。システム・トレイにアイコンは表示されません。

   Web ブラウザを開き、*http://localhost:4950* にアクセスする方法もあります。

[モニタ] タブの上ウィンドウ枠には、モニタリングされているリソースがリストされます。モニタを初めて開いたときは、自身のみがモニタリングされています。

参照

- 「レッスン 2：Mobile Link サーバをモニタリングするための設定」 211 ページ

レッスン 2：Mobile Link サーバをモニタリングするための設定

モニタは、データベースや Mobile Link サーバからメトリックを収集します。この項では、Mobile Link 同期サーバのサンプルを起動し、このサーバをモニタリング対象のリソースとして追加します。SQL Anywhere データベースのメトリックを収集する方法については、「レッスン 2：データベースをモニタリングするための設定」 『SQL Anywhere サーバ - データベース管理』を参照してください。

♦ **モニタリング対象のリソースを追加するには、次の手順に従います。**

1. Mobile Link 同期サーバのサンプルを起動します。

   [スタート] - [プログラム] - [SQL Anywhere 11] - [Mobile Link] - [同期サーバのサンプル] を選択します。

2. モニタのデフォルトの管理者としてログインします。

   a. **[ログイン]** をクリックします。

b. **[ユーザ名]** フィールドに **admin** と入力し、**[パスワード]** フィールドに **admin** と入力します。

c. **[ログイン]** をクリックします。

3. **[管理]** タブをクリックします。
4. **[リソース]** タブをクリックします。
5. **[追加]** をクリックします。
6. **[Mobile Link サーバ]** を選択して **[次へ]** をクリックします。
7. リソース名に **MobiLinkServerSample** と指定し、**[次へ]** をクリックします。
8. **[ホスト]** フィールドに **localhost** と入力し、**[次へ]** をクリックします。
9. 必要な認証情報が要求されたら、**[ユーザ ID]** フィールドに user1 などのユーザ名を入力し、**[パスワード]** フィールドに sql などのパスワードを入力します。

   これらのクレデンシャルは、Mobile Link サーバへの接続に使用されます。ユーザ ID とパスワードはモニタによって保持されます。
10. **[作成]** をクリックします。
11. 新しいリソース MobiLinkServerSample が作成され、モニタリングが開始されます。
12. **[OK]** をクリックします。
13. **[モニタ]** タブをクリックします。

    MobiLinkServerSample リソースが **[モニタ]** タブに表示され、収集されたメトリックが下ウィンドウ枠のタブに表示されます。

レッスン 3：警告のテスト

このレッスンでは、警告の処理を予行練習するために、警告を意図的にトリガします。

♦ **警告を表示および解決するには、次の手順に従います。**

1. 警告をトリガします。トリガするには、Mobile Link 同期サーバのサンプルをシャットダウンします。

   a. Windows の場合は、システム・トレイで Mobile Link サーバの Mobile Link サーバのアイコンをダブルクリックします。

   b. [Mobile Link サーバ] ウィンドウで **[シャットダウン]** をクリックします。

   c. **[はい]** をクリックします。
2. モニタの **[モニタ]** タブをクリックします。

   MobiLinkServerSample リソースの **[状態]** が **[サーバ停止]** に変わり、MobiLinkServerSample リソースの **[モニタの状態]** が **[注意が必要 !]** に変わります。

   状態やモニタの状態が変わるまで数秒かかることがあります。デフォルトでは、リソースの情報は 30 秒ごとに収集されます。

3. 下ウィンドウ枠で **[警告]** をクリックします。
4. **[使用できません]** を選択し、**[詳細]** をクリックして説明を読みます。
5. **[OK]** をクリックします。
6. 同期サーバのサンプルを再起動します。

   [スタート] - [プログラム] - [SQL Anywhere 11] - [Mobile Link] - [同期サーバのサンプル] を選択します。

   MobiLinkServerSample リソースの **[状態]** が **[実行中]** に変わりますが、**[モニタの状態]** は変わりません。変わるまで数分かかることがあります。
7. 警告を削除します。削除するには、目的の警告を選択し、**[削除]** をクリックします。

   [モニタの状態] が **[正常]** に変わります。

レッスン 4：警告発生時に電子メールが送信されるようにするための設定

発生した警告は、**[モニタ]** タブの下ウィンドウ枠の **[警告]** タブに必ず表示されます。次の手順では、警告が発生したら必ず電子メールが送信されるようにモニタを設定します。

♦ **電子メールによる通知を設定するには、次の手順に従います。**

1. 電子メールを受信するユーザを作成します。
   a. **[管理]** タブをクリックします。
   b. **[ユーザ]** タブをクリックします。
   c. **[新規]** をクリックします。
   d. **[ユーザ名]** フィールドに 「JoeSmith」 と入力します。
   e. **[パスワード]** フィールドと **[パスワードの確認]** フィールドに、**sql** と入力します。
   f. **[電子メール]** フィールドに、有効な電子メール・アドレスを入力します。
   g. **[使用する言語]** フィールドで **[日本語]** を選択します。
   h. **[ユーザのタイプ]** で **[オペレータ]** を選択します。

      オペレータは、電子メールで警告を受信したり、警告の解決と削除を実行できます。このユーザは **[モニタ]** タブにアクセスできますが、**[管理]** タブにはアクセスできません。

      ユーザのタイプの詳細については、「モニタ・ユーザの操作」 235 ページを参照してください。
   i. **[保存]** をクリックします。

      新しいユーザが作成されます。
2. このユーザを MobiLinkServerSample リソースに関連付けます。
   a. **[リソース]** タブをクリックします。

b. **MobiLinkServerSample** リソースを選択し、**[設定]** をクリックします。
c. **[リソースの設定]** ウィンドウで **[オペレータ]** をクリックします。
d. **[選択可能なオペレータ]** リストで 「[JoeSmith]」 を選択して、**[追加]** をクリックします。
e. **[保存]** をクリックします。
f. **[OK]** をクリックします。

3. 電子メールによる警告の通知を設定します。
a. **[管理]** タブをクリックします。
b. **[設定]** タブをクリックします。
c. **[編集]** をクリックします。
d. **[警告の通知を電子メールで送信]** を選択します。
e. その他の設定を必要に応じて指定します。
f. 電子メールによる通知が適切に設定されたかどうかをテストします。
[テスト電子メールを送信] をクリックします。
g. 画面の要求に従い、テスト電子メールの送信先電子メール・アドレスを入力して、**[OK]** をクリックします。
テスト電子メールが、指定された電子メール・アドレスに送信されます。
h. **[保存]** をクリックします。

警告が発生すると、指定されたユーザ宛に、その警告に関する情報が記載された電子メールが送信されます。警告の設定方法の詳細については、「レッスン 3：警告のテスト」 212 ページを参照してください。

レッスン 5：クリーンアップ

次の手順では、MobiLinkServerSample リソースを削除します。これにより、収集されたメトリックが削除され、データの収集が停止します。Mobile Link サーバのモニタリングを継続する必要がある運用環境では、Mobile Link サーバとモニタをどちらも実行させたままにします。

♦ **モニタリングを停止するには、次の手順に従います。**

1. MobiLinkServerSample リソースを削除します。
a. **[管理]** タブをクリックします。
b. **[リソース]** タブをクリックします。
c. MobiLinkServerSample リソースを選択し、**[停止]** をクリックします。
d. **[削除]** をクリックします。
e. **[はい]** をクリックして、リソースの削除を確定します。
2. モニタからログアウトします。

[ログアウト] をクリックします。

3. モニタを表示していた Web ブラウザ・ウィンドウを閉じます。
4. モニタを終了します。

   システム・トレイで [SQL Anywhere モニタ] アイコンをクリックし、**[SQL Anywhere モニタの終了]** を選択します。
5. Mobile Link サーバを停止します。
   a. システム・トレイで Mobile Link 同期サーバのサンプルの [Mobile Link サーバ] アイコンをダブルクリックします。
   b. Mobile Link サーバ・メッセージ・ウィンドウで **[シャットダウン]** をクリックします。
   c. **[はい]** をクリックします。

モニタの起動

モニタを起動すると、モニタに指定されている**すべて**のリソースについてメトリックの収集が開始されます。

モニタの起動手順は、モニタを別のコンピュータで実行しているかどうかによって異なります。

♦ **モニタを起動するには、次の手順に従います。**

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [SQL Anywhere モニタ] - [SQL Anywhere モニタの起動]** を選択します。

   [SQL Anywhere モニタ] アイコンがシステム・トレイに表示されます。

2. モニタに接続します。「モニタへの接続」 218 ページを参照してください。

♦ **別のコンピュータにインストールされたモニタを起動するには、次の手順に従います。**

1. モニタが別のコンピュータにインストールされている場合、モニタはサービスとして自動的に実行されます。ただし、モニタリングを停止した場合は再起動することもできます。再起動するには、*install-dir¥bin32* に移動します。

2. Windows の場合は、次のコマンドを実行します。

   ```
   samonitor.bat start service
   ```

   Linux の場合は、次のコマンドを実行します。

   ```
   samonitor.sh start service
   ```

   モニタがサービスとして実行されている場合、システム・トレイに [SQL Anywhere モニタ] アイコンは表示されません。

3. モニタに接続します。「モニタへの接続」 218 ページを参照してください。

参照

- 「モニタの終了」 『SQL Anywhere サーバ - データベース管理』
- 「モニタへの接続」 『SQL Anywhere サーバ - データベース管理』
- 「モニタの切断」 『SQL Anywhere サーバ - データベース管理』
- 「リソースのモニタリング」 『SQL Anywhere サーバ - データベース管理』

モニタの終了

モニタを終了すると、すべてのリソースについてメトリックの収集が停止されます。Web ブラウザを閉じるだけにして、モニタの実行は継続することをおすすめします。特定の Mobile Link サーバのモニタリングを停止する方法については、「リソースのモニタリングの停止」 231 ページを参照してください。

モニタの終了手順は、モニタを別のコンピュータで実行しているかどうかによって異なります。

♦ **モニタを終了するには、次の手順に従います。**

- システム・トレイで [SQL Anywhere モニタ] アイコンをクリックし、**[SQL Anywhere モニタの終了]** を選択します。

♦ **別のコンピュータにインストールされたモニタを終了するには、次の手順に従います。**

1. *install-dir¥bin32* に移動します。
2. Windows の場合は、次のコマンドを実行します。

   ```
   samonitor.bat stop service
   ```

   Linux の場合は、次のコマンドを実行します。

   ```
   samonitor.sh stop service
   ```

参照

- 「モニタの起動」 『SQL Anywhere サーバ - データベース管理』
- 「モニタへの接続」 『SQL Anywhere サーバ - データベース管理』
- 「モニタの切断」 『SQL Anywhere サーバ - データベース管理』
- 「リソースのモニタリング」 『SQL Anywhere サーバ - データベース管理』

モニタへの接続

モニタへの接続に使用するコンピュータは、モニタが実行されているネットワークに接続されている必要があります。

♦ **モニタに接続するには、次の手順に従います。**

1. モニタがまだ実行されていない場合は、起動します。「モニタの起動」 216 ページを参照してください。
2. データを参照します。この手順は、モニタが別のコンピュータにインストールされているかどうかによって異なります。

   [スタート] - [プログラム] - [SQL Anywhere 11] - [SQL Anywhere モニタ] - [データの参照] を選択します。

   モニタが別のコンピュータにインストールされている場合は、**[スタート] - [プログラム] - [SQL Anywhere モニタ 11] - [データの参照]** を選択します。

   モニタに接続する際のデフォルト URL である **http://***computer-name***:4950** が Web ブラウザで開きます。*computer-name* は、モニタが実行されているコンピュータの名前です。たとえば、*http://localhost:4950* のようになります。
3. 要求された場合は、モニタのユーザ名とパスワードを入力します。モニタのユーザ ID とパスワードでは、大文字と小文字が区別されます。「モニタ・ユーザの操作」 235 ページを参照してください。

参照

- 「モニタの起動」 『SQL Anywhere サーバ - データベース管理』
- 「モニタの終了」 『SQL Anywhere サーバ - データベース管理』
- 「モニタの切断」 『SQL Anywhere サーバ - データベース管理』
- 「リソースのモニタリング」 『SQL Anywhere サーバ - データベース管理』

モニタの切断

モニタから切断するには、ログアウトするか、Web ブラウザを閉じます。

モニタから切断しても、メトリックの収集には影響しません。メトリックの収集を停止する場合は、リソースのモニタリングを停止するか、モニタを終了します。「リソースのモニタリングの停止」 231 ページまたは「モニタの終了」 217 ページを参照してください。

♦ **モニタを切断するには、次の手順に従います。**

1. **[ログアウト]** をクリックします。

参照

- 「モニタの起動」 『SQL Anywhere サーバ - データベース管理』
- 「モニタの終了」 『SQL Anywhere サーバ - データベース管理』
- 「モニタへの接続」 『SQL Anywhere サーバ - データベース管理』
- 「リソースのモニタリング」 『SQL Anywhere サーバ - データベース管理』

リソースのモニタリング

モニタでは、モニタリング対象の Mobile Link サーバの正常性や可用性の概要が **[モニタ]** タブに表示されます。

[モニタ] タブ

上ウィンドウ枠の表には、モニタリングされているリソースがリストされた表があります。ここで言う「リソース」とは、Mobile Link サーバのことです。この表には、リソースが現在実行中かどうか、およびリソースに対してユーザによる何らかの操作が必要かどうかも示されます。「リソースの状態とモニタの状態の解釈」 221 ページを参照してください。

[モニタ] タブの下ウィンドウ枠には、選択された Mobile Link サーバについて、警告と各種メトリックの現状が表示されます。これらのタブのほとんどには、グラフへのリンクが含まれています。グラフの表示範囲は、各グラフの右上にあるドロップダウン・リストと矢印を使用して変更できます。

[管理] タブ

[管理] タブは、管理者専用です。このタブでは、モニタリングされる Mobile Link サーバの選択、ユーザの追加と編集、およびモニタの設定ができます。

参照

- 「モニタ・ユーザの操作」 235 ページ
- 「モニタのメトリック」 221 ページ

リソースの状態とモニタの状態の解釈

[モニタ] タブの上ウィンドウ枠には、モニタリングされている Mobile Link サーバがリストされた表があります。この表の **[状態]** カラムには、モニタとそのリソース間の接続に関する情報が示されます。**[モニタの状態]** カラムには、リソースに対してオペレータや管理者ユーザによる何らかの操作が必要かどうかが示されます。「モニタ・ユーザの操作」 235 ページを参照してください。

リソースの状態

リソースは必ず次のいずれかの状態にあります。

- **実行中**　リソースは接続されており、モニタがメトリックを収集しています。
- **ブラックアウト**　モニタは、ブラックアウト期間の終了を待機しています。終了すると、リソースのモニタリングを再開します。
- **サーバ停止**　モニタリング対象の Mobile Link サーバが停止しています。
- **ホスト停止**　モニタが、リソースを実行しているコンピュータを特定できません。
- **不明**　モニタは、リソースをモニタリングしていません。

リソースのモニタの状態

リソースのモニタの状態は次のいずれかになります。

- **正常**　リソースに未解決の警告はありません。
- **注意が必要！**　リソースに 1 つ以上の警告があります。
- **モニタ停止**　リソースはモニタリングされていません。
- **不明**　リソースは実行中ではなく、警告もありません。

モニタのメトリック

モニタはが Mobile Link サーバから収集し、保存するメトリックには次のものが含まれますが、これに限定されません。

- リソースが実行中がどうか。
- リソースを実行しているコンピュータが、正常に実行されているかどうか、およびネットワークに接続されているかどうか。
- リソースが要求を受信して処理しているかどうか。

- 所定の期間内に Mobile Link サーバが実行した同期の回数。

メトリックの収集頻度は、管理者が指定する収集間隔の設定で決まります。「収集間隔」 229 ページを参照してください。

収集されるメトリック、および警告の発行に使用されるスレッショルドは、管理者が指定するメトリックの設定で決まります。「収集するメトリックの指定」 229 ページを参照してください。

メトリックの表示

モニタの表示は、1 分ごとに自動的に再表示されます。表示の再表示間隔を変更するには、**[ユーザ設定]** をクリックします。この設定は、リソースに対する収集間隔とは異なります。リソースに対する収集間隔は、モニタリング対象のリソースからモニタがメトリックを収集する頻度です。

♦ **表示の再表示間隔を設定するには、次の手順に従います。**

1. 右上隅の **[ユーザ設定]** をクリックします。
2. **[再表示間隔]** に時間を設定します。デフォルトは 1 分です。
3. **[OK]** をクリックします。

[モニタ] タブで **[データの再表示]** をクリックすると、最新のメトリックが収集されて表示されます。

♦ **メトリックを再表示するには、次の手順に従います。**

- **[データの再表示]** をクリックします。

[F5] キーを押すと、モニタは Web ブラウザを再ロードし、その時点までに収集されたメトリックを取得して表示します。

♦ **モニタを再ロードするには、次の手順に従います。**

- [F5] キーを押します。

メトリックのタブの説明

次のタブは、SQL Anywhere と Mobile Link サーバの両方のリソースに使用されます。

- 「[モニタ] タブ : [警告] タブ」 223 ページ
- 「[モニタ] タブ : [サーバ] タブ」 223 ページ

次のタブは、SQL Anywhere のリソースだけに使用されます。

- 「[モニタ] タブ : [CPU] タブ」 『SQL Anywhere サーバ - データベース管理』
- 「[モニタ] タブ : [スケジュールされていない要求] タブ」 『SQL Anywhere サーバ - データベース管理』

- 「[モニタ] タブ : [メモリ] タブ」 『SQL Anywhere サーバ - データベース管理』
- 「[モニタ] タブ : [ディスク] タブ」 『SQL Anywhere サーバ - データベース管理』
- 「[モニタ] タブ : [HTTP] タブ」 『SQL Anywhere サーバ - データベース管理』
- 「[モニタ] タブ : [接続] タブ」 『SQL Anywhere サーバ - データベース管理』
- 「[モニタ] タブ : [失敗した接続] タブ」 『SQL Anywhere サーバ - データベース管理』
- 「[モニタ] タブ : [クエリ] タブ」 『SQL Anywhere サーバ - データベース管理』
- 「[モニタ] タブ : [ミラー] タブ」 『SQL Anywhere サーバ - データベース管理』

次のタブは、Mobile Link サーバのリソースだけに使用されます。

- 「[モニタ] タブ : [同期] タブ」 224 ページ
- 「[モニタ] タブ : [統合データベース] タブ」 225 ページ
- 「[モニタ] タブ : [マシン・リソース] タブ」 225 ページ

[モニタ] タブ : [警告] タブ

警告を新しい方から 50 個リストします。警告の数が 50 を超えると、古い警告が削除され、新しい警告が表示されます。「警告」 239 ページを参照してください。

[モニタ] タブ : [サーバ] タブ

[Mobile Link サーバ]

- **[サーバ名]** 接続されているサーバの -zs オプションで指定された Mobile Link サーバの名前を示します。デフォルト値は **<default>** です。「-zs オプション」 123 ページを参照してください。
- **[バージョン]** 実行中のソフトウェアのバージョンを示します。
- **[開始時刻]** Mobile Link サーバが起動された時刻を示します。
- **[未送信のエラー・レポート]** 該当サーバについて未送信のエラー・レポートの数を示します。エラー・レポートは、SQL Anywhere ソフトウェアがクラッシュしたときに送信されます。「リソースからの未送信のエラー・レポートに関する警告の抑制」 242 ページを参照してください。

[ライセンス]

- **[ライセンス先の会社名]** ライセンスされた会社の名前を示します。
- **[ライセンス先のユーザ名]** ライセンスされたユーザの名前を示します。

[ホスト]

- **[名前]** Mobile Link サーバを実行しているコンピュータの名前を示します。通常は、コンピュータのホスト名です。
- **[オペレーティング・システムのプラットフォーム]** ソフトウェアを実行しているオペレーティング・システムを示します。
- **[プロセッサのアーキテクチャ]** プロセッサ・タイプを表す文字列を示します。
- **[CPU 数]** ソフトウェアを実行しているコンピュータの CPU 数を示します。

[追加情報]

- **[統合データベース・タイプ]** 統合データベースのタイプを示します。たとえば、SQL Anywhere などです。
- **[データベースへの最大同時アップロード数]** データベースへの同時アップロードの最大数を示します。「-wu オプション」 109 ページを参照してください。
- **[データベース・ワーカ・スレッド]** データベース・ワーカ・スレッドの数を示します。「-w オプション」 108 ページを参照してください。
- **[最大キャッシュ・サイズ]** mlsrv11 の -cm オプションで設定された Mobile Link サーバのメモリ・キャッシュの最大サイズを示します。「-cm オプション」 57 ページを参照してください。
- **[キャッシュ内の最大ページ数]** Mobile Link のメモリ・キャッシュのページ数を示します。この値は mlsrv11 の -cm オプションで暗黙的に設定されます。「-cm オプション」 57 ページを参照してください。
- **[データベース接続の最大数]** mlsrv11 の -cn オプションまたは -w オプションで設定されたデータベース接続の最大数を示します。「-cn オプション」 58 ページと「-w オプション」 108 ページを参照してください。
- **[TCP 接続の最大数]** mlsrv11 の -nc オプションで設定された TCP 接続の最大数を示します。「-nc オプション」 77 ページを参照してください。
- **[クライアントの最大数]** クライアントの最大数を示します。「-sm オプション」 96 ページを参照してください。
- **[統合バージョン]** 統合データベースのバージョンが表示されます。
- **[ドライバのバージョン]** 統合データベースのドライバのバージョンを示します。
- **[ドライバ名]** 統合データベースのドライバの名前を示します。
- **[ファームウェア内のプライマリ・サーバ]** サーバがプライマリかセカンダリかを示します。

参照

- 「[モニタ] タブ : [サーバ] タブ」 223 ページ

[モニタ] タブ : [同期] タブ

このタブは、Mobile Link サーバのモニタリング時に使用します。

- **[同期の完了率]** サーバの同期の完了率を同期数／秒の単位で示します。
- **[同期の失敗率]** サーバの同期の失敗率を同期数／秒の単位で示します。
- **[同期エラー率]** 同期のエラー率をエラー数／秒の単位で示します。
- **[同期警告率]** サーバの同期警告率を警告数／秒の単位で示します。
- **[最も長いアクティブな同期の時間]** サーバの最も古いアクティブな同期の経過時間を秒単位で示します。
- **[アクティブな要求]** サーバ内のアクティブな要求の数を示します。
- **[アップロードの適用]** 現在サーバで、アップロードの適用フェーズまたは同期の開始フェーズにある要求の数を示します。同期の各フェーズについては、「-v オプション」 104 ページを参照してください。
- **[ダウンロードの生成]** 現在サーバで、ダウンロードの準備フェーズ、ダウンロードのフェッチ・フェーズ、ダウンロード確認の待機フェーズ、または同期の終了フェーズにある要求の数を示します。同期の各フェーズについては、「-v オプション」 104 ページを参照してください。
- **[アクティブな認証]** 現在サーバで、ユーザ認証フェーズにある要求の数を示します。同期の各フェーズについては、「-v オプション」 104 ページを参照してください。

[モニタ] タブ : [統合データベース] タブ

このタブは、Mobile Link サーバのモニタリング時に使用します。

- **[使用中の接続]** この Mobile Link サーバで現在使用中のデータベース接続の数を示します。
- **[データベース・ワーカ・スレッドの最も長いアクティブ待機]** このサーバで、アクティブな要求がデータベース・ワーカ・スレッドを待機した最長時間を示します。
- **[データベース・ワーカ・スレッドの待機]** サーバで、データベース・ワーカ・スレッドを現在待機している要求の数を示します。
- **[使用中のアップロード接続の数]** サーバで現在使用中のアップロード接続の数を示します。

[モニタ] タブ : [マシン・リソース] タブ

このタブは、Mobile Link サーバのモニタリング時に使用します。

- **[CPU 使用率]** Mobile Link サーバによって使用された CPU 時間の割合を示します。
- **[合計 CPU 時間]** Mobile Link サーバによって使用された CPU 時間の合計を秒単位で示します。
- **[使用中の Mobile Link キャッシュ・ページ]** サーバで使用中の Mobile Link キャッシュ・ページの割合を示します。この値は mlsrv11 の -cm オプションで暗黙的に設定されます。「-cm オプション」 57 ページを参照してください。

- **[ロック中の Mobile Link キャッシュ・ページ]** サーバ・メモリにロードされている Mobile Link キャッシュ・ページの割合を示します。この値は mlsrv11 の -cm オプションで暗黙的に設定されます。「-cm オプション」 57 ページを参照してください。
- **[Mobile Link キャッシュ・ページ・イン]** サーバによって 1 秒間にディスクから読み込まれている Mobile Link キャッシュ・ページ数を示します。
- **[Mobile Link キャッシュ・ページ・アウト]** 1 秒間にディスクにスワップされている Mobile Link キャッシュ・ページ数を示します。
- **[メモリ使用率]** サーバで使用中の RAM のバイト数が表示されます (Windows サーバのみ)。
- **[Mobile Link キャッシュ用ディスク空き領域]** Mobile Link キャッシュ用の、テンポラリ・ディスク上の空きディスク領域をバイト単位で示します。
- **[開いているネットワーク接続]** サーバによって現在開かれている TCP 接続の数を示します。
- **[拒否されたネットワーク接続]** サーバで 1 秒間に拒否されたネットワーク接続の総数を示します。

古いモニタ・メトリックの削除

モニタが古いメトリックを保持する時間をカスタマイズできます。任意の設定またはすべての設定を使用できます。モニタでは、デフォルトで、毎日 1 回夜中の 12 時にメンテナンスが実行されます。メンテナンスの対象はメトリックで、警告は対象ではありません。

♦ **古いメトリックの削除を設定するには、次の手順に従います。**

1. **[管理]** をクリックします。
2. **[設定]** タブをクリックします。
3. **[編集]** をクリックします。
4. **[メンテナンス]** をクリックします。
5. メンテナンスの実行時刻を指定します。デフォルトでは、夜中の 12 時に実行されます。この時刻は、モニタが実行されているコンピュータのローカル時間に対応します。
6. **[データの削減]** 設定をカスタマイズします。
   - **[次の日数よりも古い値を 1 日ごとの平均値に置換する]** このオプションを選択すると、指定された日数より古いすべての数値メトリックの平均が計算され、その数値メトリックが削除されます。非数値メトリックは削除されません。
   - **[次の日数よりも古い値を削除する]** このオプションを選択すると、指定された日数より古いすべてのメトリックが削除されます。
   - **[SQL Anywhere モニタによって使用されている総ディスク領域が次の容量 (MB) を超えた場合に古い値を削除する]** このオプションを選択した場合は、メトリックの保存に使用できる容量の上限を指定します。使用されているディスク領域がこの指定値以上になると、メトリックが古い順に削除され、メトリック用に使用されるディスク領域が増えないように

します。メトリックは、新しいメトリックを保存するのに十分な空きディスク領域が確保されるまで削除されます。

7. **[保存]** をクリックします。

リソースの管理

ここで言う「リソース」とは、Mobile Link サーバのことです。リソースをモニタに追加すると、モニタリングを開始できます。

SQL Anywhere Monitor という名前のデフォルト・リソースは、モニタ自身の正常性についてレポートします。このリソースを変更したり、このリソースのモニタリングを停止することはできません。

リソースのモニタリングの開始

リソースのモニタリングを開始すると、モニタによってメトリックの収集が開始されます。

リソースのモニタリングは、次の場合に開始されます。

- リソースが追加されたとき (自動的に開始)。「リソースの追加」 228 ページを参照してください。
- モニタが起動されたとき (自動的に開始)。デフォルトでは、モニタを起動すると、既存のすべてのリソースが自動的に起動されます。
- ブラックアウト期間が終了したとき (自動的に開始)。モニタは、リソースへの接続とモニタリングの再開を自動的に試みます。
- 管理者が、**[管理]** タブを開き、**[リソース]** をクリックし、リストからリソースを選択し、**[起動]** をクリックしたとき。

リソースの追加

Mobile Link サーバをモニタリングするには、目的のリソースをあらかじめモニタに追加しておく必要があります。

モニタリング対象のリソースとして Mobile Link サーバを追加しても、サーバは何も変更されません。サーバを追加する場合は、Mobile Link サーバに接続するためのユーザ ID とパスワードを指定する必要があります。ユーザ ID とパスワードはモニタによって保持されます。

リソースを追加できるのは管理者だけです。デフォルトでは、リソースの追加時にリソースのモニタリングが開始します。

♦ **モニタリング対象のリソースを追加するには、次の手順に従います。**

1. モニタにログインします。
2. **[管理]** タブをクリックします。
3. **[リソース]** タブで **[追加]** をクリックします。
4. **[リソースの追加]** ウィンドウの指示に従って、Mobile Link サーバをモニタリングするためのリソースを追加します。

Mobile Link サーバを追加する場合は、そのサーバのユーザ ID とパスワードを指定する必要があります。これらのクレデンシャルは、Mobile Link サーバへの接続に使用されます。ユーザ ID とパスワードはモニタによって保持されます。

5. **[作成]** をクリックします。

   リソースが追加され、モニタリングが開始されます。

6. **[OK]** をクリックします。

収集間隔

収集間隔には、次の 3 種類があります。

- **収集間隔 (高)** 同期の完了率など、頻繁に変わる情報に対して使用します。
- **収集間隔 (中)** 使用可能なディスク領域など、それほど頻繁に変わらない情報に対して使用します。
- **収集間隔 (低)** 未送信のエラー・レポートなど、まれにしか変わらない情報に対して使用します。

管理者は、リソースのメトリックをどの程度の頻度で収集するかを設定できます。収集間隔は、リソースごとに設定します。デフォルト・リソースである SQL Anywhere モニタについて設定することはできません。

♦ **収集間隔を編集するには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[リソース]** タブをクリックし、リストからリソースを選択します。
3. **[設定]** をクリックします。
4. **[収集間隔]** をクリックします。
5. 必要に応じて他の設定を指定し、**[保存]** をクリックします。
6. **[OK]** をクリックします。

参照

- 「モニタのメトリック」 221 ページ
- 「収集するメトリックの指定」 229 ページ

収集するメトリックの指定

管理者は、モニタによって収集されるメトリックと、警報を発行するタイミングを設定できます。デフォルト・リソースである SQL Anywhere モニタについて設定することはできません。

♦ **収集するメトリックを設定するには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[リソース]** タブをクリックし、リストからリソースを選択します。
3. **[設定]** をクリックします。
4. **[メトリック]** をクリックします。メトリックと警告を選択します。メトリックと警告の定義については、「メトリックと警告のタイプ」 230 ページを参照してください。
5. その他の設定を必要に応じて指定します。
6. **[保存]** をクリックします。
7. **[OK]** をクリックします。

参照

- 「モニタのメトリック」 221 ページ
- 「収集間隔」 229 ページ

メトリックと警告のタイプ

ここでは、**[リソースの設定]** ウィンドウの **[メトリック]** タブで使用できるメトリックについて説明します。多くのデフォルト設定は任意です。同期システムはそれぞれ動作と制約が異なるので、デフォルト設定が環境に不適切である場合があります。各メトリックを慎重に検討し、必要に応じてそれぞれ設定してください。

- **[CPU 使用率 (収集間隔 (高))]**
  - **[CPU 使用率が、指定した秒数の間に指定したスレッショルドを超えた場合に警告します]** **[スレッショルド]** のデフォルトは 100% です。**[秒]** のデフォルトは 300 です。
- **[メモリ使用率 (収集間隔 (中))]**
  - **[使用されるキャッシュ・ページの割合が次の値 (%) よりも大きい場合に警告します : *X*]** デフォルトは 100 です。
  - **[ロックされているキャッシュ・ページの割合が次の値 (%) よりも大きい場合に警告します : *X*]** デフォルトは 80 です。
  - **[1 秒間にスワップ・インおよびスワップ・アウトされているページ数が、指定した秒数の間に指定したスレッショルドを超えた場合に警告します]** [スレッショルド] のデフォルトは 256 です。[秒] のデフォルトは 120 です。
- **[ネットワーク使用率 (収集間隔 (高))]** このオプションを選択すると、サーバのネットワーク使用率に関するメトリックが収集されます。該当するメトリックは **[マシン・リソース]** タブで確認できます。「[モニタ] タブ : [マシン・リソース] タブ」 225 ページを参照してください。
- **[同期 (収集間隔 (高))]** このオプションを選択すると、サーバの同期に関するメトリックが収集されます。該当するメトリックは **[同期]** タブで確認できます。「[モニタ] タブ : [同期] タブ」 224 ページを参照してください。

- ○ **[最も長いアクティブな同期の時間が次の値 (秒) よりも長い場合に警告します : *X*]** デフォルトは 600 です。
- ○ **[失敗した同期の数が、指定した分数の間に指定したスレッショルドを超えた場合に警告します]** 失敗した同期の [スレッショルド] のデフォルトは 20 です。[分] のデフォルトは 60 です。

- **[同期のスループット (収集間隔 (高))]** このオプションを選択すると、サーバの同期のスループットに関するメトリックが収集されます。該当するメトリックは **[同期]** タブで確認できます。「[モニタ] タブ : [同期] タブ」 224 ページを参照してください。
- **[エラー率 (収集間隔 (高))]** このオプションを選択すると、サーバのエラー率に関するメトリックが収集されます。
  - ○ **[エラー数が、指定した分数の間に指定したスレッショルドを超えた場合に警告します]** [スレッショルド (エラー数)] のデフォルトは 50 です。[分] のデフォルトは 60 です。
- **[警告率 (収集間隔 (高))]** このオプションを選択すると、警告率に関するメトリックが収集されます。
- **[使用中のデータベース接続 (収集間隔 (高))]** このオプションを選択すると、サーバで使用中のデータベース接続数に関するメトリックが収集されます。該当するメトリックは **[統合データベース]** タブで確認できます。「[モニタ] タブ : [統合データベース] タブ」 225 ページを参照してください。
- **[Mobile Link キャッシュ用ディスク空き領域 (収集間隔 (中))]** このオプションを選択すると、サーバで Mobile Link のキャッシュに使用できるディスク領域に関するメトリックが収集されます。該当するメトリックは **[マシン・リソース]** タブで確認できます。「[モニタ] タブ : [マシン・リソース] タブ」 225 ページを参照してください。
  - ○ **[Mobile Link キャッシュ用ディスク空き領域が次の値 (MB) を下回ったときに警告します : *X*]** デフォルトは 100 です。
- **[データベース・ワーカ・スレッドの最も長いアクティブ待機 (収集間隔 (高))]** このオプションを選択すると、サーバのデータベース・ワーカ・スレッドの最も長いアクティブ待機時間に関するメトリックが収集されます。
  - ○ **[データベース・ワーカ・スレッドの最も長いアクティブ待機が次の値 (秒) よりも長い場合に警告します : *X*]** デフォルトは 300 です。
- **[最も長いアクティブな同期の時間]** このオプションを選択すると、サーバの最も長いアクティブな同期の時間に関するメトリックが収集されます。該当するメトリックは **[同期]** タブで確認できます。このメトリックは収集間隔の高いリソースに対して設定する必要があります。「[モニタ] タブ : [同期] タブ」 224 ページを参照してください。
- **[同じ状況が次の時間内で発生した場合は警告しません]** このオプションを選択すると、指定した時間内に警告を重複して受信することがなくなります。デフォルトは 30 分です。

リソースのモニタリングの停止

モニタで Mobile Link サーバからメトリックを収集しない場合は、リソースのモニタリングを停止します。たとえば、目的のリソースが使用不可のときにモニタリングを停止します。そうしな

いと、リソースが使用可能になるまで警告を受信し続けることになります。デフォルト・リソースであるモニタを除き、リソースのモニタリングはいつでも停止できます。

リソースのモニタリングを停止すると、モニタは次のように動作します。

- 該当するリソースのメトリックの収集を停止します。
- 該当するリソースに関する警告の発行を停止します。

リソースのモニタリングを停止するには、次の 2 つの方法があります。

- **定期的に繰り返されるブラックアウト期間をスケジュールする**　この方法は、次の状況が当てはまる場合に適しています。
  - Mobile Link サーバのモニタリングを繰り返し停止する必要がある場合。たとえば、毎月末に定期メンテナンスを行う場合です。
  - Mobile Link サーバが使用不可になる期間があらかじめわかっている場合。たとえば、定期メンテナンスに 4 時間かかることがわかっている場合です。
  - モニタリングを自動的に再開する必要がある場合。ブラックアウト期間が終わると、モニタはリソースへの再接続を行い、データの収集を続けます。

  この方法を使用するには、ブラックアウト期間を設定して、指定した時刻にモニタによるモニタリングが停止されるようにする必要があります。「ブラックアウトを使用したリソースのモニタリングの自動停止」 233 ページを参照してください。
- **モニタリングを手動で停止する**　この方法は、次の状況が当てはまる場合に適しています。
  - まれにしか発生しないタスクや 1 回限りのタスクのためにモニタリングを停止する必要がある場合。たとえば、リソースが実行されているコンピュータを特別なメンテナンスのためにオフラインにする必要があり、モニタリングを停止する必要がある場合です。
  - モニタリングを後で再開するときに立ち会える場合。リソースが手動で停止されている場合、モニタはモニタリングの再開を待機します。

  この方法を使用する場合については、「リソースのモニタリングの手動停止」 232 ページを参照してください。

リソースのモニタリングを永久に停止する場合は、該当するリソースをモニタから削除することも可能です。「リソースの削除」 233 ページを参照してください。

リソースのモニタリングの手動停止

ここでは、リソースを手動で停止する方法について説明します。リソースを停止したときの動作の詳細については、「リソースのモニタリングの停止」 231 ページを参照してください。

♦ **リソースを手動で停止するには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. 停止するリソースを選択します。

3. **[リソース]** タブで **[停止]** をクリックします。

参照

- 「リソースのモニタリングの開始」 228 ページ
- 「ブラックアウトを使用したリソースのモニタリングの自動停止」 233 ページ

ブラックアウトを使用したリソースのモニタリングの自動停止

ここでは、リソースをブラックアウトを使用して停止する方法について説明します。リソースを停止したときの動作、およびブラックアウトの使用に適した状況の詳細については、「リソースのモニタリングの停止」 231 ページを参照してください。

ブラックアウトとは、モニタでメトリックを収集しない期間のことです。ブラックアウト期間が終わると、モニタはリソースへの再接続を行い、データの収集を続けます。

ブラックアウト期間は、該当リソースのローカル時間に対応します。

♦ **ブラックアウト期間を設定するには、次の手順に従います。**

1. モニタに管理者としてログインします。
2. **[管理]** タブをクリックします。
3. **[リソース]** タブで、ブラックアウト期間の指定対象リソースを選択します。
4. **[設定]** をクリックします。
5. **[ブラックアウト]** タブをクリックします。
6. **[新規]** をクリックします。
7. **[新規ブラックアウト期間]** ウィンドウで、ブラックアウトの日付と時刻を指定します。

   この日時は、対象リソースである Mobile Link サーバがあるコンピュータのローカル時間に対応します。
8. **[保存]** をクリックします。
9. **[保存]** をクリックします。
10. **[OK]** をクリックします。

参照

- 「リソースのモニタリングの開始」 228 ページ
- 「リソースのモニタリングの手動停止」 232 ページ

リソースの削除

削除するリソースは、使用されなくなったサーバなど、モニタリングが不要になったことが確実なリソースに限定してください。

リソースを削除したときのモニタの動作は次のとおりです。

- 該当リソースのモニタリングが永久的に停止されます。
- 該当リソースについて収集されたメトリックが破棄されます。

リソースを削除できるのは管理者だけです。**SQL Anywhere モニタ・リソース**は削除できません。

♦ **リソースを削除するには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[リソース]** タブでリソースを選択し、**[削除]** をクリックします。
3. **[はい]** をクリックします。

参照

- 「リソースのモニタリングの停止」 231 ページ

モニタ・ユーザの操作

モニタは次の 3 種類のユーザをサポートしています。

- **読み込み専用ユーザ**　モニタ・リソースに対する読み込み専用アクセスがあります。読み込み専用ユーザは **[モニタ]** タブのメトリックを表示できますが、**[管理]** タブにはアクセスできません。ユーザ名とパスワードが必要です。
- **オペレータ**　モニタ・リソースに対する読み込み専用アクセスがあり、警告を受信できます。このユーザは **[モニタ]** タブのメトリックの表示、電子メール警告の受信、および警告の解決と削除ができます。ただし、**[管理]** タブにはアクセスできません。ユーザ名とパスワードが必要です。
- **管理者**　オペレータと同じアクセス権を持ち、さらにリソースの設定とユーザの追加ができます。また、**[管理]** タブにもアクセスできます。デフォルト・ユーザである **admin** は管理者です。ユーザ名とパスワードが必要です。

モニタにログインするためのユーザ ID とパスワードでは、大文字と小文字が区別されます。

デフォルト・ユーザ

モニタの初回起動時には、デフォルトでユーザ名が **admin** でパスワードが **admin** の管理者ユーザが 1 つ用意されています。このユーザにはデフォルトですべてのパーミッションが与えられています。デフォルトの管理者パスワードを変更して、モニタへのアクセスを制限することをおすすめします。「モニタ・ユーザの編集」 236 ページを参照してください。

ユーザ名なしの読み込み専用アクセス

デフォルトでは、モニタへの読み込み専用アクセスにログインは不要です。ただし、セキュリティなどの理由から、管理者はログインを必須に設定できます。「モニタ・ユーザのログインの必須化」 237 ページを参照してください。

モニタ・ユーザの作成

モニタ・ユーザを追加するには、管理者である必要があります。

♦ **新しいモニタ・ユーザを追加するには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[ユーザ]** タブをクリックします。
3. **[新規]** をクリックします。
4. 新しいユーザの情報を入力します。電子メール・アドレスは、モニタからの電子メール警告を受信するユーザにだけ必要です。

   [保存] をクリックします。

5. オペレータまたは管理者を作成する場合は、そのユーザをリソースに関連付けることができます。「モニタ・ユーザとリソースの関連付け」 236 ページを参照してください。

参照

- 「モニタ・ユーザの編集」 『SQL Anywhere サーバ - データベース管理』

モニタ・ユーザとリソースの関連付け

ユーザがリソースに関する電子メール警告を受信するようにするには、そのユーザを該当するリソースに関連付ける必要があります。リソースとの関連付けの対象にできるのは、オペレータまたは管理者だけです。

♦ **オペレータまたは管理者をリソースと関連付けるには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[リソース]** タブをクリックします。
3. 目的のリソースを選択し、**[設定]** をクリックします。
4. **[オペレータ]** をクリックします。
5. **[選択可能なオペレータ]** リストでユーザを選択して、**[追加]** をクリックします。
6. **[保存]** をクリックします。
7. **[OK]** をクリックします。
8. モニタが警告の通知を電子メールで送信するよう設定されていることを確認します。「警告の電子メール送信」 241 ページを参照してください。

参照

- 「モニタ・ユーザの操作」 235 ページ

モニタ・ユーザの編集

管理者は、モニタ・ユーザについて次の設定値を編集できます。

- パスワード
- 電子メール・アドレス
- 言語設定
- ユーザ・タイプ

♦ **既存のモニタ・ユーザを編集するには、次の手順に従います。**

1. **[管理]** タブをクリックします。

2. **[ユーザ]** タブをクリックします。
3. 編集するユーザを選択します。
4. **[編集]** をクリックします。
5. 必要に応じてユーザの設定を変更します。
6. **[保存]** をクリックします。
7. オペレータまたは管理者を編集する場合は、そのユーザをリソースに関連付けることができます。「モニタ・ユーザとリソースの関連付け」 236 ページを参照してください。

参照

- 「モニタ・ユーザの操作」 235 ページ
- 「モニタ・ユーザの作成」 235 ページ
- 「モニタ・ユーザの削除」 237 ページ

モニタ・ユーザの削除

ユーザを削除すると、そのユーザがモニタから削除され、リソースとの関連付けがすべて破棄されます。

モニタ・ユーザを削除するには、管理者である必要があります。

♦ **既存のモニタ・ユーザを削除するには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[ユーザ]** タブをクリックします。
3. 削除するユーザを選択します。
4. **[削除]** をクリックします。
5. **[はい]** をクリックして、選択したユーザを削除します。すべてのユーザを削除するには、**[すべて削除]** をクリックします。

   ユーザがモニタから削除されます。

参照

- 「モニタ・ユーザの作成」 235 ページ
- 「モニタ・ユーザの編集」 236 ページ
- 「モニタ・ユーザとリソースの関連付け」 236 ページ

モニタ・ユーザのログインの必須化

デフォルトでは、あらゆるユーザにモニタへの読み込み専用アクセスがあります。この動作を変更して、ユーザがモニタを Web ブラウザで開く際に、ユーザ名とパスワードを入力しないとモニタリング・データが表示されないようにすることができます。

♦ **SQL Anywhwere モニタへのアクセスを制限するには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[設定]** タブで **[編集]** をクリックします。
3. **[認証]** をクリックします。
4. **[すべてのユーザに SQL Anywhere モニタの読み込み専用アクセスを許可します。]** オプションをオフにします。
5. **[保存]** をクリックします。

参照

- 「モニタ・ユーザの作成」 235 ページ
- 「モニタ・ユーザの編集」 236 ページ

警告

ここで言う「警告」とは、管理者またはオペレータによる注目を要する特定の状況または状態のことです。警告には、問題の原因に関する情報や、問題解決のアドバイスなどが含まれます。

空きディスク領域の減少、重要なソフトウェア更新、ログイン試行の失敗、高メモリ使用率などの状況については、事前に定義された警告があります。警告の条件が満たされると、その警告が **[モニタ]** タブの下ウィンドウ枠にリストされます。上ウィンドウ枠では、Mobile Link サーバの **[モニタの状態]** が変化して警告の存在を示します。警告が発生したときにオペレータや管理者に電子メールが送信されるようモニタを設定できます。「警告の電子メール送信」 241 ページを参照してください。

警告は、収集されたメトリックを基にモニタによって検出されます。モニタリングされている Mobile Link サーバでは検出されません。リソースを編集すると、デフォルトのスレッショルド値を変更したり、有効にする警告を選択したりできます。「モニタのメトリック」 221 ページを参照してください。

警告の表示

警告はあらゆるユーザが表示できますが、警告の解決と削除ができるのはオペレータと管理者だけです。

♦ **警告を表示するには、次の手順に従います。**

1. **[モニタ]** タブをクリックします。
2. リストからリソースを選択します。
3. 下ウィンドウ枠で **[警告]** タブをクリックします。
4. 警告リストのいずれかのローを選択します。
5. **[詳細]** をクリックします。
6. **[OK]** をクリックします。

参照

- 「警告の解決」 239 ページ
- 「警告の削除」 240 ページ
- 「警告の電子メール送信」 241 ページ

警告の解決

警告の原因となった問題に対応したら、その警告を解決済みとマークできます。警告を解決すると、警告の [モニタの状態] カラムが変更されますが、警告そのものは警告リストに残ります。警告をリストから削除するには、該当する警告を削除する必要があります。「警告の削除」 240 ページを参照してください。

警告を解決できるのは、オペレータと管理者だけです。

♦ **警告を解決するには、次の手順に従います。**

1. **[モニタ]** タブをクリックします。
2. リストからリソースを選択します。
3. 下ウィンドウ枠で **[警告]** タブをクリックします。
4. 警告リストの該当するローを選択します。
5. **[解決済みとマークする]** をクリックして、選択された警告を解決します。リストに含まれるすべての警告を解決するには、**[すべて解決済みとマークする]** をクリックします。

   [警告] タブの **[モニタの状態]** カラムの値が **[解決済み]** に変わります。

   その警告が該当リソースの唯一の未解決警告だった場合、リソースのモニタの状態は **[正常]** に変わります。

参照

- 「警告の削除」 240 ページ
- 「警告の解決」 239 ページ
- 「警告の電子メール送信」 241 ページ
- 「警告の表示」 239 ページ
- 「警告」 239 ページ

警告の削除

モニタでは、警告リストに新しい方から 50 個の警告だけが表示されます。警告リストに表示する必要がない警告は、削除してかまいません。警告は、そのモニタの状態に関係なく削除できます。

警告を削除できるのは、オペレータと管理者だけです。

♦ **警告を削除するには、次の手順に従います。**

1. **[モニタ]** タブをクリックします。
2. リストからリソースを選択します。
3. 下ウィンドウ枠で **[警告]** タブをクリックします。
4. 警告リストのいずれかのローを選択します。
5. **[削除]** をクリックします。

   選択した警告が警告リストから削除されます。

参照

- 「警告の解決」 239 ページ
- 「警告の電子メール送信」 241 ページ
- 「警告の表示」 239 ページ
- 「警告」 239 ページ

警告の電子メール送信

警告が発生したときにオペレータや管理者に電子メールが送信されるようモニタを設定できます。

警告の通知が電子メールで送信されるようにするには、次の処理が必要です。

1. 電子メール・アドレスを持つ管理者またはオペレータを作成します。「モニタ・ユーザの作成」 235 ページを参照してください。
2. 作成した管理者またはオペレータにリソースを関連付けます。「モニタ・ユーザとリソースの関連付け」 236 ページを参照してください。
3. モニタでの電子メールの送信を有効にします。「モニタによる警告電子メールの送信の有効化」 241 ページを参照してください。

モニタによる警告電子メールの送信の有効化

管理者は、警告が発生した場合に電子メールが送信されるようモニタを設定できます。モニタでは、SMTP プロトコルと MAPI プロトコルを使用した電子メール送信をサポートしています。

♦ **モニタによって警告の通知が電子メールで送信されるようにするには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[設定]** タブをクリックします。
3. **[編集]** をクリックします。
4. **[警告の通知]** をクリックします。
5. **[警告の通知を電子メールで送信]** を選択します。
6. **[電子メールでの警告の送信に使用するプロトコルを指定してください。]** フィールドで [SMTP] または [MAPI] を選択します。
7. その他の設定を必要に応じて指定します。
   - **MAPI**
     - **[ユーザ名]**　MAPI サーバのユーザ名を入力します。
     - **[パスワード]**　MAPI サーバのパスワードを入力します。
   - **SMTP**

- **[サーバ]** 使用する SMTP サーバを指定します。SMTP サーバのサーバ名か IP アドレスを入力します。たとえば、*SMTP.yourcompany.com* のように入力します。
- **[ポート]** SMTP サーバの接続ポート番号を指定します。デフォルトは 25 です。
- **[送信者名]** 送信者の電子メール・アドレスのエイリアスを指定します。たとえば、*JoeSmith* のように指定します。
- **[送信者のアドレス]** 送信者の電子メール・アドレスを指定します。たとえば、*jsmith@emailaddress.com* のように指定します。
- **[この SMTP サーバは認証が必要]** 使用する SMTP サーバに認証が必要な場合は、このオプションを選択します。
  - **[ユーザ名]** 認証が必要な SMTP サーバに対して使用するユーザ名を指定します。
  - **[パスワード]** 認証が必要な SMTP サーバに対して使用するパスワードを指定します。

8. 電子メールによる通知が適切に設定されたかどうかをテストします。

   [テスト電子メールを送信] をクリックします。

9. 画面の要求に従い、テスト電子メールの送信先電子メール・アドレスを入力して、**[OK]** をクリックします。

   テスト電子メールが、指定された電子メール・アドレスに送信されます。

10. **[保存]** をクリックします。

参照

- 「警告の解決」 239 ページ
- 「警告の削除」 240 ページ
- 「警告の表示」 239 ページ

リソースからの未送信のエラー・レポートに関する警告の抑制

管理者は、リソースに未送信のエラー・レポートがある場合に警告を送信するかどうかを設定できます。デフォルトでは、このような警告は送信されません。エラー・レポートとその送信方法の詳細については、「SQL Anywhere のエラー・レポート」『SQL Anywhere サーバ - データベース管理』を参照してください。

♦ **未送信のエラー・レポートに関する警告を抑制するには、次の手順に従います。**

1. **[管理]** タブをクリックします。
2. **[設定]** タブをクリックします。
3. **[編集]** をクリックします。
4. **[オプション]** をクリックします。
5. **[保存]** をクリックします。

別のコンピュータへの SQL Anywhere モニタのインストール

ここで示す各手順では、SQL Anywhere が実行されているコンピュータとは別のコンピュータに SQL Anywhere モニタをインストールする方法を説明します。

SQL Anywhere モニタを別のコンピュータで実行することには、次の利点があります。

- モニタがサービスとしてバックグラウンドで実行される。
- コンピュータの起動時にモニタが自動的に起動される。
- モニタが別のコンピュータにインストールされている場合、SQL Anywhere のアップグレードまたはアップデート時にモニタが上書きされない。これは別のコンピュータが運用環境にある場合に重要です。

♦ **モニタを別のコンピュータにインストールするには、次の手順に従います。**

- インストール・メディアの *Monitor* ディレクトリにある *setup.exe* ファイルを実行し、表示される指示に従います。

モニタのトラブルシューティング

問題	推奨事項
[F5] キーを押して Web ブラウザ・ウィンドウを再表示すると、モニタへのログインが要求される。	Web ブラウザで JavaScript を有効にします。
モニタに接続しようとすると、ネットワーク通信エラーになる。	モニタを起動します。「モニタの起動」 216 ページを参照してください。
Adobe Flash Player を最新バージョンにアップグレードした後も Adobe Flash Player のアップグレードを求めるメッセージが表示される。	インストールされている Adobe Flash Player のバージョンが、オペレーティング・システムでサポートされていることを確認します。モニタには、Adobe Flash Player のバージョン 9 との下位互換性があります。適切なバージョンを確認するには、http://www.adobe.com/jp/products/flashplayer/systemreqs/ にアクセスしてください。
モニタで SQL Anywhere データベース・リソースのモニタリングを開始できない。	該当リソースのパスワード検証関数とログイン・プロシージャで、user sa_monitor_user によるリソースへの接続が許可されていることを確認します。
警告の電子メールをまったく受信しない。	モニタが電子メールを送信するよう適切に設定されていることを確認し、テスト電子メールを送信します。「モニタによる警告電子メールの送信の有効化」 241 ページを参照してください。 モニタが送信する警告の電子メールがウイルス・スキャナによってブロックされていないことを確認します。「xp_startsmtp システム・プロシージャ」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

問題	推奨事項
モニタからレポートされるスケジュールされていない要求の数が、実際の数より少ない。	スケジュールされていない要求の数に関するメトリックを収集する際、モニタはリソースに対してクエリを実行します。このクエリがスケジュールされていない要求になっている可能性が考えられます。 スケジュールされていないクエリは、到着順に処理されます。そのため、モニタがクエリを実行しようとしたときにスケジュールされていない要求が存在すると、そのクエリは既存のスケジュールされていない要求が完了するのを待って処理されます。 その結果、モニタがスケジュールされていない要求の数を収集するとき、その数には、モニタがクエリを発行してからそのクエリが実行されるまでに存在したスケジュールされていない要求が含まれません。
データベースのディスク領域が指定されたスレッショルドを超えたのに警告を受信しない。	データベースは、モニタによる収集間隔の間に、指定されたディスク領域の警告スレッショルドと使用可能な領域の上限を超える可能性があります。このような状況になると、モニタがディスク使用率メトリックを収集して警告を発行する前に、データベースが応答しなくなります。 データベースのサイズが急激に増大する場合は、ディスク領域の警告スレッショルドの値を大きくして、データベースが領域不足になる前に警告を受信できるようにします。「メトリックと警告のタイプ」 230 ページを参照してください。

問題	推奨事項
英語以外のコンピュータで、Firefox Web ブラウザを使用してモニタを開くと、モニタが英語で表示される。	Firefox では、コンピュータの優先ロケールが正しく使用されません。Internet Explorer を使用するか、Firefox で次の回避方法を試してください。 1. Firefox で新しいタブを開きます。 2. アドレス・バーに次のように入力します。 about:config **[Enter]** キーを押します。 プロンプトが表示されたら、**[細心の注意を払って使用する]** をクリックします。 3. **[フィルタ]** フィールドに次のように入力します。 general.useragent.locale 4. 設定名のリストで **[general.useragent.locale]** をダブルクリックします。 5. **[文字列を入力してください]** ウィンドウにロケールを入力します。たとえば、フランス語は fr-FR、ドイツ語は de-DE、中国語は zh-CN、日本語は ja-JP と入力します。 6. **[OK]** をクリックします。

リレー・サーバ

目次

リレー・サーバの概要

リレー・サーバは、Web サーバを通じて通信する Mobile Link、Afaria、iAnywhere Mobile Office サーバとモバイル・デバイス間で、安全で負荷分散された通信を実現します。リレー・サーバには次の機能があります。

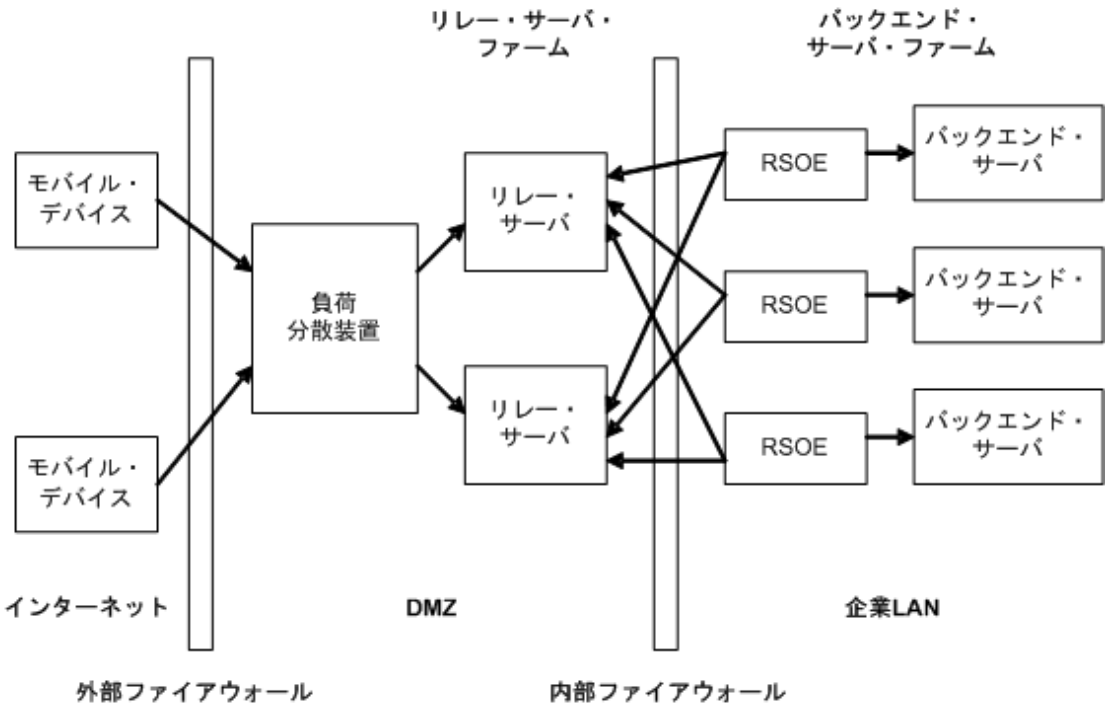
- Mobile Link、Afaria、iAnywhere Mobile Office サーバと通信するモバイル・デバイス用の共通の通信アーキテクチャ。
- Mobile Link、Afaria、iAnywhere Mobile Office サーバ用の負荷分散されたフォールト・トレラントな環境を可能にするメカニズム。
- 企業の既存のファイアウォール設定やポリシーと簡単に統合可能で、Mobile Link、Afaria、iAnywhere Mobile Office サーバとモバイル・デバイス間の通信を容易にする方法。

リレー・サーバのアーキテクチャ

リレー・サーバの配備環境は、次の要素で構成されます。

- 企業 LAN 内で実行されているバックエンド・サーバと通信する必要のあるクライアント・アプリケーションやサービスを実行しているモバイル・デバイス。
- モバイル・デバイスからの要求をリレー・サーバのグループに送信する、オプションの負荷分散装置。
- 企業 DMZ 内で実行されている 1 つ以上のリレー・サーバ。
- 企業 LAN 内で実行され、クライアント要求を処理する役割を持っているバックエンド・サーバ。
- バックエンド・サーバごとに 1 つの Mobile Link リレー・サーバ Outbound Enabler (RSOE)。Outbound Enabler は、バックエンド・サーバとリレー・サーバ・ファームとの間のあらゆる通信を管理します。

次の図は、リレー・サーバのアーキテクチャを示しています。



リレー・サーバは Web 拡張機能、ステータス情報を保持するバックグラウンド・プロセス、Web サーバのセットで構成されます。

リレー・サーバは Web サーバで実行されている Web 拡張機能であるため、すべての通信は HTTP または HTTPS を使用して実行されます。HTTP を使用すると、企業の既存のファイアウォール設定やポリシーとの統合が容易です。リレー・サーバでは、企業 LAN からリレー・サーバへの接続を企業 LAN 内から開始することが必要になります。これによって、DMZ から企業 LAN 内へのインバウンド接続が必要ではなくなるため、より安全な配備環境が提供されます。

リレー・サーバは、クライアント拡張機能、サーバ拡張機能の 2 つの Web 拡張機能が含まれます。クライアント拡張機能は、モバイル・デバイスで実行されているアプリケーションから行われたクライアント要求を処理します。サーバ拡張機能は、バックエンド・サーバに代わって Outbound Enabler によって行われた要求を処理します。

リレー・サーバ・ファーム

リレー・サーバ・ファームは、任意の数のリレー・サーバとフロントエンドの負荷分散装置から成ります。単一のリレー・サーバを持つリレー・サーバ・ファームを設定することもできます。この場合、負荷分散装置は必要ありません。また、モバイル・デバイスはリレー・サーバに直接接続できます。

バックエンド・サーバ・ファーム

バックエンド・サーバ・ファームは、同種のバックエンド・サーバで構成されるグループです。リレー・サーバ・ファームを通じて要求を行うクライアントは、対象とするバックエンド・サーバ・ファームを指定する必要があります。

負荷分散装置

負荷分散装置は、モバイル・デバイスからの要求をリレー・サーバ・ファームで実行されているリレー・サーバに送信します。リレー・サーバが 1 つだけである場合、負荷分散装置は必要ありません。

Mobile Link リレー・サーバ Outbound Enabler

Mobile Link リレー・サーバ Outbound Enabler は、バックエンド・サーバと同じコンピュータ上で実行されます。主な機能は、バックエンド・サーバに代わってリレー・サーバ・ファーム内のすべてのリレー・サーバへのアウトバウンド接続を開始することです。バックエンド・サーバごとに 1 つの Outbound Enabler があります。「Outbound Enabler」 256 ページを参照してください。

リレー・サーバ設定ファイル

リレー・サーバ設定ファイルは、リレー・サーバ・ファームと、リレー・サーバ・ファームに接続するバックエンド・サーバ・ファームの両方を定義するために使用します。リレー・サーバ設定ファイルは、いくつかのセクションに分かれています。

- 「リレー・サーバ・セクション」 251 ページ
- 「バックエンド・ファーム・セクション」 252 ページ
- 「バックエンド・サーバ・セクション」 253 ページ
- 「オプション・セクション」 253 ページ

各セクションは、セクション・タグで始まります。セクション・タグは、セクション名を識別するキーワードを角カッコで囲んだ形式になっています。たとえば [relay_server] は、リレー・サーバ・セクションの始まりを示します。

セクション・タグの後には、定義されているセクションに関連するさまざまなプロパティを定義する多くの行が続きます。プロパティは、等号の左側にプロパティ名、等号の右側にその値を指定することによって定義されます。たとえば、property name = value のように指定します。どのセクション名およびプロパティ名も、大文字と小文字が区別されません。コメントは、行の先頭のシャープ記号 (#) 文字によって示されます。

設定ファイルには、7 ビット ASCII 文字以外を含めることはできません。セクションは任意の順序で指定できます。

リレー・サーバ・セクション

リレー・サーバ・セクションは、1 つのリレー・サーバを定義するために使用されるため、ファーム内のリレー・サーバごとにリレー・サーバ・セクションが必要です。このセクションは、relay_server キーワードによって識別されます。

リレー・サーバ・セクションのプロパティ

リレー・サーバ・セクションでは、次のプロパティを指定できます。

- **enable**　リレー・サーバ・ファームにこのリレー・サーバを含めるかどうかを指定します。可能な値は、次のとおりです。
  - **Yes**　リレー・サーバ・ファームにこのリレー・サーバを含めることを示します。
  - **No**　リレー・サーバ・ファームにこのリレー・サーバを含めないことを示します。

  デフォルトは Yes です。このプロパティは省略可能です。
- **host**　Outbound Enabler がリレー・サーバへの直接接続を行うために使用するホスト名または IP アドレス。
- **http_port**　Outbound Enabler がリレー・サーバへの直接接続を行うために使用する HTTP ポート。値 「0」 または 「off」 を指定すると、HTTP 接続が無効になります。デフォルトでは、このプロパティは有効であり、80 に設定されています。

- ○ **0 または off**　Outbound Enabler からの HTTP アクセスを無効にします。
- ○ **1 ～ 65535**　指定されたポートでの HTTP アクセスを有効にします。

● **https_port**　Outbound Enabler がリレー・サーバへの直接接続を行うために使用する HTTPS ポート。値 「0」 または 「off」 を指定すると、HTTPS 接続が無効になります。デフォルトでは、このプロパティは有効であり、443 に設定されています。

- ○ **0 または off**　Outbound Enabler からの HTTPS アクセスを無効にします。
- ○ **1 ～ 65535**　指定されたポートでの HTTPS アクセスを有効にします。

● **description**　カスタム説明を最大 2048 文字で入力します。このプロパティは省略可能です。

バックエンド・ファーム・セクション

バックエンド・ファーム・セクションは、バックエンド・サーバ・ファームのプロパティを指定します。バックエンド・サーバ・ファームは、同種のバックエンド・サーバで構成されるグループです。リレー・サーバ・ファームを通じて要求を行うクライアントは、対象とするバックエンド・サーバ・ファームを指定する必要があります。バックエンド・サーバ・ファームごとに、1 つのバックエンド・ファーム・セクションがあります。

このセクションは、backend_farm キーワードによって識別されます。

バックエンド・ファーム・セクションのプロパティ

バックエンド・ファーム・セクションでは、次のプロパティを指定できます。

● **enable**　このバックエンド・サーバ・ファームからの接続を許可するかどうかを指定します。可能な値は、次のとおりです。

- ○ **Yes**　このバックエンド・サーバ・ファームからの接続を許可します。
- ○ **No**　このバックエンド・サーバ・ファームからの接続を禁止します。

デフォルトは Yes です。このプロパティは省略可能です。

● **id**　バックエンド・サーバ・ファームに割り当てる最大 2048 文字の名前。

● **client_security**　バックエンド・サーバ・ファームがクライアントに要求するセキュリティ・レベルを指定します。値は次のいずれかです。

- ○ **on**　クライアントが HTTPS を使用して接続する必要があることを示します。
- ○ **off**　クライアントが HTTP を使用して接続する必要があることを示します。

このプロパティは省略可能です。値が指定されていない場合、クライアントは HTTP、HTTPS のどちらを使用しても接続できます。

● **backend_security**　バックエンド・サーバ・ファーム内の Outbound Enabler がリレー・サーバ・ファームに接続するのに必要なセキュリティ・レベルを指定します。値は次のいずれかです。

- ○ **on**　バックエンド・ファームからのすべての接続を、HTTPS を使用して行う必要があることを示します。

- ○ **off**　バックエンド・ファームからのすべての接続を、HTTP を使用して行う必要があることを示します。

このプロパティは省略可能です。値が指定されていない場合、HTTP、HTTPS のどちらを使用しても接続できます。

- **description**　カスタム説明を最大 2048 文字で入力します。このプロパティは省略可能です。

バックエンド・サーバ・セクション

バックエンド・サーバ・セクションは、バックエンド・サーバ接続を定義します。このセクションは、Outbound Enabler がバックエンド・サーバに代わってリレー・サーバ・ファームに接続するときに使用する情報を指定します。リレー・サーバ・ファームに接続する Outbound Enabler ごとに、1 つのバックエンド・サーバ・セクションがあります。バックエンド・サーバ・セクションは、バックエンド・サーバをバックエンド・サーバ・ファームに割り当てることも行います。

このセクションは、backend_server キーワードによって識別されます。

バックエンド・サーバ・セクションのプロパティ

バックエンド・サーバ・セクションでは、次のプロパティを指定できます。

- **enable**　このバックエンド・サーバからの接続を許可するかどうかを指定します。可能な値は、次のとおりです。
  - ○ **Yes**　このバックエンド・サーバからの接続を許可します。
  - ○ **No**　このバックエンド・サーバからの接続を禁止します。

  デフォルトは Yes です。このプロパティは省略可能です。
- **id**　バックエンド・サーバ接続に割り当てる最大 2048 文字の名前。
- **farm**　このバックエンド・サーバが属するバックエンド・サーバ・ファームの名前。
- **MAC**　Outbound Enabler がリレー・サーバと通信するために使用するネットワーク・アダプタの MAC アドレス。アドレスは、IEEE 802 MAC-48 フォーマットを使用して指定します。正しいフォーマットの MAC アドレスを取得するには、Mobile Link リレー・サーバ Outbound Enabler のコンソールまたはログを確認します。このプロパティは省略可能です。指定しない場合、MAC アドレスのチェックは行われません。
- **token**　リレー・サーバがバックエンド・サーバ接続を認証するために使用する、最大 2048 文字のセキュリティ・トークン。このプロパティは省略可能です。
- **description**　カスタム説明を最大 2048 文字で入力します。このプロパティは省略可能です。

オプション・セクション

オプション・セクションは、ファーム内の各リレー・サーバに適用するプロパティを指定します。オプション・セクションは 1 つのみ使用できます。

このセクションは、options キーワードによって識別されます。

オプション・セクションのプロパティ

オプション・セクションでは、次のプロパティを指定できます。

- **start**　ステイト・マネージャを起動する方法。値は次のいずれかです。
  - **auto**　ステイト・マネージャは、ステイト・マネージャのコマンド・ラインのデフォルトを使用して自動的に起動されます。
  - **no**　ステイト・マネージャは、Windows サービスとして外部で起動されます。
  - **full path**　ステイト・マネージャの実行プログラム (*rshost*) へのフル・パスを指定します。

  デフォルトは auto です。このプロパティは省略可能です。
- **shared_mem**　リレー・サーバがステータスの追跡機能に使用する共有メモリの最大量を指定します。デフォルトは 10 メガバイトです。このプロパティは省略可能です。
- **verbosity**　verbosity は次のレベルに設定できます。
  - **0**　エラーのみをログに取ります。配備時には、このログ・レベルを使用してください。これはデフォルトです。
  - **1**　要求レベル・ロギング。すべての HTTP 要求がログ・ファイルに書き込まれます。

  エラーは指定されたログ・レベルに関係なく表示され、警告はログ・レベルが 0 よりも大きい場合にのみ表示されます。

リレー・サーバ設定ファイルのフォーマット

リレー・サーバ設定ファイルの基本的なフォーマットは、次のとおりです。

```
#
# Options
#
[options]
# List of Relay Server properties that apply to all Relay Servers
option = value

#
# Define a Relay Server section, one for each
# Relay Server in the Relay Server farm
#
[relay_server]
# List of properties for the Relay Server
property = value

#
# Define a backend server farm section, one for each backend
# server farm
#
[backend_farm]
# List of properties for a backend server farm
property = value
```

```
#
# Define a backend server section, one for each
# Outbound Enabler connecting to the Relay Server farm
#
[backend_server]
# List of properties for the backend server connection
property = value
```

Outbound Enabler

Outbound Enabler は、バックエンド・サーバと同じコンピュータ上で実行されます。その目的は、次のとおりです。

- 企業 LAN 内で稼働しているコンピュータから DMZ 内で実行されているリレー・サーバ・ファームへのアウトバウンド接続を開く。
- リレー・サーバから受信したクライアント要求をバックエンド・サーバに転送し、バックエンド・サーバからの応答をリレー・サーバ経由でクライアントに転送する。

Outbound Enabler は、起動されると、ファーム内で実行されているリレー・サーバのリストを取得するための HTTP 要求を行います。これは、リレー・サーバの Web サーバ拡張機能コンポーネントにマッピングするサーバ URL を使用して行われます。サーバ URL は、リレー・サーバに直接マッピングすることも、負荷分散装置にマッピングすることもできます。サーバ URL が負荷分散装置にマッピングされている場合、負荷分散装置は、要求をファーム内で実行されているリレー・サーバの 1 つに転送します。Outbound Enabler からの要求を受信したリレー・サーバは、ファーム内のすべてのリレー・サーバの接続情報を返します。次に、Outbound Enabler は、返された各リレー・サーバへのアウトバウンド接続 (チャネルと呼ばれます) を 2 つ作成します。1 つは上りチャネルと呼ばれ、基本的に応答の長さが無限である HTTP 要求を使用して作成されます。応答は、リレー・サーバから Outbound Enabler へのクライアント要求から構成される連続的なストリームです。2 つ目のチャネルは下りチャネルと呼ばれ、基本的にコンテンツの長さが無限である HTTP 要求を使用して作成されます。要求は、クライアント要求に対するサーバ応答の連続的なストリームから構成されます。

Outbound Enabler は、接続しているリレー・サーバのいずれかから上りチャネルでクライアント要求を受信すると、それを Outbound Enabler が処理しているバックエンド・サーバに転送します。応答は、バックエンド・サーバから受信されると、下りチャネルを使用して対応する要求を受信したリレー・サーバに転送されます。

Outbound Enabler の構文

rsoe [*option*]+

rsoe @{ *filename* | *environment-variable* } ...

パラメータ

オプション　Outbound Enabler では次のオプションを使用できます。これらはすべて省略可能です。

rsoe のオプション	説明
@data	指定された環境変数または設定ファイルからオプションを読み込みます。設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化できます。「ファイル難読化ユーティリティ (dbfhide)」 258 ページを参照してください。

rsoe のオプション	説明
-f *farm*	バックエンド・サーバが属するファームの名前。
-id *id*	バックエンド・サーバに割り当てられた名前。
-cs *"connection-string"*	バックエンド・サーバへの接続に使用するホストとポート。デフォルトは "host=localhost;port=80". です。
-cr *"connection-string"*	リレー・サーバの接続文字列。リレー・サーバの接続文字列のフォーマットは、名前と値のペアがセミコロンで区切られたリストです。名前と値のペアは、次の要素で構成されます。 ● **host** リレー・サーバの IP アドレスまたはホスト名。デフォルトは localhost です。 ● **port** リレー・サーバが受信しているポート。このオプションは必須です。 ● **url_suffix** リレー・サーバのサーバ拡張機能への URL パス。 Windows の場合のデフォルトは、*/ias_relay_server/server/rs_server.dll* です。 Linux の場合のデフォルトは、*/srv/iarelayserver* です。 ● **https** 0 - HTTP (デフォルト) 1 - HTTPS **https=1** の場合は、次のオプションも指定できます。 ● **tls_type** RSA ● **certificate_name** 証明書の通称フィールド。 ● **certificate_company** 証明書の組織名フィールド。 ● **certificate_unit** 証明書の組織単位フィールド。 ● **trusted_certificates** 信頼できるルート証明書のリストを含むファイル。
-t *token*	リレー・サーバに渡すセキュリティ・トークン。

rsoe のオプション	説明
-v *level*	ログを取るために使用する冗長性レベルを設定します。*level* は、**0**、**1**、**2** に設定できます。 ● **0** エラーのみをログに取ります。配備時には、このログ・レベルを使用してください。 ● **1** セッション・レベル・ロギング。これは、同期セッションの概要です。 ● **2** 要求レベル・ロギング。同期セッション内の HTTP 要求についてさらに詳細を提供します。
-d *seconds*	リレー・サーバ接続のリトライ間隔。デフォルトは 5 秒です。
-s	Outbound Enabler を停止します。

ファイル難読化ユーティリティ (dbfhide)

ファイル難読化ユーティリティ (dbfhide) は、単純暗号化を使用して、設定ファイルと初期化ファイルの内容を難読化します。

構文

dbfhide *original-configuration-file encrypted-configuration-file*

オプション	説明
original-configuration-file	元のファイルの名前を指定します。
encrypted-configuration-file	難読化された新しいファイルの名前を指定します。

リレー・サーバと Outbound Enabler は、dbfhide を使用して設定ファイルが難読化されたことを検出し、それに応じて処理します。

このユーティリティは、設定ファイルからオプションを読み込む @data パラメータを受け入れません。

配備に関する考慮事項

Outbound Enabler を使用しているときは、次の考慮事項に注意してください。

- **Windows サービスとしての Outbound Enabler** サービス・ユーティリティを使用して、Outbound Enabler を Windows サービスとして設定および管理することもできます。「リレー・サーバ・ステイト・マネージャ」 259 ページを参照してください。
- **認証** 単純認証やダイジェスト認証は使用できません。*rsoe.exe* は、Web サーバの種類やオペレーティング・システムにかかわらず、Web サーバでの単純認証やダイジェスト認証はサポートしていません。

リレー・サーバ・ステイト・マネージャ

リレー・サーバ・ステイト・マネージャは、クライアント要求と Outbound Enabler セッションを通じてリレー・サーバのステータス情報を保持するプロセスです。ステイト・マネージャは、リレー・サーバによって使用されるログ・ファイルを管理する役割も持っています。ステイト・マネージャは、リレー・サーバによって自動的に起動することも、Windows サービスとして起動することも (Windows の場合のみ) できます。

デフォルトのログ・ファイル名は、*ias_relay_server_host.log* です。Windows では、このファイルは、TEMP 環境変数によって指定されたディレクトリにあります。Linux では、このファイルは TMP、TEMP、または TMPDIR 環境変数によって指定されたディレクトリにあります。いずれの変数も設定されていない場合、ログ・ファイルはルートに作成されます。

注意
どのような場合も、Apache ユーザ・プロセスには選択した *tmp* ディレクトリのロケーションに対する書き込みパーミッションが必要です。

通常のシャットダウン時に、ステイト・マネージャは、ログ・ファイルを <*yymmdd*><*nn*>*.log* という形式のファイル名に変更します。<*yymmdd*> はログ・ファイルの名前が変更されたときの日付を表し、<*nn*> はその日のログ・ファイルの連続するバージョン番号です。

ステイト・マネージャを Windows サービスとして起動することをおすすめします。ステイト・マネージャをコマンド・ラインから手動で起動することはサポートされていません。

ステイト・マネージャを起動するために、リレー・サーバによって使用されるオプションを指定することができます。オプションを変更するには、リレー・サーバ設定ファイルのオプション・セクションで 「start」 プロパティを設定します。次に例を示します。

```
[options]
start = "rshost -o c:¥temp¥myrshost.log"
```

オプションの前に、リレー・サーバ・ステイト・マネージャの実行プログラムの名前 (rshost) を指定してください。

リレー・サーバ・ステイト・マネージャを Windows サービスとして起動する

Windows の場合のみ、サービス・ユーティリティ *dbsvc.exe* を使用して、ステイト・マネージャを Windows サービスとして起動することができます。これには、リレー・サーバ設定ファイルのオプション・セクションで start プロパティを 「no」 に設定する必要があります。「オプション・セクション」 253 ページを参照してください。

サービス・ユーティリティ (dbsvc) は、サービスを作成、変更、削除するときに使用されます。使用法の詳細については、オプションを指定しないで *dbsvc.exe* を実行してください。

自動起動するステイト・マネージャ・サービス rs を設定する場合

```
dbsvc -as -s auto -w rs "C:¥inetpub¥wwwroot¥ias_relay_server¥server¥rshost.exe" -q -qc -f c:¥inetpub
¥wwwroot¥ias_relay_server¥server¥rstest.config -o c:¥temp¥rs.log
```

サービスを開始する場合

dbsvc.exe -u rs

サービスを停止する場合

dbsvc.exe -x rs

サービスをアンインストールする場合

dbsvc.exe -d rs

リレー・サーバ・ステイト・マネージャを自動的に起動する

最初の Outbound Enabler がリレー・サーバに接続すると、ステイト・マネージャ・プロセスが自動的に開始されます。リレー・サーバ設定ファイルのオプション・セクションで、start プロパティが指定されていないか、このプロパティが明示的に auto と指定されている場合、これがデフォルトの動作です。デフォルトのログ・ファイルのロケーションは、*%temp%¥ias_relay_server_host.log* です。「オプション・セクション」 253 ページを参照してください。

カスタマイズ・オプションを指定してリレー・サーバ・ステイト・マネージャを自動的に起動する

自動起動が必要で、冗長性レベルやログ・ファイル・ロケーションなど一部のデフォルト動作を上書きしたい場合は、リレー・サーバ設定ファイルのオプション・セクションの start プロパティで、ステイト・マネージャのコマンド・ラインを明示的に指定することができます。この場合、-f オプションは使用できません。設定ファイルの名前は *rs.config* にして、サーバ拡張機能と同じディレクトリに配置する必要があります。「リレー・サーバ・ステイト・マネージャのコマンド・ラインの構文」 260 ページを参照してください。

注意
wwwroot ディレクトリの下にログ・ファイル・ロケーションを指定しないでください。IIS では、ワーカ・プロセスがパブリッシュされたツリーの下にファイルを作成することを許可していません。

リレー・サーバ・ステイト・マネージャのコマンド・ラインの構文

rshost [*option*]+

パラメータ

オプション　ステイト・マネージャを設定するために使用できるオプションは次のとおりです。これらはすべて省略可能です。

rshost のオプション	説明
-f *filename*	リレー・サーバ設定ファイルのファイル名。
-o *filename*	ログを取るために使用するファイル名。
-oq	起動エラーの発生時にポップアップ・ウィンドウを表示しません。
-q	最小化ウィンドウで実行します。
-qc	完了後、ウィンドウを閉じます。
-u	実行しているリレー・サーバの設定を更新します。
-ua	ログ・ファイルを <yymmdd><nn>.log にアーカイブしてトランケートします。

リレー・サーバの配備

次に、リレー・サーバを Windows 上の IIS に配備する方法の概要について説明します。

1. リレー・サーバ・コンポーネントを配備します。「Windows 上の IIS へのリレー・サーバ・コンポーネントの配備」 262 ページを参照してください。
2. Web サーバ拡張機能とステイト・マネージャを配備します。「Web サーバ拡張機能とステイト・マネージャの配備」 263 ページを参照してください。
   a. アプリケーション・プールを作成します。「アプリケーション・プールの作成」 263 ページを参照してください。
   b. リレー・サーバ Web 拡張機能を有効にし、リレー・サーバ設定ファイルを配備します。「Web サーバ拡張機能とステイト・マネージャの配備」 263 ページを参照してください。
3. 必要に応じて、リレー・サーバ設定の更新を配備します。「Windows 上の IIS 用のリレー・サーバ設定の更新」 269 ページを参照してください。

次に、リレー・サーバを Linux 上の Apache に配備する方法の概要について説明します。

1. リレー・サーバ・コンポーネントを配備します。「Linux 上の Apache へのリレー・サーバ・コンポーネントの配備」 265 ページを参照してください。
2. Web 拡張機能ファイルとステイト・マネージャを配備します。「Web 拡張機能ファイルとステイト・マネージャの配備」 265 ページを参照してください。
3. 必要に応じて、リレー・サーバ設定の更新を配備します。「Linux 上の Apache 用のリレー・サーバ設定の更新」 269 ページを参照してください。

Windows 上の IIS へのリレー・サーバ・コンポーネントの配備

Windows 用のリレー・サーバは、次の実行プログラムから構成されます。

- *rs_client.dll*
- *rs_server.dll*
- *rshost.exe*
- *dblgen11.dll*
- *dbsvc.exe*
- *dbfhide.exe*
- *dbicu11.dll*
- *dbicudt11.dll*
- *dbsupport.exe*
- *dbghelp.dll*

参照

- 「リレー・サーバ・ステイト・マネージャ」 259 ページ
- 「ファイル難読化ユーティリティ (dbfhide)」 258 ページ

Web サーバ拡張機能とステイト・マネージャの配備

♦ **リレー・サーバ・ファイルを配備するには、次の手順に従います。**

1. リレー・サーバに使用する、Web サイトのホーム・ディレクトリの下に次のディレクトリを作成します。
   - *ias_relay_server*
   - *ias_relay_server¥client*
   - *ias_relay_server¥server*
2. *rs_client.dll* を *ias_relay_server¥client* ディレクトリにコピーします。
3. リレー・サーバ設定ファイル *rs.config* を作成します。「リレー・サーバ設定ファイル」 251 ページを参照してください。
4. *rs_server.dll*、*rshost.exe*、*rs.config* を *ias_relay_server¥server* ディレクトリにコピーします。
5. Web サイトのホーム・ディレクトリの接続タイムアウト・プロパティが 60 秒以上に設定されていることを確認します。

アプリケーション・プールの作成

rs_server.dll と *rs_client.dll* の各 Web サーバ拡張機能用に専用のアプリケーション・プールを作成する必要があります。リレー・サーバは長時間実行するワーカ・プロセスを利用するため、すべてのワーカ・リサイクル・オプションをオフにする必要があります。

♦ **アプリケーション・プールを作成するには、次の手順に従います。**

1. IIS マネージャ・コンソールを起動します。
2. **[アプリケーション プール]** を右クリックし、RS_POOL などの新しいアプリケーション・プールを作成します。
3. 作成したアプリケーション・プールのプロパティを編集します。
   a. **[リサイクル]** タブを選択し、すべてのリサイクル・オプションをオフにします。
   b. **[パフォーマンス]** タブを選択し、次の操作を行います。
      i. **[アイドルなワーカー プロセスの解放までの待ち時間]** をオフにします。
      ii. ワーカ・プロセスの数をプロセッサ・コアの合計数に設定します。使用率やパフォーマンスの傾向に応じてこの値をさらに調整することができます。詳細については、Web ガーデン・サイズに関する IIS パフォーマンスの記述を参照してください。

リレー・サーバの Web 拡張機能の有効化

次の手順では、リレー・サーバの Web 拡張機能を有効にする処理について説明します。

♦ ias_relay_server のプロパティを編集し、リレー・サーバの Web 拡張機能を有効にするには、次の手順に従います。

1. **[ディレクトリ]** タブを選択し、次の操作を行います。
   a. 実行のアクセス許可を **[スクリプトおよび実行可能ファイル]** に設定します。
   b. **[アプリケーションの設定]** の下の **[作成]** をクリックします。関連付けるアプリケーション・プールとして「アプリケーション・プールの作成」 263 ページで作成したアプリケーション・プールを選択します。
2. **[ディレクトリ セキュリティ]** タブを選択し、次の操作を行います。
   a. **[認証とアクセス制御]** で **[編集]** をクリックします。
   b. 匿名アクセスを有効にし、Administrators グループに属するユーザ名とパスワードを設定します。

      または、設定を組み込みユーザ **IUSR_%computername%** のままにして、IIS メタベースへのアクセス・パーミッションを付与するために次のコマンドを実行する方法もあります。

      ```
      C:¥Windows¥Microsoft.Net¥Framework¥<Version>¥aspnet_regiis.exe -ga IUSR_
      %computername%
      ```
3. IIS マネージャの **[Web サーバー拡張]** で、*rs_server.dll* と *rs_client.dll* を ISAPI として実行することを許可します。
4. リレー・サーバ設定ファイルを作成し、*ias_relay_server¥server* ディレクトリにコピーすることによって、リレー・サーバ設定ファイルを配備します。

参照

- 「リレー・サーバ設定ファイル」 251 ページ

パフォーマンスに関するヒント

リレー・サーバを Windows 上の IIS に配備するときは、次の点に注意してください。

- リレー・サーバ Web 拡張機能は、ASP.NET を使用しません。ASP.NET ISAPI フィルタを削除すると、パフォーマンスが向上します。標準の IIS インストールでは、デフォルトでこのフィルタがオンになっています。フィルタをオフにするには、次の手順に従います。
  1. IIS マネージャ・コンソールを起動します。
  2. **[既定の Web サイト]** のプロパティを編集します。
  3. **[ISAPI フィルタ]** タブで、ASP.NET フィルタを削除します。
- パフォーマンスを向上させるために、IIS アクセス・ログをオフにすることができます。アクセス・ログをオフにするには、次の手順に従います。
  1. IIS マネージャ・コンソールを起動します。
  2. **[既定の Web サイト]** で *ias_relay_server* ディレクトリのプロパティを編集します。

3. **[ディレクトリ]** タブで、**[ログ アクセス]** セクションをクリアします。

- 運用環境では、リレー・サーバ設定ファイルでリレー・サーバの冗長性を 0 に設定できます。これにより、高負荷時のパフォーマンスが向上します。
- リレー・サーバには、Web ガーデン・サイズに関する制限がありません。1 つのワーカ・プロセスがすべての Outbound Enabler とすべてのクライアントからの要求を処理できます。ただし、プロセスで作成できるスレッド数は、スレッドの作成に使用できる残りのプロセス・ヒープ領域によって制限されます。IIS によって作成されるスレッドのスタック・サイズは、256 K です。マシンに十分なリソースがあり、サーバが数千もの同時要求による負荷を受けているときに同時実行の制限に達するおそれがある場合は、プロセス数を増やすことを試してみてください。

Linux 上の Apache へのリレー・サーバ・コンポーネントの配備

Linux 用のリレー・サーバは、次の実行ファイルから構成されます。

- *mod_rs_ap_client.so*
- *mod_rs_ap_server.so*
- *rshost*
- *dblgen11.res*
- *libdbtasks11_r.so*
- *libdbicudt11.so*
- *libdbicu11_r.so*
- *libdblib11_r.so*
- *dbsupport*
- *dbfhide*

参照

- 「リレー・サーバ・ステイト・マネージャ」 259 ページ
- 「ファイル難読化ユーティリティ (dbfhide)」 258 ページ

Web 拡張機能ファイルとステイト・マネージャの配備

♦ **リレー・サーバ・ファイルを配備するには、次の手順に従います。**

1. 上記のファイルを Apache インストールの *modules* ディレクトリにコピーします。
2. リレー・サーバ設定ファイル *rs.config* を作成します。「リレー・サーバ設定ファイル」 251 ページを参照してください。
3. *rs.config* を *modules* ディレクトリにコピーします。サーバ・モジュールは、*rshost* 実行プログラムが *rs.config* ファイルをコピーしたディレクトリと同じディレクトリにあることを想定します。

4. PATH 環境変数と LD_LIBRARY_PATH 環境変数を、Apache の *modules* ディレクトリを含むように設定します。
5. Apache の *conf/httpd.conf* ファイルを編集します。
   a. リレー・サーバのクライアント・モジュールとサーバ・モジュールをロードするための、次の行を追加します。

      ```
      LoadModule iarelayserver_client_module modules/mod_rs_ap_client.so

      LoadModule iarelayserver_server_module modules/mod_rs_ap_server.so
      ```

      クライアント・モジュールとサーバ・モジュールは、異なる URL を使用して呼び出されます。クライアント・モジュールは、URL パスで *iarelayserver* 文字列を明示的に指定して検索します。URL のその部分を変更する必要はありません。

   b. クライアント・モジュールの <*location*> セクションを作成するための、次の行を追加します。

      ```
      <LocationMatch /cli/iarelayserver/* >
         SetHandler iarelayserver-client-handler
      </LocationMatch>
      ```

   c. サーバ・モジュールの <*location*> セクションを作成するための、次の行を追加します。

      ```
      <Location /srv/iarelayserver/* >
         SetHandler iarelayserver-server-handler
         RSConfigFile "/<apache-install>/modules/rs.config"
      </Location>
      ```

      リレー・サーバ設定ファイル *rs.config* の場所を指定する RSConfigFile ディレクティブを指定してください。*rs.config* ファイルは、rshost 実行プログラムが配備されているディレクトリと同じディレクトリに存在している必要があります。

   d. TimeOut ディレクティブが設定されている場合は、60 秒以上に設定されていることを確認します。
6. Linux の場合、Apache がプロセスを生成するときに $TMP、$TMPDIR、$TEMP のいずれかの環境変数がグローバルに設定されている場合、Apache をこれ以上設定する必要はありません。

   これらの環境変数のいずれもグローバルに設定されていない場合、またはデフォルトのリレー・サーバ・ログ・ファイルを特定のテンポラリ・ディレクトリに保存する場合は (たとえばカスタマイズしないでステイト・マネージャを自動的に起動する場合)、ファイル */<apache-dir>/bin/envvars* を編集して TMP を設定し、エクスポートします。

   たとえば、envvars ファイルで $TMP を編集する場合は、次のようにします。

   ```
   set TMP="/tmp"
   export TMP
   ```

   これにより、Apache がプロセスを作成する前に、Apache が作成するシェル内で環境変数が設定されます。

注意
どのような場合も、Apache ユーザ・プロセスには選択した *tmp* ディレクトリのロケーションに対する書き込みパーミッションが必要です。

リレー・サーバ・ファーム設定の更新

リレー・サーバ・ファーム設定は、リレー・サーバ設定ファイルの内容によって定義されます。リレー・サーバ・ファーム内の各リレー・サーバは、同じリレー・サーバ設定ファイルを共有します。そのため、リレー・サーバ・ファーム設定を更新するときは、ファーム内のリレー・サーバごとにリレー・サーバ設定ファイルを更新する必要があります。更新には、次の操作が含まれます。

- 新しいリレー・サーバをリレー・サーバ・ファームに追加する。
- 新しいバックエンド・サーバ・ファームを作成して、リレー・サーバ・ファームへのアクセスを許可する。
- 新しいバックエンド・サーバを既存のバックエンド・サーバ・ファームに追加する。
- リレー・サーバ、バックエンド・サーバ・ファーム、またはバックエンド・サーバのプロパティを変更する。
- オプションを変更する。

リレー・サーバ設定を更新する 1 つの方法は、すべてのリレー・サーバをシャットダウンし、リレー・サーバ設定ファイルを更新したバージョンで置き換えて、すべてのリレー・サーバを再起動することです。ただし、リレー・サーバをシャットダウンして再起動するということは、リレー・サーバのユーザがサービスの中断を被る可能性があります。

リレー・サーバ設定を更新する際の推奨方法は、リレー・サーバ・ファームが稼働している状態でサービスを中断せずに、リレー・サーバ・ステイト・マネージャを使用して設定を更新することです。

リレー・サーバ設定を更新するには、次のコマンド・ライン・フォーマットを使用して、リレー・サーバ・ステイト・マネージャの新しいインスタンスを起動します。

```
rshost -u -f <filename>
```

-u オプションは、更新操作を実行することをリレー・サーバ・ステイト・マネージャに指示します。-f オプションは、更新された設定が含まれる設定ファイルの名前を指定します。「リレー・サーバ・ステイト・マネージャ」 259 ページを参照してください。

次に、リレー・サーバ・ファーム設定を更新するために必要な手順の概要を示します。

1. リレー・サーバ設定ファイルのマスタ・コピーに変更を加えます。
2. 更新中のリレー・サーバ・ファームに属し、リレー・サーバのインスタンスを実行しているコンピュータごとに、次の操作を行います。
   a. 古い設定ファイルを更新した設定ファイルで置き換えます。
   b. 更新された設定ファイルを使用してリレー・サーバ・ステイト・マネージャを実行します。

Windows 上の IIS 用のリレー・サーバ設定の更新

♦ **Windows 上の IIS 用のリレー・サーバ設定を更新するには、次の手順に従います。**

1. 更新中のリレー・サーバ・ファームに属するコンピュータごとに、更新した設定ファイルをリレー・サーバ Web サイトのホーム・ディレクトリにある *ias_relay_server¥server* ディレクトリにコピーします。自動起動を使用する場合は、設定ファイルを *rs.config* という名前にしてください。
2. *ias_relay_server¥server* ディレクトリから、次のコマンドを実行して設定の更新を適用します。

   ```
   rshost -u -f rs.config
   ```
3. 更新中のリレー・サーバ・ファーム内のコンピュータごとに、前述の手順を繰り返します。

Linux 上の Apache 用のリレー・サーバ設定の更新

♦ **Linux 上の Apache 用のリレー・サーバ設定を更新するには、次の手順に従います。**

1. 更新した設定ファイルを Apache インストール・ディレクトリにある */modules* ディレクトリにコピーします。自動起動を使用する場合は、設定ファイルを *rs.config* という名前にしてください。
2. */<Apache-install>/modules* ディレクトリから、次のコマンド・ラインを実行して設定の更新を適用します。

   ```
   rshost -u -f rs.config
   ```
3. 更新中のリレー・サーバ・ファーム内のコンピュータごとに、前述の手順を繰り返します。

Sybase リレー・サーバのホスティング・サービス

Sybase リレー・サーバのホスティング・サービスは、Sybase をホストとするリレー・サーバのファームです。Mobile Link データ同期を使用するモバイル・アプリケーションの開発を容易にすること、また特に公共無線ネットワークを使用してデータを送信する場合に開発者による評価プロセスを簡素化することを目的としています。何かをインストールしたり企業ファイアウォールに穴を開けたりするように IT 部門に依頼する必要はありません。Mobile Link とホスティング・サービスの間の通信はすべて、Mobile Link 側で開始されたアウトバウンド接続を介した HTTP(S) を使用して行われます。

Sybase リレー・サーバのホスティング・サービスは、運用環境への配備を目的としていません。運用アプリケーションを配備する前に、自社のインフラストラクチャにリレー・サーバをインストールしておく必要があります。

リレー・サーバのホスティング・サービスの使用

以降の項では、基本的な作業を実行する方法について説明します。

リレー・サーバのホスティング・サービスのサブスクライブ

Sybase リレー・サーバのホスティング・サービスを使用するには、先にサブスクライブする必要があります。

♦ Sybase リレー・サーバのホスティング・サービスをサブスクライブするは、次の手順に従います。

1. Web ブラウザから http://relayserver.sybase.com/account にアクセスします。これにより、Sybase リレー・サーバのホスティング・サービスのホーム・ページが開きます。
2. **[Register]** をクリックしてアカウントを作成します。
3. **[Subscription ID]** と **[Password]** を指定し、自分および組織の連絡先情報を入力し、**[Hosted Relay Service Terms of Service]** に同意するように求められます。[Subscription ID] には組織でユニークな ID を選択してください。**[Submit]** をクリックします。

   登録に成功したら、登録確認の電子メールが送られてきます。

リレー・サーバのホスティング・サービスへのログイン

♦ リレー・サーバのホスティング・サービスにログインするには、次の手順に従います。

1. **[Log In]** をクリックして新しく作成したアカウントにログインします。
2. 登録処理中に入力した **[Subscription ID]** と **[Password]** を入力します。ログインすると、**[Account Information]** ページが開きます。アカウント情報のページでは、サブスクライバ情報を変更したり、このサービスにアクセスするバックエンド・サーバ・ファームを指定したりすることができます。

サーバ・ファームの追加

♦ **サーバ・ファームを追加するには、次の手順に従います。**

1. **[Add New Farm]** をクリックします。
2. サーバ・ファームを説明するユニークな **[Farm Name]** を入力します。
3. ドロップダウン・リストから **[Type]** を選択します。
4. ファーム内の各サーバに対してユニークな名前を指定します。最大で 2 つのサーバを指定できます。
5. **[Create Farm]** をクリックします。ファームが正常に追加された場合は確認が表示されます。
6. **[Configuration Instructions]** をクリックして、サービスの使用方法の詳細を確認します。表示される指示内容は、入力した情報によって異なります。
7. 完了したら、**[Log Out]** をクリックします。

リレー・サーバと Mobile Link の使用

以降の項では、リレー・サーバと Mobile Link を一緒に使用する場合について説明します。

リレー・サーバでサポートされるオペレーティング・システムとブラウザについては、http://www.ianywhere.jp/sas/os.html を参照してください。

Outbound Enabler の配備の詳細については、「Mobile Link サーバの配備」 849 ページを参照してください。

クライアントからリレー・サーバ・ファームへの接続

リレー・サーバ・ファームが適切に設定されたら、クライアントは次の URL を使用してリレー・サーバ・ファームに接続します。

http://<*Relay Server client extension URI*>/<*farm*>

オプション

オプション	説明
<*Relay Server client extension URI*>	Windows 上の IIS の場合、<*domain name*>/*ias_relay_server/client/rs_client.dll* です。 Linux 上の Apache の場合、<*domain name*>/*cli/iarelayserver* です。
<*farm*>	リレー・サーバがクライアント要求を転送するバックエンド・ファーム (バックエンド・サーバのグループ) を識別します。

SQL Anywhere Mobile Link クライアントの接続の例

サーバ・ファーム **F1** に接続するには、SQL Anywhere Mobile Link クライアントで次のオプションを指定する必要があります。

```
-e "ctp=http;
    adr='host=relayserver.sybase.com;
      url_suffix=/ias_relay_server/client/rs_client.dll/F1'"
```

HTTPS の場合は、http を https に変更します。

Ultra Light／Ultra Light J Mobile Link クライアントの接続の例

サーバ・ファーム **F1** に接続するには、Ultra Light／Ultra Light J Mobile Link クライアントで ULSyncParms クラスの次のプロパティを設定する必要があります。

- ストリーム・タイプを HTTPS に設定します。
- ストリーム・パラメータを次のように設定します。


```
"host=testrelay.ianywhere.com; url_suffix=/ias_relay_server/client/rs_client.dll/F1"
```

QAnywhere クライアントの接続の例

サーバ・ファーム **F1** に接続するには、QAnywhere クライアントで次のオプションを指定する必要があります。

```
-x "http(host=relayserver.sybase.com;url_suffix=/ias_relay_server/client/rs_client.dll/F1"
```

サンプル・シナリオ

ABC 社がモバイル・アプリケーションの開発を完了し、モバイル・アプリケーションにサービスを提供する配備ランタイムを設定したいと仮定します。モバイル配備は、最初は 10000 個のデバイスから構成され、将来さらに増加します。したがって、顧客は、現在の負荷を処理でき、将来より多くのモバイル配備を処理できるように拡張しやすい、フォールト・トレラントで負荷分散された環境を要求しています。モバイル・アプリケーションのデータ同期特性に基づいて、顧客は次のような設定が必要であると判断しました。

- 2 つの Mobile Link サーバ
- 2 つのリレー・サーバ
- 1 つの負荷分散装置

同社は Web サーバとして IIS を使用しているため、リレー・サーバの IIS バージョンが使用されます。

注意

- 各リレー・サーバは、専用のコンピュータ上に配備されます。ホスト名が **rs1.abc.com** と **rs2.abc.com** の 2 つのコンピュータを使用します。
- 各 Mobile Link サーバは、専用のコンピュータ上に配備されます。2 つの Mobile Link サーバは、**ml1** と **ml2** という名前が割り当てられ、abc.mobilink という名前のバックエンド・サーバ・ファームに属しています。
- 負荷分散装置は、ホスト名 **www.abc.com** を使用して指定できます。
- 最高レベルのセキュリティを得るため、リレー・サーバに接続するすべてのクライアントと Outbound Enabler では HTTPS が使用されます。すべての Web サーバには既知の認証局 (CA) による証明書が配置され、すべてのバックエンド・サーバ・コンピュータには対応する信頼できるルート証明書が標準の証明書ストアに保存されていると仮定します。

♦ リレー・サーバ・ファームを設定するには、次の手順に従います。

1. 初めに、リレー・サーバ設定ファイルを作成します。

   設定を含むファイルの名前は、***rs.config*** にしてください。このシナリオでは、次の設定ファイルを使用しています。

```
#
# Options
#
```

```
[options]
verbosity = 1

#
# Define the Relay Server farm
#
[relay_server]
host = rs1.abc.com

[relay_server]
host = rs2.abc.com

#
# Define the MobiLink backend server farm
#
[backend_farm]
id = abc.mobilink
client_security = on
backend_security = on

#
# List MobiLink servers that are connecting to the Relay Server farm
#
[backend_server]
farm = abc.mobilink
id = ml1
token = mltoken1

[backend_server]
farm = abc.mobilink
id = ml2
token=mltoken2
```

2. リレー・サーバを実行している 2 つのコンピュータに、リレー・サーバ・コンポーネントとともに設定ファイル *rs.config* を配備します。
3. Mobile Link サーバを実行する 2 つのコンピュータで、Mobile Link サーバを起動し、次のコマンドを使用して対応する Outbound Enabler を起動します。

   ID が ml1 の Mobile Link サーバを実行しているコンピュータで、次のコマンドを実行します。

   ```
   mlsrv11 -x http -z ml1 -ss <other ML options>
   rsoe -f abc.mobilink -id ml1 -t mltoken1 -cr "host=www.abc.com;port=443;https=1"
   ```

   ID が ml2 の Mobile Link サーバを実行しているコンピュータで、次のコマンドを実行します。

   ```
   mlsrv11 -x http -z ml2 -ss <other ML options>
   rsoe -f abc.mobilink -id ml2 -t mltoken2 -cr "host=www.abc.com;port=443;https=1"
   ```

4. すべてのサーバと Outbound Enabler が実行中になったら、Mobile Link クライアントは次の接続情報を使用してファームに接続できます。
   - **HTTPS** protocol
   - **host** www.abc.com
   - **url_suffix** /ias_relay_server/client/rs_client.dll/abc.mobilink

リダイレクタ (旧式)

目次

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

リダイレクタの概要 (旧式)

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

Mobile Link には、「**リダイレクタ**」と呼ばれる Web サーバの拡張機能が組み込まれています。これによって、クライアントと Mobile Link サーバ間の要求や応答のルートが指定されます。このようなプラグインは、一般に「**リバース・プロキシ**」とも呼ばれます。

Web サーバ経由で要求を送信する主な理由は、HTTP または HTTPS 同期用の、Web サーバとファイアウォールの既存の設定を利用するためです。ただし、Web サーバはリダイレクタがなくてもプロキシとして動作できます。リダイレクタは、複数の Mobile Link サーバが稼働している場合に最も便利です。

「Web サーバを使用した場合のオプション」 277 ページを参照してください。

リダイレクタを使用すると、Mobile Link サーバが稼働する 1 台以上のコンピュータに特定の URL 要求をルート指定するように、Web サーバを設定できます。

Web サーバは、特定の URL または特定の範囲の URL からの要求を拡張プログラム (通常、Perl 言語の CGI スクリプト、DLL、またはその他の拡張メカニズムで記述されている) に渡すように設定できます。これらの拡張プログラムは、外部データ・ソースにアクセスしたり、Web サーバがクライアントに配信する応答を提供したりできます。

負荷分散とフェールオーバ

リダイレクタは、単純なラウンド・ロビン・アルゴリズム (固定の循環的順序でサーバを選択) を使用して、負荷分散とフェールオーバを実装しています。各 Mobile Link サーバに対する ping を実行し、応答しないサーバへの要求の送信を停止します。Mobile Link サーバが再び稼働するとそれを検出し、そのときに要求の送信を再開します。

HTTPS 同期

Mobile Link クライアントに HTTPS プロトコルを指定すると、リモート・データベースと Web サーバ間の接続には HTTPS が使用されます。つまり、HTTP ヘッダは、RSA 暗号化を使用して TLS で暗号化されてから、Web サーバとの間で送受信が行われます。Web サーバは、HTTPS を復号化し、リダイレクタ経由で Mobile Link に HTTP を送信します。すべてのリダイレクタは、Mobile Link クライアントと Web サーバとの接続に対してのみ使用される、このバージョンの HTTPS をサポートしています。

HTTPS プロトコルは、その他の安全なプロトコルよりも低速です。

完全 HTTPS

一部のリダイレクタ (Apache リダイレクタ、ISAPI リダイレクタ、Windows 用 NSAPI リダイレクタ) には、ストリームを HTTPS として再暗号化して Mobile Link サーバに送信するオプションが用意されています。

リダイレクタから Mobile Link サーバへの HTTPS 通信がサポートされているリダイレクタのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

サポートされる Web サーバ

プラグインは、次の Web サーバで使用できます。

リダイレクタ・プラグイン	サポート対象
ISAPI リダイレクタ	Microsoft Web サーバ
NSAPI リダイレクタ	Windows と UNIX 上の Sun One (以前の Netscape) および iPlanet Web サーバ
サーブレット・リダイレクタ	Apache Tomcat、UNIX 上の Sun One Web サーバを含む、Java Servlet API 2.3 をサポートする Web サーバ
ネイティブな Apache リダイレクタ	Apache Web サーバ
M-Business Anywhere リダイレクタ	M-Business Anywhere Web サーバ

リダイレクタのサポートの詳細については、次を参照してください。

- http://www.ianywhere.jp/sas/os.html

Web サーバを使用した場合のオプション

リダイレクタは、Web サーバ経由で Mobile Link 同期をルート指定する 1 つの方法です。リダイレクタは、ファイアウォール経由の同期や、複数の Mobile Link サーバとの同期で特に役立ちます。

Web サーバ経由で要求を送信する主な理由は、HTTP または HTTPS 同期用の、Web サーバとファイアウォールの既存の設定を利用するためです。リダイレクタは、複数の Mobile Link サーバが稼働している場合に最も便利です。

また、リダイレクタを使用せずに、Web サーバを経由して同期をルート指定することもできます。この場合は、Web サーバをプロキシとして設定して、Mobile Link サーバに同期をルート指定します。Web サーバでルート指定を行う方法の詳細については、Web サーバのマニュアルを参照してください。

次の表は、Mobile Link 同期の最適なルート指定方法を判断する上での推奨事項を示しています。

	直接接続が可能	直接接続が不可能
One MobiLink server	HTTP の代わりに TCP/IP を使用する	Web サーバを経由し、Mobile Link サーバにメッセージを送信するために HTTP または HTTPS プロキシを使用する

	直接接続が可能	直接接続が不可能
Multiple MobiLink servers	HTTP または HTTPS でリダイレクタを使用する	HTTP または HTTPS でリダイレクタを使用する

「リダイレクタ (旧式)」 275 ページを参照してください。

リダイレクタの設定

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

次の項では、同期要求を管理するための Web サーバの設定方法について説明します。

♦ 設定処理の概要

1. Mobile Link サーバを設定します。

   「Mobile Link クライアントとサーバのリダイレクタ設定」 280 ページを参照してください。

2. リダイレクタ設定ファイルを修正します。使用しているリダイレクタがサーバ・グループをサポートしているかどうかに応じて、手順が異なります。次の項を参照してください。

   - 「リダイレクタのプロパティの設定 (サーバ・グループをサポートするリダイレクタの場合)」 284 ページ
   - 「リダイレクタのプロパティの設定 (サーバ・グループをサポートしないリダイレクタの場合)」 286 ページ

3. Web サーバ固有の設定を行います。

   次のうちの 1 つを参照してください。

   - 「Windows 上の Netscape/Sun Web サーバ用の NSAPI リダイレクタ (旧式)」 289 ページ
   - 「Microsoft Web サーバ用の ISAPI リダイレクタ (旧式)」 294 ページ
   - 「サーブレット・リダイレクタ (旧式)」 296 ページ
   - 「Apache リダイレクタ (旧式)」 299 ページ
   - 「M-Business Anywhere リダイレクタ (旧式)」 302 ページ

4. Mobile Link クライアントを設定します。

   「Mobile Link クライアントとサーバのリダイレクタ設定」 280 ページを参照してください。

Mobile Link クライアントとサーバのリダイレクタ設定

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

この項では、Mobile Link クライアントと Mobile Link サーバを、Web サーバ経由で同期するように設定する方法について説明します。次の手順では、Web サーバ経由で送信される要求に必要なパラメータを設定します。

Mobile Link クライアント

♦ **Mobile Link クライアント(SQL Anywhere と Ultra Light) を設定するには、次の手順に従います。**

1. Mobile Link クライアントの通信タイプを、HTTP または HTTPS に設定します。

   SQL Anywhere クライアントの通信タイプの設定の詳細については、「CommunicationType (ctp) 拡張オプション」 『Mobile Link - クライアント管理』を参照してください。

   Ultra Light クライアントの通信タイプの設定の詳細については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」 『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。

2. Mobile Link クライアントに、次の HTTP/HTTPS 同期プロトコル・オプションを指定します。
   - **host**　Web サーバの名前または IP アドレス
   - **port**　HTTP 要求または HTTPS 要求を受け付ける Web サーバのポート
   - **url_suffix**　この設定は、使用しているリダイレクタのタイプによって異なります。
     - ISAPI リダイレクタの場合

       ```
       exe_dir/iaredirect.dll/ml/[server-group/]
       ```

       exe_dir は *iaredirect.dll* のロケーション、*server-group* (省略可能) はグループ名です。
     - NSAPI リダイレクタの場合

       ```
       mlredirect/ml/[server-group/]
       ```

       mlredirect は *obj.conf* ファイル内でマッピングされている名前です。
     - サーブレット・リダイレクタの場合

       ```
       iaredirect/ml/
       ```
     - Apache 用のネイティブなリダイレクタの場合、*httpd.conf* ファイル内のリダイレクタの <location> タグで指定した内容がそのまま設定されます。たとえば、ロケーションが <Location /iaredirect/ml> の場合、url_suffix は次のようになります。

       ```
       iaredirect/ml/
       ```


○ M-Business Anywhere リダイレクタの場合は、*sync.conf* ファイル内のリダイレクタの <location> タグで指定した内容がそのまま設定されます。たとえば、ロケーションが <Location /iaredirect/ml> の場合、url_suffix は次のようになります。

```
iaredirect/ml/
```

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

Ultra Light クライアントのプロトコル・オプションの設定の詳細については、「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『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。

SQL Anywhere クライアントのプロトコル・オプションの設定の詳細については、「Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧」『Mobile Link - クライアント管理』を参照してください。

Mobile Link サーバ

♦ **Mobile Link 同期サーバを設定するには、次の手順に従います。**

1. HTTPS をサポートするリダイレクタの場合は、HTTPS プロトコルを使用して Mobile Link サーバを起動できます。HTTPS をサポートしているリダイレクタのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

   HTTPS をサポートしないリダイレクタの場合、クライアントとプロキシとの間の通信に HTTP または HTTPS を使用するには、HTTP プロトコルを指定して Mobile Link サーバを起動する必要があります。このようなリダイレクタで HTTPS を直接使用することはできません。

   たとえば、HTTP プロトコルを指定するには、mlsrv11 コマンド・ラインで次のように入力します。

   ```
   mlsrv11 -x http
   ```

   「-x オプション」 110 ページを参照してください。

2. また、Mobile Link サーバに対して、次のパラメータを設定できます。

   ● **port**　Mobile Link は、HTTP プロトコルのデフォルトとしてポート番号 80 を使用し、HTTPS プロトコルのデフォルトとしてポート番号 443 を使用します。Mobile Link サーバと Web サーバが同じコンピュータ上で稼働している場合、ポート番号 80 は通常、Web サーバによって使用されます。この場合、別のポート番号を指定してください。たとえば、ポート番号 2439 を使用できます。これは、Internet Assigned Numbers Authority (IANA) に登録された Mobile Link サーバ用のポート番号です。

   ポートの詳細については、「-x オプション」 110 ページを参照してください。

リダイレクタのプロパティの設定

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

リダイレクタのプロパティは、サーバ・グループをサポートするものとサポートしないものとで異なります。

Mobile Link サーバ・グループ

Mobile Link サーバをいくつかのグループに分割できます。これにより、Mobile Link サーバで構成された個々のグループにアクセスするクライアントのグループを作成できます。

サーバ・グループをサポートしているリダイレクタのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

サーバ・グループを作成するには、リダイレクタ設定ファイル (*redirector_server_group.config*) に、角カッコで囲んだグループ名に続いてグループの設定を指定したセクションを追加します。グループには、少なくとも ML ディレクティブを 1 つ指定する必要があります。また、ML_CLIENT_TIMEOUT オプションをグループに設定することもできます。作成したグループは、クライアントの url_suffix オプションで参照します。

デフォルトのサーバ・グループを作成するには、設定ファイル内で、名前付きのグループの前に名前のないグループを指定します。デフォルト・グループは、下位互換性を保つために役立ちます。このグループは、クライアントが url_suffix オプションで特定のサーバ・グループ名を指定しなかった場合に使用されます。

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

SLEEP と LOG_LEVEL プロパティについては、すべてのサーバ・グループで使用するデフォルト設定を指定できます。これは、設定ファイルのどこに指定してもかまいません。

古いクライアントと新しいクライアントのサポート

Mobile Link サーバで、バージョン 8 または 9 のリモート・データベースと、バージョン 10 以降のリモート・データベースをともにサポートする必要がある場合は、少なくともポートを 2 つ開く必要があります。mlsrv11 -x オプションを使用して新しいクライアント用のポートを開き、mlsrv11 -xo オプションを使用して古いクライアント用のポートを開きます。リダイレクタも使用している場合は、リダイレクタによってクライアントが適切なポートに割り振られるように、サーバ・グループを設定する必要があります。

一般的なリダイレクタ設定では、複数の Mobile Link サーバを起動します。最も単純なケースでは、-x と -xo を指定して 2 つのポートが開かれている単一の Mobile Link サーバ構成において、各ポートで使用する 2 つのサーバ・グループを作成します。次に、Mobile Link サーバで使用する 2 つのポートを開くための mlsrv11 コマンド・ラインの一部分を示します。

```
mlsrv11 -c "dsn=YourDSN" -x http(port=111) -xo http(port=222)
```


作成した 2 つのサーバ・グループについて、リダイレクタ設定ファイルに次のセクションを追加します。

```
[v10service]
   ML="host=mySrv.myCorp.com;port=111"
[v9service]
   ML="host=mySrv.myCorp.com;port=222"
```

クライアントを起動するときは、url_suffix オプションにサーバ・グループ名を指定します。たとえば、SQL Anywhere クライアントと ISAPI Web サーバの場合、バージョン 10 クライアント用の dbmlsync コマンド・ラインの一部分は次のようになります。

```
dbmlsync -e "adr='host=somehost;port=5001;url_suffix=scripts/iaredirect.dll/ml/v10service'"...
```

バージョン 9 クライアント用の dbmlsync コマンド・ラインの一部分は次のようになります。

```
dbmlsync -e "adr='host=somehost;port=5001;url_suffix=scripts/iaredirect.dll/ml/v9service'"...
```

参照

- 「-x オプション」 110 ページ
- 「-xo オプション」 117 ページ
- 「リダイレクタのプロパティの設定 (サーバ・グループをサポートするリダイレクタの場合)」 284 ページ
- 「url_suffix」 『Mobile Link - クライアント管理』

例

次に、*redirector_server_group.config* ファイルの例を示します。ここには、一般的な設定やサーバ・グループの作成が示されています。

```
#
# Set up the default server group:
#
  ML="host=mySrv1.myCorp.com;port=222"
  ML="host=mySrv2.myCorp.com;port=222"
#
# Set up a server group named myOldGroup:
#
[myOldGroup]
  ML="host=myOldSrv1.myCorp.com;port=111"
  ML="host=myOldSrv2.myCrop.com;port=111"
  ML_CLIENT_TIMEOUT=30
#
# Set up a server group named myNewGroup:
#
[myNewGroup]
  ML="host=myNewSrv1.myCorp.com;port=333"
  ML="host=myNewSrv2.myCorp.com;port=555"
  ML_CLIENT_TIMEOUT=240
#
# Set up a server group named mlSecureGroup:
#
[theirSecureGroup]
  ML="https=true;Srv1.Corp.com;trusted_certificates=c:¥Corp¥publicRoot.crt"
  ML="https=true;Srv2.Corp.com;trusted_certificates=c:¥Corp¥publicRoot.crt"
#
# Set global properties:
#
```

```
LOG_LEVEL=5
SLEEP=15
```

リダイレクタのプロパティの設定 (サーバ・グループをサポートするリダイレクタの場合)

この項では、リダイレクタのプロパティを設定するための、Web サーバに共通の設定手順について説明します。ここでの説明に該当するのは、サーバ・グループをサポートするリダイレクタです。

サーバ・グループをサポートしているリダイレクタのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

サーバ・グループの詳細については、「Mobile Link サーバ・グループ」 282 ページを参照してください。

♦ **リダイレクタのプロパティを設定するには、次の手順に従います。**

1. 「Mobile Link クライアントとサーバのリダイレクタ設定」 280 ページの手順を完了します。
2. **リダイレクタ設定ファイル**を設定します。テンプレート・ファイル *redirector_server_group.config* は、*install-dir¥MobiLink¥Redirector* にあります。リダイレクタ設定ファイルを設定する最も簡単な方法は、*redirector_server_group.config* を編集することです。

   リダイレクタ設定ファイルには、以下の規則を適用します。

   - 1 行の長さの最大値は 2000 文字
   - コメントはハッシュ文字 (#) で開始する
   - ISAPI リダイレクタの場合、設定ファイル名を *redirector.config* とし、*iaredirect.dll* と同じディレクトリに置く必要があります。

   このファイルでは、次のディレクティブを設定できます。

   - **サーバ・グループ**　サーバ・グループを作成するには、*redirector_server_group.config* に角カッコで囲まれたサーバ・グループ名で始まるセクションを作成して、サーバ・グループを定義します。

     「Mobile Link サーバ・グループ」 282 ページを参照してください。

   - **LOG_LEVEL**　ログ・ファイルに書き込まれる出力の量を設定するときに使用します。値は 0 ～ 7 で、値が大きいほど生成される出力が増えます。デフォルトでは、ログ・ファイル名は *redirector.log* で、リダイレクタ設定ファイルと同じ場所に置かれます。NSAPI リダイレクタの場合は、magnus.conf に指定されている名前とロケーションを logFile ディレクティブを使用して変更できます。
   - **ML**　ML ディレクティブを使用するには次の 2 つの方法があります。
     - リダイレクタから Mobile Link サーバへの HTTPS 通信をサポートしていないリダイレクタの場合、または HTTPS を使用していない場合は、ML ディレクティブを使用して、Mobile Link サーバを実行しているコンピュータのリストを ML=host:port の形式

で指定できます。複数のコンピュータを指定するには、改行してこの構文を繰り返し指定します。次に例を示します。

```
ML=209.123.123.1:8080
ML=myCompany.com:8081
```

○ リダイレクタから Mobile Link サーバへの HTTPS 通信をサポートしているリダイレクタで HTTPS を使用している場合は、次のようなセミコロン区切りのリストで Mobile Link クライアントのネットワーク・プロトコル・オプションを指定する必要があります。

ML="https=true;*network-client-options*;..."

次に例を示します。

```
ML="https=true;host=My-pc;port=82;trusted_certificates=rsaroot.crt"
```

ネットワーク・クライアント・オプションのリストについては、「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』を参照してください。

リダイレクタでの HTTPS サポートの詳細については、「完全 HTTPS」 276 ページを参照してください。

リダイレクタから Mobile Link サーバへの HTTPS 通信がサポートされているリダイレクタのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

Mobile Link サーバは、ML ディレクティブに指定されているものと同じプロトコルとポート番号を使用して起動する必要があります。異なる場合は、Mobile Link サーバを停止して、正しい指定で再起動してください。

- **ML_CLIENT_TIMEOUT** 同じリモート・データベースからの同期の重複を Mobile Link サーバに確実に検出させるために使用します。このタイムアウトは、同じサーバ・グループを使用しているクライアントの中で最も長いタイムアウトに設定する必要があります。このプロパティを 0 に設定すると、別のサーバに対する再同期がただちに許可されます。デフォルト値は 240 秒です。
- **SLEEP** サーバが機能していることをリダイレクタがチェックする間隔を秒単位で設定するときに使用します。リダイレクタは、あるサーバをチェックし、このオプションで設定された時間だけ待機してから、次のサーバをチェックする、というサイクルを続けます。たとえば SLEEP=10 のようになります。SLEEP では大文字と小文字を区別します。デフォルトは 20 秒です。

3. リダイレクタ設定ファイルを Web サーバにコピーします。

Mobile Link サーバが Web サーバと同じコンピュータにインストールされていない場合には、Web サーバが設定されているコンピュータ (またはそのコンピュータがアクセスできるドライブ) にリダイレクタ設定ファイルをコピーしてください。

ISAPI Web サーバの場合は、リダイレクタ設定ファイルを *Inetpub¥scripts* ディレクトリにコピーし、名前を *redirector.config* に変更します。

その他の Web サーバの場合、リダイレクタ設定ファイルは任意のディレクトリにコピーできます。

4. 次のいずれかの項で Web サーバ固有の設定を完了します。
   - 「Microsoft Web サーバ用の ISAPI リダイレクタ (旧式)」 294 ページ
   - 「Windows 上の Netscape/Sun Web サーバ用の NSAPI リダイレクタ (旧式)」 289 ページ

例

次に、リダイレクタ設定ファイルの例を示します。このファイルで指定されている内容は、以下のとおりです。

- リダイレクタは、サーバが機能していることをチェックしたら、10 秒間スリープする。
- Mobile Link サーバを稼働しているコンピュータで、要求を処理できるコンピュータは 3 台。

```
SLEEP=10
ML=myServ-pc:80
ML=209.123.123.1:8080
ML=myCompany.com:8081
```

リダイレクタのプロパティの設定 (サーバ・グループをサポートしないリダイレクタの場合)

この項では、リダイレクタのプロパティを設定するための、Web サーバに共通の設定手順について説明します。ここでの説明に該当するのは、サーバ・グループをサポートしないリダイレクタです。

サーバ・グループをサポートしているリダイレクタのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

♦ **リダイレクタのプロパティを設定するには、次の手順に従います。**

1. 「Mobile Link クライアントとサーバのリダイレクタ設定」 280 ページの手順を完了します。
2. *redirector.config* を Web サーバにコピーします。

   ファイル *redirector.config* は、*install-dir¥MobiLink¥Redirector* にあります。

   Mobile Link サーバが Web サーバと同じコンピュータにインストールされていない場合には、Web サーバが設定されているコンピュータに *redirector.config* をコピーしてください。
3. リダイレクタ設定ファイルを設定します。

   Web サーバと Mobile Link サーバ間の通信を設定するには、Web サーバがインストールされているコンピュータの *redirector.config* ファイルを編集してください。

   redirector.config には、以下の規則を適用します。
   - 1 行の長さの最大値は 300 文字
   - コメントはハッシュ文字 (#) で開始する
   - ディレクティブを定義するときには、スペースやタブを使用しない

   このファイルでは、次のディレクティブを設定できます。

- **LOG_LEVEL**　ログ・ファイルに書き込まれる出力の量を設定するときに使用します。値は 0、1、2 です。デフォルトは 1 で、2 に設定すると最大量の出力が生成されます。Apache リダイレクタの場合、この設定は何も影響しません。Apache 設定ファイル *httpd.conf* の LogLevel セクションで、ログ・レベルを設定してください。
- **ML**　ML では大文字と小文字を区別します。ML ディレクティブを使用するには次の 2 つの方法があります。

  HTTPS をサポートしていないリダイレクタの場合、または HTTPS を使用していない場合は、ML ディレクティブを使用して、Mobile Link サーバを実行しているコンピュータのリストを ML=host:port の形式で指定できます。複数のコンピュータを指定するには、改行してこの構文を繰り返し指定します。次に例を示します。

  ```
  ML=209.123.123.1:8080
  ML=myCompany.com:8081
  ```

  リダイレクタから Mobile Link サーバへの HTTPS 通信をサポートしているリダイレクタでは、次のようなセミコロン区切りのリストで Mobile Link クライアントのネットワーク・プロトコル・オプションを指定する必要があります。

  ML="https=true;*network-client-options***;..."**

  次に例を示します。

  ```
  ML="https=true;host=My-pc;port=82;trusted_certificates=rsaroot.crt"
  ```

  ネットワーク・クライアント・オプションのリストについては、「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』を参照してください。

  リダイレクタから Mobile Link サーバへの HTTPS 通信がサポートされているリダイレクタのリストについては、http://www.ianywhere.jp/sas/os.html を参照してください。

  Mobile Link サーバは、ML ディレクティブに指定されているものと同じプロトコルとポート番号を使用して起動する必要があります。異なる場合は、Mobile Link サーバを停止して、正しい指定で再起動してください。
- **ML_CLIENT_TIMEOUT**　ある 1 つの同期のすべての手順が同じ Mobile Link サーバに送信されていることを確認するときに使用します。ML_CLIENT_TIMEOUT が設定されている間、リダイレクタはクライアントとサーバ間の関係を維持します。また、この値は、同じリモート・データベースからの同期の重複を Mobile Link サーバに確実に検出させるためにも使用します。このパラメータの値は、ユーザの同期手順の最大値より大きい値にしてください。

  デフォルト値は 600 秒 (10 分) です。
- **SLEEP**　サーバが機能していることをリダイレクタがチェックする間隔を秒単位で設定するときに使用します。デフォルト値は 1800 (30 分) です。たとえば SLEEP=3600 のようになります。SLEEP では大文字と小文字を区別します。

4. 次のいずれかの項で Web サーバ固有の設定を完了します。
   - 「UNIX 上の Netscape/Sun Web サーバ用の NSAPI リダイレクタ (旧式)」 292 ページ
   - 「サーブレット・リダイレクタ (旧式)」 296 ページ

- 「Apache リダイレクタ (旧式)」 299 ページ
- 「M-Business Anywhere リダイレクタ (旧式)」 302 ページ

例

次に、*redirector.config* ファイルの例を示します。このファイルで指定されている内容は、以下のとおりです。

- リダイレクタは、サーバが機能していることを 1800 秒ごとにチェックする。
- Mobile Link サーバを稼働しているコンピュータで、要求を処理できるコンピュータは 3 台。複数のサーバを指定する場合、負荷分散は自動的に有効になります。

```
SLEEP=1800
ML=myServ-pc:80
ML=209.123.123.1:8080
ML=myCompany.com:8081
```


Windows 上の Netscape/Sun Web サーバ用の NSAPI リダイレクタ (旧式)

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

NSAPI リダイレクタは、Sun Java System Web サーバ用に提供されています。これは、以前 Sun One および Netscape iPlanet Enterprise Edition Web サーバと呼ばれていました。

バージョンのサポートの詳細については、http://www.ianywhere.jp/sas/os.html を参照してください。

このリダイレクタを UNIX で使用する方法については、「UNIX 上の Netscape/Sun Web サーバ用の NSAPI リダイレクタ (旧式)」 292 ページを参照してください。

その他のプラットフォームで Netscape/Sun Web サーバ用のリダイレクタを使用するには、サーブレット・リダイレクタを使用します。「サーブレット・リダイレクタ (旧式)」 296 ページを参照してください。

♦ NSAPI リダイレクタを設定するには、次の手順に従います。

1. 「リダイレクタのプロパティの設定 (サーバ・グループをサポートするリダイレクタの場合)」 284 ページの手順を完了します。
2. 必要に応じて、Web サーバが設定されているコンピュータに *iaredirect.dll* ファイルをコピーします。このファイルは、*install-dir¥MobiLink¥Redirector¥web-server* にあります。*web-server* は NSAPI Web サーバの名前です。
3. Web サーバがリダイレクタとは別のコンピュータ上にある場合は、次のファイルをそのコンピュータにコピーし、コピーしたファイルがパスに存在することを確認してください。必要なファイルは、暗号化の種類によって異なります (暗号化を使用している場合)。

   次に示すファイル・ロケーションは、*install-dir* を基準とした相対ディレクトリです。

設定	必要なファイル
すべて	● *bin32¥dblgen11.dll*[1] ● *bin32¥dbicu11.dll* ● *bin32¥dbicudt11.dll*
ECC 暗号化	● *bin32¥mlcecc11.dll*
RSA 暗号化	● *bin32¥mlcrsa11.dll*
FIPS 認定の RSA 暗号化	● *bin32¥mlcrsafips11.dll* ● *bin32¥sbgse2.dll*

[1] ドイツ語、日本語、中国語の版では、en をそれぞれ de、ja、zh と置き換えます。

言語を変更する方法については、「ロケール言語の知識」 『SQL Anywhere サーバ - データベース管理』を参照してください。

4. Apache Web サーバの設定ファイル *magnus.conf* と *obj.conf* を次のように更新します。

> **サンプル・ファイル**
> Mobile Link サーバ用として設定済みの *magnus.conf* と *obj.conf* のサンプル・コピーは、*install-dir¥MobiLink¥Redirector¥web-server* にあります。*web-server* は NSAPI Web サーバの名前です。

magnus.conf と *obj.conf* の次のセクションを更新します。

- *magnus.conf* で、*iaredirect.dll* とリダイレクタ設定ファイルが配置される場所を指定します。

  Init セクションの末尾に、次のテキストを追加します。*location* にはファイルの実際のロケーションを入力します (*iaredirect.dll* とリダイレクタ設定ファイルは異なるロケーションに配置することもできますが、どちらも、Web サーバと同じコンピュータ上または Web サーバからアクセスできるドライブ上に存在する必要があります)。

  ```
  Windows:
  Init fn="load-modules" shlib="location/iaredirect.dll"
  funcs="redirector,initialize_redirector"
  Init fn="initialize_redirector" configFile="location/redirector.config"
  ```

- *obj.conf* で、URL で使用されるリダイレクタの名前を指定します。

  default オブジェクトのセクションの先頭に、次のテキストを追加します。このセクションは、次のように表示されます。ただし、*mlredirect* は他の値に変更できます。*http://host:port/mlredirect/ml/** という形式のすべての要求は、リダイレクタとともに稼働している Mobile Link サーバのいずれかに送信されます。

  ```
  <Object name=default>
  NameTrans fn="assign-name" from="/mlredirect/ml/*" name="redirectToML"
  ```

- *obj.conf* で、リダイレクタが呼び出すオブジェクトを指定します。default オブジェクトのセクションの後に、次のセクションを追加します。

  ```
  <Object name="redirectToML">
  Service fn="redirector" serverType="ml"
  </Object>
  ```

例

次に、カスタマイズが必要な *magnus.conf* のセクションの例を示します。

```
Init fn="load-modules" shlib="D:/iaredirect.dll"
funcs="redirector,initialize_redirector"
Init fn="initialize_redirector" configFile="D:/redirector.config"
```

次に、リダイレクタにとって重要な *obj.conf* のセクションの例を示します。

```
<Object name=default>
NameTrans fn="assign-name" from="/mlredirect/ml/*" name="redirectToML"
...
<Object name="redirectToML">
Service fn="redirector" serverType="ml"
</Object>
```


♦ **設定をテストするには、次の手順に従います。**

1. 次の構文を使用してリダイレクタを呼び出します。

   http://host:port/mlredirect/ml/

2. ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

   「注意：」このテストは、Mobile Link サーバへの接続を作成しません。

UNIX 上の Netscape/Sun Web サーバ用の NSAPI リダイレクタ (旧式)

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

NSAPI リダイレクタは、Sun Java System Web サーバ用に提供されています。これは、以前 Sun One および Netscape iPlanet Enterprise Edition Web サーバと呼ばれていました。

バージョンのサポートの詳細については、http://www.ianywhere.jp/sas/os.html を参照してください。

このリダイレクタを Windows で使用する方法については、「Windows 上の Netscape/Sun Web サーバ用の NSAPI リダイレクタ (旧式)」 289 ページを参照してください。

その他のプラットフォームで Netscape/Sun Web サーバ用のリダイレクタを使用するには、サーブレット・リダイレクタを使用します。「サーブレット・リダイレクタ (旧式)」 296 ページを参照してください。

♦ NSAPI リダイレクタを設定するには、次の手順に従います。

1. 「リダイレクタのプロパティの設定 (サーバ・グループをサポートしないリダイレクタの場合)」 286 ページの手順を完了します。
2. 必要に応じて、Web サーバが設定されているコンピュータに *iaredirect.so* ファイルをコピーします。このファイルは、*install-dir¥MobiLink¥edirector¥web-server* にあります。*web-server* は NSAPI Web サーバの名前です。
3. NSAPI Web サーバの設定ファイル *magnus.conf* と *obj.conf* を次のように更新します。

   サンプル・ファイル
   magnus.conf と *obj.conf* のサンプル・コピーは、*install-dir¥MobiLink¥Redirector¥web-server* にあります。*web-server* は NSAPI Web サーバの名前です。これらのサンプル・ファイルを使用して、次のセクションがファイルのどこに該当しているかを確認できます。

   magnus.conf と *obj.conf* の次のセクションを更新します。

   - *magnus.conf* で、 *iaredirect.so* と *redirector.config* が配置される場所を指定します。

     Init セクションの末尾に、次のテキストを追加します。*location* にはファイルの実際のロケーションを入力します(*iaredirect.so* と *redirector.config* は Web サーバと同じコンピュータに置いてください。ロケーションはそれぞれ別でもかまいません)。

     ```
     Solaris:
     Init fn="load-modules" shlib="location/iaredirect.so"
     funcs="redirector,initialize_redirector"
     Init fn="initialize_redirector" configFile="location/redirector.config"
     ```

   - *obj.conf* で、URL で使用されるリダイレクタの名前を指定します。

"default object" セクションの先頭に、次のテキストを追加します。このセクションは、次のように表示されます。ただし、*mlredirect* は他の値に変更できます。*http://host:port/mlredirect/ml/** という形式のすべての要求は、リダイレクタとともに稼働している Mobile Link サーバのいずれかに送信されます。

```
<Object name=default>
NameTrans fn="assign-name" from="/mlredirect/ml/*" name="redirectToML"
```

- *obj.conf* で、リダイレクタが呼び出すオブジェクトを指定します。"default object" セクションの後に、次のセクションを追加します。

```
<Object name="redirectToML">
Service fn="redirector" serverType="ml"
</Object>
```

例

次に、カスタマイズが必要な *magnus.conf* のセクションの例を示します。

```
Init fn="load-modules" shlib="location/iaredirect.so"
funcs="redirector,initialize_redirector"
Init fn=" initialize_redirector " configFile="location/redirector.config"
```

次に、リダイレクタにとって重要な *obj.conf* のセクションの例を示します。

```
<Object name=default>
NameTrans fn="assign-name" from="/mlredirect/ml/*" name="redirectToML"
...
<Object name="redirectToML">
Service fn="redirector" serverType="ml"
</Object>
```

♦ **設定をテストするには、次の手順に従います。**

1. 次の構文を使用してリダイレクタを呼び出します。

   ```
   http://host:port/mlredirect/ml/
   ```

2. ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

   「注意：」このテストは、Mobile Link サーバへの接続を作成しません。

Microsoft Web サーバ用の ISAPI リダイレクタ (旧式)

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

Microsoft Web サーバを使用している場合は、ISAPI 版のリダイレクタを使用できます。

バージョンのサポートの詳細については、http://www.ianywhere.jp/sas/os.html を参照してください。

♦ **Microsoft Web サーバに ISAPI リダイレクタを設定するには、次の手順に従います。**

1. 「リダイレクタのプロパティの設定 (サーバ・グループをサポートするリダイレクタの場合)」 284 ページの手順を完了します。
2. *iaredirect.dll* ファイルを、Web サーバが設定されているコンピュータの *Inetpub¥scripts* にコピーします。

   ファイル *iaredirect.dll* は、*install-dir¥MobiLink¥Redirector¥IIS5* にあります。

   Inetpub¥scripts ディレクトリは、Microsoft Web サーバのインストール・ディレクトリにあります。
3. Web サーバがリダイレクタとは別のコンピュータ上にある場合は、次のファイルをそのコンピュータにコピーし、コピーしたファイルがパスに存在することを確認してください。必要なファイルは、暗号化の種類によって異なります (暗号化を使用している場合)。

   次に示すファイル・ロケーションは、*install-dir* を基準とした相対ディレクトリです。

設定	必要なファイル
すべて	● *bin32¥dblgen11.dll*[1] ● *bin32¥dbicu11.dll* ● *bin32¥dbicudt11.dll*
ECC 暗号化	● *bin32¥mlcecc11.dll*
RSA 暗号化	● *bin32¥mlcrsa11.dll*
FIPS 認定の RSA 暗号化	● *bin32¥mlcrsafips11.dll* ● *bin32¥sbgse2.dll*

[1] ドイツ語、日本語、中国語の版では、en をそれぞれ de、ja、zh と置き換えます。

言語を変更する方法については、「ロケール言語の知識」 『SQL Anywhere サーバ - データベース管理』を参照してください。

注意

♦ **設定をテストするには、次の手順に従います。**

1. 次の構文を使用して ISAPI リダイレクタを呼出します。

   ```
   protocol://host[:port]/exec_dir/iaredirect.dll/ml/
   ```

   文中の各項目を次に説明します。

   - **protocol** http または https です。
   - **host** Web サーバのホスト名です。
   - **port** デフォルトのポートでない場合は、Web サーバが受信しているポートです。
   - **exec_dir** リダイレクタの DLL である *iaredirect.dll* がインストールされているディレクトリです。デフォルトのディレクトリは scripts です。

     次に例を示します。

     ```
     http://server:8080/scripts/iaredirect.dll/ml/
     ```

2. ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

   「注意：」このテストは、Mobile Link サーバへの接続を作成しません。

3. 正しく設定できなかった場合は、以下を確認してください。

   - *Inetpub¥scripts* ディレクトリが Web サーバのインストール時に実行アクセス許可付きで作成されているか確認します。
   - リダイレクタ設定ファイルと *iaredirect.dll* を異なるディレクトリに保管できるのは、インターネット インフォメーション サービスを使用してディレクトリに実行アクセス許可を付与できる場合だけです。
   - *Inetpub¥scripts* ディレクトリを指す仮想ディレクトリが必要です。これがない場合は、インターネット インフォメーション サービスを起動して、仮想ディレクトリを手動で作成する必要があります。この仮想ディレクトリは、*Inetpub¥scripts* を指すようにし、[実行アクセス許可] を [スクリプトおよび実行可能ファイル] に設定してください。その手順については、IIS のオンライン・ヘルプを参照してください。

サーブレット・リダイレクタ (旧式)

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

サーブレット・リダイレクタは、Java サーブレット仕様バージョン 2.3 以降をサポートする Web サーバで提供されます。次の手順は、Tomcat バージョン 5.5.9 と Apache 2.0.55 用のサーブレット・リダイレクタの設定例です。

バージョンのサポートの詳細については、http://www.ianywhere.jp/sas/os.html を参照してください。

また、Apache Web サーバ用のネイティブなリダイレクタもあります。詳細については、「Apache リダイレクタ (旧式)」 299 ページを参照してください。

概要

この項では、Tomcat サーブレット・コンテナとともに Apache Web サーバで動作するように、サーブレット版のリダイレクタをインストールする方法を説明します。

インストールには、以下の手順が必要です。

♦ Apache Tomcat 用のサーブレット・リダイレクタを設定する手順

1. 「リダイレクタのプロパティの設定 (サーバ・グループをサポートしないリダイレクタの場合)」 286 ページの手順を完了します。
2. サーブレット版のリダイレクタを Tomcat にインストールします。
3. プロキシとして動作するように Apache Web サーバを設定します。

Tomcat へのサーブレット・リダイレクタのインストール

次の手順で、*%CATALINA_HOME%* は、Tomcat インストールのルート・ディレクトリです。

♦ Tomcat にサーブレット・リダイレクタをインストールするには、次の手順に従います。

1. Tomcat をスタンドアロン・サーバとしてインストールします。
2. 必要に応じて、Tomcat の HTTP ポートを設定します。

   Tomcat はデフォルトでポート 8080 にバインドされます。競合が発生する場合、別の Web サーバがこのポートを使用している可能性があります。

   - ファイル *%CATALINA_HOME%/conf/server.xml* を開きます。
   - 8080 を検索します (<Connector> タグ内にあります)。
   - これを使用中でないポートに変更します。
3. サーブレット・リダイレクタを Web アプリケーションとしてインストールします。

- *iaredirect.war* ファイルを *%CATALINA_HOME%/webapps* にコピーします。
- Tomcat を停止し、再起動します。

  war ファイルが展開され、リダイレクタ Web アプリケーション用のディレクトリ *iaredirect* が作成されます。
- ファイル *%CATALINA_HOME%/webapps/iaredirect/WEB-INF/web.xml* を編集します。**redirector.config** (<init-param> タグ内) を検索し、*redirector.config* ファイルのパスを修正します。

  drive:/path/redirector.config を読み取るようにエントリ **redirector.config** を変更します。Windows オペレーティング・システムの場合も、*d:/redirector.config* のように、パスの区切り文字として通常のスラッシュを使用します。
- Tomcat を停止し、再起動して、変更を有効にします。

  変更を有効にすると、配備された場所に war ファイルが存在する必要はなくなります。
- これで、リダイレクタは次の URL から呼び出すことができます。

  http://*tc-host*:*tc-port*/iaredirect/ml/

  tc-host はコンピュータで、*tc-port* は Tomcat が受信しているポートです。

Apache Web サーバのプロキシとしての設定

次の手順で、*%APACHE_HOME%* は、Apache インストールのルート・ディレクトリです。

♦ Apache Web サーバをプロキシとして設定するには、次の手順に従います。

1. Apache Web サーバをインストールします。
2. 必要に応じて、Apache Web サーバのポートを変更します。
   - ファイル *%APACHE_HOME%/conf/httpd.conf* を編集し、**Port** 設定を変更します。
3. プロキシとして動作するように Apache を設定します。

   %APACHE_HOME%/conf/httpd.conf に、次のディレクティブを追加します。

   ```
   LoadModule proxy_module module-path/mod_proxy.so
   LoadModule proxy_connect_module module-path/mod_proxy_connect.so
   LoadModule proxy_http_module module-path/mod_proxy_http.so
   ```

   module-path は、module が存在するロケーションです。たとえば、パスは *modules/mod_proxy.so* (デフォルト) とします。
4. リダイレクタの URL を Tomcat に転送するように Apache を設定します。

   %APACHE_HOME%/conf/httpd.conf ファイルで、次のディレクティブを追加します。追加すると、Apache は http://localhost/iaredirect/* という形式の URL を、ポート 8080 で受信している Tomcat 5 コネクタに転送します。

   ```
   ProxyPass /iaredirect http://localhost:8080/iaredirect
   ```

ポート番号は、Tomcat で使用されているポート番号に一致している必要があります。Tomcat と Apache が同じコンピュータで実行されていない場合は、**localhost** の代わりに、Tomcat が実行されているコンピュータ名を指定します。

設定の確認

♦ **設定を確認するには、次の手順に従います。**

1. 次の構文を使用してリダイレクタを呼び出します。

   http://host:port/iaredirect/ml/

2. ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。

   「注意：」このテストは、Mobile Link サーバへの接続を作成しません。

Apache リダイレクタ (旧式)

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

以下の設定手順は Apache Web サーバ用に記述されています。

バージョンのサポートの詳細については、http://www.ianywhere.jp/sas/os.html を参照してください。

Tomcat を使用している場合は、サーブレット・リダイレクタも使用できます。詳細については、「サーブレット・リダイレクタ (旧式)」 296 ページを参照してください。

♦ Apache リダイレクタを設定するには、次の手順に従います。

1. 「リダイレクタのプロパティの設定 (サーバ・グループをサポートしないリダイレクタの場合)」 286 ページの手順を完了します。
2. 次に示すように、*mod_iaredirect.dll* ファイルまたは *mod_iaredirect.so* ファイルを、Web サーバの適切なディレクトリにコピーします。
   - Windows の Apache の場合は、ファイル *mod_iaredirect.dll* が *install-dir¥MobiLink¥Redirector¥Apache¥V20¥* にあります。このファイルを、Web サーバが設定されているコンピュータの *%apache-home%¥modules* ディレクトリにコピーします。
   - Solaris または Linux の Apache の場合は、ファイル *mod_iaredirect.so* が *install-dir/mobiLink/redirector/apache/v20/* にあります。このファイルを、Web サーバが設定されているコンピュータの *$APACHE_HOME/modules* ディレクトリにコピーします。
3. Web サーバがリダイレクタとは別のコンピュータ上にある場合は、次のファイルをそのコンピュータにコピーし、コピーしたファイルがパス (Windows) または共有パス (UNIX) に存在することを確認してください。必要なファイルは、暗号化の種類によって異なります (暗号化を使用している場合)。

   次に示すファイル・ロケーションは、*install-dir* を基準とした相対ディレクトリです。

設定	必要なファイル
ECC 暗号化	● Windows : *bin32¥mlcecc11.dll* ● UNIX : *lib32/libmlcecc11_r.so*
RSA 暗号化	● Windows : *bin32¥mlcrsa11.dll* ● UNIX : *lib32/libmlcrsa11_r.so*
FIPS 認定の RSA 暗号化	● Windows : *bin32¥mlcrsafips11.dll* と *bin32¥sbgse2.dll* ● UNIX : *lib32/libmlcrsafips11_r.so* と *libsbgse2_r.so*

4. Apache Web サーバの設定ファイル *httpd.conf* を次のように更新します。

- Windows の場合は、LoadModule セクションで、次の行を追加します。

  ```
  LoadModule iaredirect_module modules/mod_iaredirect.dll
  ```

  Solaris または Linux の場合は、次の行を追加します。

  ```
  LoadModule iaredirect_module modules/mod_iaredirect.so
  ```

- 次のセクションをファイルに追加します。

  ```
  <Location /iaredirect/ml>
    SetHandler iaredirect-handler
    iaredirectorConfigFile location/redirector.config
  </Location>
  ```

  ここで、*/iaredirect/ml* はリダイレクタを起動するために使用する相対 URL パスで、*location* は *redirector.config* が配置されたディレクトリです。

- Solaris または Linux で Apache を使用している場合は、作成した <Location> セクションに、次のオプションのディレクティブを追加することもできます。

  - **MaxSyncUsers *number*** リダイレクタを経由して同期する Mobile Link ユーザの最大数。この値を使用して、リダイレクタに必要なリソースが割り当てられます。この値は 60 未満であってはなりません。デフォルトは 1000 です。デフォルトのユーザ数が実際の数より少ない場合のみ、この設定を変更します。
  - **ShmemDiagnosis on|off** on に設定すると、メモリ・リソースをデバッグできます。デフォルトは off です。

5. デバッグに役立つように、リダイレクタが出力するログ情報量を増やすことができます。このためには、*httpd.conf* 内の LogLevel ディレクティブを変更して、これを **LogLevel info** に設定します。ログ・レベルは、debug、info、notice、warn、error、crit、alert、emerg (情報量が多いものから少ないものの順) です。

例

次の例は、Apache Web サーバが Mobile Link サーバに要求をルート指定するように設定している *httpd.conf* のセクションを示しています。この例は、Windows でのみ使用できます。UNIX と Linux の場合は、*mod_iaredirect.dll* を *mod_iaredirect.so* に変更します。

```
LoadModule iaredirect_module modules/mod_iaredirect.dll
...

<Location /iaredirect/ml>
   SetHandler iaredirect-handler
   iaredirectorConfigFile c:/redirector.config
</Location>
```

♦ **設定をテストするには、次の手順に従います。**

1. 次の構文を使用してリダイレクタを呼び出します。

   ```
   http://host:port/iaredirect/ml/
   ```

   ここで、*iaredirect/ml* は、*httpd.conf* の <Location> タグで指定した相対 URL パスです。

2. ログ・ファイルをチェックして、リダイレクタが要求をログに記録しているかどうかを確認します。ログ・ファイルのデフォルトのロケーションは、*$APACHE_HOME/logs/error.log* です。

   「注意：」このテストは、Mobile Link サーバへの接続を作成しません。

M-Business Anywhere リダイレクタ (旧式)

注意
リダイレクタは推奨されなくなりました。代わりに、リレー・サーバを使用してください。「リレー・サーバ」 247 ページを参照してください。

以下の設定手順は、Windows、Solaris、Linux 上の M-Business Anywhere 用に記述されています。

バージョンのサポートの詳細については、http://www.ianywhere.jp/sas/os.html を参照してください。

♦ M-Business Anywhere リダイレクタを設定するには、次の手順に従います。

1. 「リダイレクタのプロパティの設定 (サーバ・グループをサポートしないリダイレクタの場合)」 286 ページの手順を完了します。
2. *mod_iaredirect.dll* ファイルまたは *mod_iaredirect.so* ファイルを、Web サーバが設定されているコンピュータの M-Business Anywhere *¥bin* ディレクトリにコピーします。このファイルは *install-dir¥MobiLink¥Redirector¥MBusinessAnywhere* にあります。
3. Web サーバがリダイレクタとは別のコンピュータ上にある場合は、次のファイルをそのコンピュータにコピーし、コピーしたファイルがパス (Windows) または共有パス (UNIX) に存在することを確認してください。必要なファイルは、暗号化の種類によって異なります (暗号化を使用している場合)。

   次に示すファイル・ロケーションは、*install-dir* を基準とした相対ディレクトリです。

設定	必要なファイル
ECC 暗号化	● Windows : *bin32¥mlcecc11.dll* ● UNIX : *lib32/libmlcecc11_r.so*
RSA 暗号化	● Windows : *bin32¥mlcrsa11.dll* ● UNIX : *lib32/libmlcrsa11_r.so*
FIPS 認定の RSA 暗号化	● Windows : *bin32¥mlcrsafips11.dll* と *bin32¥sbgse2.dll* ● UNIX : *lib32/libmlcrsafips11_r.so* と *libsbgse2_r.so*

4. Windows の場合は、M-Business Anywhere の *sync.conf.default* Web サーバ設定ファイルを次のように更新します。
   - LoadModule セクションで、次の行を追加します。

     ```
     LoadModule iaredirect_module @@ServerRoot@@/bin/mod_iaredirect.dll
     ```

   - SyncLoadFile セクションで、次の行を追加します。

     ```
     SyncLoadFile @@ServerRoot@@/bin/mod_iaredirect.dll
     ```

   - 次のセクションをファイルに追加します。


```
<Location /iaredirect/ml>
  SetHandler iaredirect-handler
  iaredirectorConfigFile @@ServerRoot@@/conf/redirector.config
</Location>
```

- *setup_defaults.bat* コマンド・ファイルを実行して、これらの変更を *sync.conf* ファイルに組み込みます。

5. Solaris と Linux の場合は、M-Business Anywhere Web サーバの *sync.conf* Web サーバ設定ファイルを次のように更新します。
   - LoadModule セクションで、次の行を追加します。

     ```
     LoadModule iaredirect_module path/bin/mod_iaredirect.so
     ```

     path は、M-Business Anywhere *bin* ディレクトリのロケーションです。
   - 次のセクションをファイルに追加します。

     ```
     <Location /iaredirect/ml>
       SetHandler iaredirect-handler
       iaredirectorConfigFile location/redirector.config
     </Location>
     ```

     ここで、*/iaredirect/ml* はリダイレクタを起動するために使用する相対 URL パスで、*location* は *redirector.config* が配置されたディレクトリです。
   - 作成した <Location> セクションに、次のオプションのディレクティブを追加することもできます。
     - **MaxSyncUsers *n*** リダイレクタを経由して同期する Mobile Link ユーザの最大数。この値を使用して、リダイレクタに必要なリソースが割り当てられます。この値は 60 未満であってはなりません。デフォルトは 1000 です。デフォルトのユーザ数が実際の数より少ない場合のみ、この設定を変更します。
     - **ShmemDiagnosis on|off** on に設定すると、メモリ・リソースをデバッグできます。デフォルトは off です。
6. デバッグに役立つように、リダイレクタが出力するログ情報量を増やすことができます。このためには、*sync.conf* 内の LogLevel ディレクティブを変更して、これを **LogLevel info** に設定します。ログ・レベルは、debug、info、notice、warn、error、crit、alert、emerg (情報量が多いものから少ないものの順) です。
7. M-Business Sync Server を再起動して、変更を有効にします。

例

次の例は、M-Business Anywhere Web サーバが Mobile Link サーバに要求をルート指定するように設定している *sync.conf* のセクションを示しています。

この例は、Windows で使用できます。

```
LoadModule iaredirect_module "c:¥program files¥M-Business Anywhere/bin/mod_iaredirect.dll"
...
SyncLoadFile "c:¥program files¥M-Business Anywhere/bin/mod_iaredirect.dll"
...
<Location ¥iaredirect¥ml>
   SetHandler iaredirect-handler
   iaredirectorConfigFile "c:¥AvantGoServer¥conf/redirector.config"
</Location>
```

次の例は、UNIX と Linux で使用できます。

```
LoadModule iaredirect_module modules/mod_iaredirect.so
...
<Location /iaredirect/ml>
   SetHandler iaredirect-handler
   iaredirectorConfigFile "/redirector.config"
</Location>
```

♦ **設定をテストするには、次の手順に従います。**

1. 次の構文を使用してリダイレクタを呼び出します。

   ```
   http://host:port/iaredirect/ml/
   ```

   ここで、*iaredirect* は、*sync.conf* の <Location> タグで指定したパスです。

2. ログ・ファイル *sync_access.log* と *sync_error.log* をチェックして、リダイレクタが要求をログに記録していることを確認します。

   「注意：」このテストは、Mobile Link サーバへの接続を作成しません。

Mobile Link ファイルベースのダウンロード

目次

ファイルベースのダウンロードの概要

ファイルベースのダウンロードは、SQL Anywhere リモート・データベースにデータをダウンロードする、もう 1 つの方法です。ダウンロードの内容はファイルとして配布でき、同期の変更をオフラインで配布できます。このため、ファイルを一度作成すれば、多数のリモート・データベースにこのファイルを配布できます。

ファイルベースのダウンロードでは、ダウンロードした同期の変更内容をファイルに保存し、ファイルを転送可能なあらゆる方法を使用して SQL Anywhere リモート・データベースに転送できます。次に例を示します。

- 衛星マルチキャストでデータをブロードキャストする。
- Sybase Afaria を使用して更新を適用する。
- ファイルを電子メールまたは FTP でユーザに送信する。

ファイルを送信するユーザを選択します。ファイルベースのダウンロードでは、競合の検出と解決を含め、同期の整合性は完全に保護されます。このファイルにサード・パーティの暗号化を適用することにより、ファイルの安全性を保証できます。

使用する場合

ファイルベースのダウンロードは、統合データベースで大量のデータが変更されるが、リモート・データベースではデータの更新の頻度が低いか、更新がまったく行われない場合に便利です。たとえば、価格リスト、製品リスト、コードのテーブルなどです。

ファイルベースのダウンロードは、ダウンロードされたデータがリモート・データベースで頻繁に更新される場合、またはアップロード専用の同期を頻繁に実行している場合には適していません。このような状況では、ダウンロード・ファイルの適用時に実行される整合性のチェックが原因で、リモート・サイトはダウンロード・ファイルを適用できないことがあります。

現時点では、ファイルベースのダウンロードは SQL Anywhere リモート・データベースでのみ使用可能です。

ダウンロード専用のパブリケーション

ほとんどの場合、ファイルベースのダウンロードにはダウンロード専用のパブリケーションを使用する必要があります。通常のパブリケーションは、ファイルベースのダウンロードの実行と同じパブリケーションを使用してアップロードを実行する必要がある場合にかぎり使用してください。

「ダウンロード専用のパブリケーション」『Mobile Link - クライアント管理』を参照してください。

通常のパブリケーションを使用する場合は、リモート・データベースを更新する手段として、ファイルベースのダウンロードのみを使用することはできません。この場合でも、完全な同期またはアップロード専用の同期を定期的に実行する必要があります。完全な同期またはアップロード専用の同期は、ログ・オフセットを進めたり、ログ・ファイルを保守したりするために必要です。こうしないと、ログ・ファイルのサイズが大きくなり、同期処理に時間がかかるようになります。また、完全な同期ではエラーからのリカバリが必要になることもあります。

ファイルベースのダウンロードの設定

以下の手順では、ファイルベースのダウンロードを設定するのに必要なタスクの概要について説明します。この手順では、Mobile Link 同期がすでに設定されているものとします。

ほとんどの場合、ファイルベースのダウンロードにはダウンロード専用のパブリケーションを使用する必要があります。通常のパブリケーションは、ファイルベースのダウンロードの実行と同じパブリケーションを使用してアップロードを実行する必要がある場合にかぎり使用してください。

♦ ファイルベースのダウンロードの設定の概要

1. ファイル定義データベースを作成します。

   「ファイル定義データベースの作成」 307 ページを参照してください。

2. 統合データベースで、新しいスクリプト・バージョンを使用してスクリプトを作成します。

   「統合データベースでの変更」 308 ページを参照してください。

3. ダウンロード・ファイルを作成します。

   「ダウンロード・ファイルの作成」 308 ページを参照してください。

4. ダウンロード・ファイルを適用します。

   「新しいリモートの同期」 309 ページを参照してください。

クイック・スタートのためのその他の資料

- 「ファイルベースのダウンロード例」 315 ページ

ファイル定義データベースの作成

ファイルベースのダウンロードを設定するには、「ファイル定義データベース」を作成します。これは、リモート・データベースと同じ同期テーブルと同期パブリケーションを持つ SQL Anywhere データベースです。配置する場所は、どこでもかまいません。このデータベースには、データまたはステータスの情報はありません。バックアップまたは保守を行う必要がなく、必要に応じて削除したり、再作成できます。

ファイル定義データベースには、次のものが含まれている必要があります。

- リモート・データベースと同じパブリケーション、そのパブリケーションで使用されるテーブルとカラム、これらのテーブルとカラムの外部キー関係と制約、これらの外部キー関係に必要なテーブル。
- ダウンロード・ファイルを適用するリモート・データベースのグループを識別する Mobile Link ユーザ名。リモート・データベースのグループを識別するには、このグループの Mobile Link ユーザ名を同期スクリプトで使用します。

統合データベースでの変更

統合データベースでは、ファイルベースのダウンロード用のスクリプト・バージョンを新規に作成し、既存の同期システムに必要なスクリプトを実装します。アップロード・スクリプトは必要ありません。このスクリプト・バージョンは、ファイルベースのダウンロードのみに使用されます。このスクリプト・バージョンの場合、パラメータとして Mobile Link ユーザ名を利用するすべてのスクリプトは、リモート・データベースのグループを示す Mobile Link ユーザ名をパラメータとして利用します。これは、ファイル定義データベースで定義されているユーザ名です。

定義した各スクリプト・バージョンには、begin_publication スクリプトを実装します。

タイムスタンプベースのダウンロードの場合は、各スクリプト・バージョンに modify_last_download_timestamp スクリプトを実装します。このスクリプトの実装方法は、各ダウンロード・ファイルで送信するデータ量によって異なります。たとえば、グループのユーザによる前回の正常なダウンロードの時刻の中で最も早いものを使用する方法があります。このスクリプトに渡される ml_username パラメータは、実際にはグループ名です。

参照

- 「スクリプト・バージョン」 341 ページ
- 「begin_publication 接続イベント」 399 ページ
- 「modify_last_download_timestamp 接続イベント」 486 ページ

ダウンロード・ファイルの作成

ダウンロード・ファイルには、同期されるデータが格納されています。ダウンロード・ファイルを作成するには、上記の説明のようにファイル定義データベースと統合データベースを設定します。-bc オプションを使用し、拡張子が *.df* のファイル名を指定して dbmlsync を実行します。次に例を示します。

```
dbmlsync -c "uid=DBA;pwd=sql;eng=fbdl_eng;dbf=fdef.db" -v+
  -e "sv=filebased" -bc file1.df
```

ダウンロード・ファイルの作成時にオプションを指定することもできます。

- **-be オプション** -be オプションを使用すると、sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを使用してリモート・データベースでアクセス可能なダウンロード・ファイルに文字列を追加できます。

  「-be オプション」 『Mobile Link - クライアント管理』と「sp_hook_dbmlsync_validate_download_file」 『Mobile Link - クライアント管理』を参照してください。

- **-bg オプション** -bg オプションを使用すると、一度も同期されていないリモート・データベースによって使用可能なダウンロード・ファイルを作成できます。

新しいリモートの同期

Mobile Link を使用して同期されたことのないリモート・データベースにダウンロード・ファイルを適用する場合は、リモート・データベースで通常の同期を実行してからダウンロード・ファイルを適用するか、ダウンロード・ファイルの作成時に dbmlsync -bg オプションを使用する必要があります。

タイムスタンプベースの同期の場合は、この 2 つの方法のいずれかを行うと、データの初期のスナップショットがダウンロードされます。タイムスタンプベースとスナップショットベースの両方の同期では、この手順によって、統合データベースの begin_publication スクリプトが生成する値に世代番号が設定されます。

通常の同期の実行

ダウンロード・ファイルを使用しない同期を実行することによって、ダウンロード・ファイルを受信するためのリモート・データベースを準備します。

-bg オプションの使用

別の方法として、まだ同期されていないリモート・データベースで使用するために、-bg オプションを使用してダウンロード・ファイルを作成できます。この初期ダウンロード・ファイルを適用して、ファイルベースの同期に使用されるリモート・データベースを準備します。

- **スナップショットのダウンロード**　スナップショットのダウンロードを実行している場合は、初期ダウンロード・ファイルに世代番号を設定する必要があります。このファイルにデータの初期スナップショットを含めることは可能ですが、各スナップショットのダウンロードにはすべてのデータが含まれ、前のダウンロードに依存しないため必須ではありません。

  スナップショットのダウンロードは、-bg オプションを使用すると簡単です。ダウンロード・ファイルを作成するときに、dbmlsync コマンドで -bg を指定するだけです。同じスクリプト・バージョンを使用して、以降のダウンロード・ファイルに使用する初期ダウンロード・ファイルを作成できます。

- **タイムスタンプベースのダウンロード**　タイムスタンプベースのダウンロードを実行している場合は、初期ダウンロードでリモート・データベースの世代番号を設定し、データのスナップショットを含める必要があります。タイムスタンプベースのダウンロードでは、各ダウンロードは前のダウンロードに基づいています。ダウンロード・ファイルには、それぞれ最終ダウンロード・タイムスタンプが格納されています。ファイルの最終ダウンロード・タイムスタンプの後に統合データベースで変更されたローは、すべてこのファイルに格納されています。ファイルを適用するには、ファイルの最終ダウンロード・タイムスタンプの前に発生したすべての変更をリモート・データベースが受信している必要があります。この確認は、ファイルの最終ダウンロード・タイムスタンプが、リモート・データベースの最終ダウンロード・タイムスタンプ (リモート・データベースが統合データベースからすべての変更を受信するまでの時刻) 以降であることをチェックして行われます。

  リモート・データベースでは、最初の通常のダウンロード・ファイルを適用する前に、ファイルの最終ダウンロード・タイムスタンプより前に変更され、しかも 1900 年 1 月 1 日より後に変更されたすべてのデータを受信している必要があります。このデータを選択する最も簡単な方法は、通常のファイルベースの同期スクリプト・バージョンと同じ download_cursor を使用していても、modify_last_download_timestamp スクリプトは含まない別のスクリプト・バージョンを作成することです。no modify_last_download_timestamp スクリプトが定義されて

いない場合、ファイルベースのダウンロードの最終ダウンロード・タイムスタンプは、デフォルトで 1900 年 1 月 1 日に設定されます。

-bg オプションを使用して構築されたダウンロード・ファイルを同期済みのリモート・データベースに適用すると、この -bg オプションにより、ダウンロード・ファイルが作成されたときの統合データベースの値を使用してリモート・データベースで世代番号が更新されます。このため、世代番号は無効になります。世代番号は、消失または破損した統合データベースをリカバリする場合にアップロードが実行されるまで、ファイルベースのダウンロードをそれ以上適用しないようにするためのものです。

「Mobile Link の世代番号」 313 ページを参照してください。

検証チェック

dbmlsync は、同期が有効であることを確認するためにいくつかの処理を行ってから、ダウンロード・ファイルをリモート・データベースに適用します。

- dbmlsync は、ダウンロード・ファイルの作成に使用されたファイル定義データベースに次のものが含まれていることを確認するため、このダウンロード・ファイルをチェックします。
  - リモート・データベースと同じパブリケーション
  - そのパブリケーションで使用される同じテーブルとカラム
  - それらのテーブルとカラムと同じ外部キー関係
- dbmlsync は、リモート・データベースからアップロードされていないデータがパブリケーションに存在するかどうかをチェックします。データが存在する場合は、ダウンロード・ファイルは適用されません。これは、ダウンロード・ファイルを適用すると、保留中のアップロードが消失することがあるためです。
- dbmlsync は、最終ダウンロード・タイムスタンプ、次の最終ダウンロード・タイムスタンプ、ダウンロード・ファイルの作成時刻をチェックして、次のことを確認します。
  - リモート・データベースの新しいデータが、ダウンロード・ファイルに含まれる古いデータで上書きされないこと。
  - ダウンロード・ファイルを適用すると、統合データベースで発生した変更の一部をリモート・データベースが取得しない場合には、ダウンロード・ファイルを適用しないこと。この状況は、リモート・データベースが前のファイルベースのダウンロードを適用しなかった場合に発生することがあります。

    「自動検証」 311 ページを参照してください。
- オプションで、dbmlsync はダウンロード・ファイルの世代番号と一致することを確認するために、リモート・データベースで世代番号をチェックします。

  「Mobile Link の世代番号」 313 ページを参照してください。
- オプションで、sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを使用して、カスタムの検証論理を作成できます。

  詳細については、「カスタム検証」 313 ページを参照してください。

自動検証

dbmlsync は、最終ダウンロード・タイムスタンプ、次の最終ダウンロード・タイムスタンプ、ダウンロード・ファイルの作成時刻、トランザクション・ログに特別なチェックを実行してから、ダウンロード・ファイルを適用します。

最終ダウンロード・タイムスタンプと次の最終ダウンロード・タイムスタンプ

各ダウンロード・ファイルには、ファイルの最終ダウンロード・タイムスタンプから次の最終ダウンロード・タイムスタンプまでの間に統合データベースで発生したダウンロード対象のすべての変更が格納されています。この時刻は、統合データベースの時刻です。デフォルトでは、ファ

イルの最終ダウンロード時刻は 1900 年 1 月 1 日 12:00 AM で、ファイルの次の最終ダウンロード・タイムスタンプはダウンロード・ファイルが作成された時刻です。これらのデフォルト値を上書きするには、modify_last_download_timestamp スクリプトと modify_next_last_download_timestamp スクリプトを統合データベースに実装します。

リモート・サイトは、ファイルの最終ダウンロード・タイムスタンプが、リモートの最終ダウンロード・タイムスタンプ以前である場合にのみ、ダウンロード・ファイルを適用できます。これにより、リモートは統合データベースで発生した操作を失うことはありません。通常、このチェックに基づいたファイルベースのダウンロードが失敗した場合、リモート・サイトは 1 つまたは複数のダウンロード・ファイルを失っていることになります。この状況を修正するには、取得しなかったダウンロード・ファイルを適用するか、完全な同期またはダウンロード専用の同期を実行します。

さらに、リモート・サイトは、次のファイルの最終ダウンロード・タイムスタンプが、リモートの最終ダウンロード・タイムスタンプよりも後である場合にのみ、ダウンロード・ファイルを適用できます。リモートの最終ダウンロード・タイムスタンプは、ダウンロード対象のすべての変更をリモートが受信するまでの時刻 (統合データベースでの時刻) です。リモート・データベースの最終ダウンロード時刻は、通常またはファイルベースのダウンロードをリモートが正常に適用するたびに更新されます。このチェックを行うことにより、より新しいデータがすでにダウンロードされている場合はダウンロード・ファイルが適用されることはありません。一般的には、これが発生するのは、ダウンロード・ファイルが正常に適用されなかった場合です。たとえば、ダウンロード・ファイル *F1.df* が作成され、別のファイル *F2.df* が後で作成されたとします。このチェック機能により、*F2.df* の後に *F1.df* が適用されることはありません。これは、*F2.df* の新しいデータが、*F1.df* の古いデータで上書きされてしまうのを防ぐためです。

次の最終ダウンロード・タイムスタンプに基づいたファイルベースのダウンロードが失敗した場合、このファイルを削除する以外に必要な作業はありません。新しいファイルを受信すると、同期は成功します。

作成時刻

ダウンロード・ファイルの作成時刻は、ファイルの作成が開始された時点の統合データベースでの時刻を示しています。ダウンロード・ファイルを適用できるのは、ファイルの作成時刻が、リモート・データベースの最終アップロード時刻よりも後の場合だけです。リモートの最終アップロード時刻は、リモートの正常な最終アップロードがコミットされた時点の統合データベースでの時刻です。このチェックにより、ダウンロードの作成後にアップロードされた (ダウンロードよりも新しい) データは、ダウンロード・ファイルの古いデータで上書きされることはありません。

このチェックに基づいてダウンロード・ファイルが拒否されても、必要な作業はありません。リモート・サイトは、次のダウンロード・ファイルの適用が可能になっている必要があります。

dbmlsync がアップロードを Mobile Link サーバに送信した後に確認を取得しなかったためにアップロードが失敗した場合は、リモート・データベースの最終アップロード時刻が正しくないことがあります。この場合、作成時刻のチェックを実行できません。また、リモート・データベースは通常の同期を完了するまでダウンロード・ファイルを適用できません。

トランザクション・ログ

dbmlsync は、リモート・データベースのトランザクション・ログをスキャンし、アップロードする必要があるすべての変更のリストを構築してから、ダウンロード・ファイルを適用します。

dbmlsync がダウンロード・ファイルを適用するのは、アップロードが必要な変更のあるローに影響する操作がダウンロード・ファイルに含まれていない場合だけです。

Mobile Link の世代番号

世代番号とは、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です。これは、統合データベースで問題が発生したためにデータが失われ、そのデータをリモート・データベースからリカバリする必要があるときに、特に役立ちます。

リモート・データベースでは、各サブスクリプションに対して別々の世代番号が自動的に管理されています。統合データベースでは、各サブスクリプションの世代番号は begin_publication スクリプトによって決定されます。リモート・データベースが正常にアップロードを行うたびに、リモート・データベースの世代番号は統合データベースの begin_publication スクリプトによって設定された値で更新されます。

ダウンロード・ファイルが作成されるたびに、begin_publication スクリプトによって設定された世代番号がダウンロード・ファイルに格納されます。リモート・サイトは、ダウンロード・ファイルの世代番号がリモート・データベースに格納されている世代番号と同じ場合にのみ、ダウンロード・ファイルを適用します。

注意
begin_publication スクリプトによってファイルベースのダウンロード用に生成された世代番号が変更された場合、リモート・データベースは正常なアップロードを実行してから、新しいダウンロード・ファイルを適用する必要があります。

sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを使用すると、デフォルトで行われる世代番号のチェックを無効にすることができます。

Mobile Link の世代番号の管理については、次の項を参照してください。

- 「begin_publication 接続イベント」 399 ページ
- 「end_publication 接続イベント」 445 ページ
- 「sp_hook_dbmlsync_validate_download_file」 『Mobile Link - クライアント管理』

カスタム検証

カスタムの検証論理を作成すると、ダウンロード・ファイルをリモート・データベースに適用する必要があるかどうかを判断できます。これには、sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを使用します。このストアド・プロシージャを使用すると、ダウンロード・ファイルを拒否し、デフォルトで行われる世代番号のチェックを無効にすることができます。

dbmlsync -be オプションを使用すると、文字列をファイルに埋め込むことができます。-be オプションは、ダウンロード・ファイルを作成するときに、ファイル定義データベースに対して使用します。この文字列は、#hook_dict テーブルを介して sp_hook_dbmlsync_validate_download_file に渡され、検証論理で使用できます。

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

ファイルベースのダウンロード例

この項には 2 つの例が用意されています。それぞれの例では、統合データベースと 1 つのテーブルのみを使用して、ファイルベースのダウンロードの同期を設定します。1 番目は簡単なスナップショットの例で、2 番目は多少複雑なタイムスタンプベースの例です。

スナップショットの例

この例では、スナップショット同期のファイルベースのダウンロードを実行します。最初に、ファイルベースのダウンロードで必要な 3 つのデータベースを設定し、次に、データをダウンロードする方法を示します。この例は、参考にするだけでもかまいませんし、テキストをコピー・アンド・ペーストしてサンプルを実行することもできます。

サンプル用のデータベースの作成

次のコマンドは、この例で使用される統合データベース、リモート・データベース、ファイル定義データベースの 3 つのデータベースを作成します。

```
dbinit scons.db
dbinit sremote.db
dbinit sfdef.db
```

次のコマンドは、この 3 つのデータベースを起動し、統合データベースへの接続に使用する Mobile Link のデータ・ソース名を作成します。

```
dbeng11 -n sfdef_eng sfdef.db
dbeng11 -n scons_eng scons.db
dbeng11 -n sremote_eng sremote.db
dbdsn -y -w fbd_demo -c "eng=scons_eng;dbf=scons.db;uid=DBA;
  pwd=sql;astart=off;astop=off"
```

Interactive SQL を開き、*scons.db* に接続して、Mobile Link 設定スクリプトを実行します。次に例を示します。

```
read "c:¥Program Files¥SQL Anywhere 11¥MobiLink¥Setup¥syncsa.sql"
```

Mobile Link サーバを起動します。

```
start mlsrv11 -v+ -c "dsn=fbd_demo" -zu+ -ot scons.txt
```

スナップショットの例で使用する統合データベースの設定

この例では、統合データベースには T1 というテーブルが 1 つあります。統合データベースに接続すると、次の SQL を実行してテーブル T1 を作成できます。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);
```

次のコードは、filebased というスクリプト・バージョンを作成し、そのスクリプト・バージョンのダウンロード・スクリプトを作成します。

```
CALL ml_add_table_script( 'filebased',
  'T1', 'download_cursor',
  'SELECT pk, c1 FROM T1' );
```

次のコードは、normal というスクリプト・バージョンを作成し、そのスクリプト・バージョンのアップロード・スクリプトとダウンロード・スクリプトを作成します。

```
CALL ml_add_table_script ( 'normal', 'T1',
  'upload_insert',
  'INSERT INTO T1 VALUES ({ml r.pk}, {ml r.c1})');

CALL ml_add_table_script( 'normal', 'T1',
  'upload_update',
  'UPDATE T1 SET c1 = {ml r.c1} WHERE pk = {ml r.pk} ' );

CALL ml_add_table_script( 'normal', 'T1',
 'upload_delete',
 'DELETE FROM T1 WHERE pk = {ml r.pk}' );

CALL ml_add_table_script( 'normal', 'T1',
  'download_cursor',
  'SELECT pk, c1 FROM T1' );

COMMIT;
```

次のコマンドは、ストアド・プロシージャ begin_pub を作成し、begin_pub が、"normal" スクリプト・バージョンと "filebased" スクリプト・バージョンの両方を対象とした begin_publication スクリプトであることを指定します。

```
CREATE PROCEDURE begin_pub (
      INOUT generation_num integer,
      IN    username       varchar(128),
      IN    pubname        varchar(128) )
BEGIN
   SET generation_num=1;
END;

CALL ml_add_connection_script(
  'filebased',
  'begin_publication',
  '{ call begin_pub(
     {ml s.generation_number},
     {ml s.username},
     {ml s.publication_name} ) }' );

CALL ml_add_connection_script( 'normal',
  'begin_publication',
  '{ call begin_pub(
     {ml s.generation_number},
     {ml s.username},
     {ml s.publication_name} ) }' );
```

スナップショット例で使用するリモート・データベースの作成

この例では、リモート・データベースにも T1 というテーブルが 1 つあります。リモート・データベースに接続し、次の SQL コマンドを実行して、テーブル T1、パブリケーション P1、ユーザ U1 を作成します。また、この SQL は P1 に対する U1 のサブスクリプションも作成します。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
```



```
);

CREATE PUBLICATION P1 (
   TABLE T1
);

CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

次のコードは、sp_hook_dbmlsync_validate_download_file フックを作成して、ユーザ定義の検証論理をリモート・データベースに実装します。

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
   DECLARE udata varchar(256);
SELECT value
   INTO udata
   FROM #hook_dict
   WHERE name = 'user data';
   IF udata <> 'ok' THEN
   UPDATE #hook_dict
   SET value = 'FALSE'
   WHERE name = 'apply file';
   END IF;
END
```

スナップショット例で使用するファイル定義データベースの作成

ファイルベースのダウンロードを使用する Mobile Link システムには、ファイル定義データベースが必要です。このデータベースのスキーマはファイルベースのダウンロードで更新されるリモート・データベースのスキーマと同じですが、データとステータス情報は格納されていません。ファイル定義データベースは、ダウンロード・ファイルに格納されるデータの構造を定義するためだけに使用します。リモート・データベースの Mobile Link グループのユーザ名で定義された、多数のグループのリモート・データベースに対して、1 つのファイル定義データベースを使用できます。

次のコードは、この例で使用するファイル定義データベースを定義します。このコードはリモート・データベースと同じスキーマを作成し、さらに以下を作成します。

- P1 という名前のパブリケーション。T1 テーブルのすべてのローをパブリッシュします。ファイル定義データベースとリモート・データベースでは、同じパブリケーション名を使用する必要があります。
- G1 という名前の Mobile Link ユーザ。このユーザは、ファイルベースのダウンロードで更新されるすべてのリモート・データベースを表しています。
- パブリケーションに対するサブスクリプション

sfdef.db に接続してから、次のコードを実行してください。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
```

```
    TABLE T1
);

CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;
```

初期同期の準備

ダウンロード・ファイルを適用できるようにするために新しいリモート・データベースを準備するには、通常の同期を実行するか、dbmlsync -bg オプションを使用してダウンロード・ファイルを作成します。この例は、通常の同期を実行して新しいリモート・データベースを初期化する方法を示しています。

リモート・データベースの初期同期は、以前に作成した normal というスクリプト・バージョンで実行できます。

```
dbmlsync -c "uid=DBA;pwd=sql;eng=sremote_eng;
  dbf=sremote.db" -v+ -e "sv=normal"
```

スナップショット例におけるファイルベースのダウンロードの実行

統合データベースに接続し、ファイルベースのダウンロードで同期される次のようなデータをいくつか挿入します。

```
INSERT INTO T1 VALUES( 1, 1 );
INSERT INTO T1 VALUES( 2, 4 );
INSERT INTO T1 VALUES( 3, 9 );
COMMIT;
```

次のコマンドは、ファイル定義データベースのあるコンピュータで実行してください。次の処理が行われます。

- dbmlsync -bc オプションにより、ダウンロード・ファイルが作成され、*file1.df* という名前が付けられます。
- -be オプションにより、"OK" という文字列がダウンロード・ファイルに追加され、sp_dbmlsync_validate_download_file フックへのアクセスが可能になります。

```
dbmlsync -c
  "uid=DBA;pwd=sql;eng=sfdef_eng;dbf=sfdef.db"
  -v+ -e "sv=filebased" -bc file1.df -be ok -ot fdef.txt
```

ダウンロード・ファイルを適用するには、-ba オプションと、適用するダウンロード・ファイルの名前を指定して、リモート・データベースで dbmlsync を実行します。

```
dbmlsync -c "uid=DBA;pwd=sql;eng=sremote_eng;
  dbf=sremote.db" -v+ -ba file1.df -ot remote.txt
```

これで、リモート・データベースに変更が適用されました。Interactive SQL を開いてリモート・データベースに接続し、次に示す SQL 文を実行して、リモート・データベースにデータがあることを確認します。

```
SELECT * FROM T1
```


スナップショット例のクリーンアップ

次のコマンドは、3 つのデータベース・サーバをすべて停止してから、ファイルを消去します。

```
del file1.df
mlstop -h -w
dbstop -y -c "eng=sfdef_eng; uid=DBA; pwd=sql"
dbstop -y -c "eng=scons_eng; uid=DBA; pwd=sql"
dbstop -y -c "eng=sremote_eng; uid=DBA; pwd=sql"
dberase -y sfdef.db
dberase -y scons.db
dberase -y sremote.db
```

タイムスタンプベースの例

この例では、タイムスタンプベースの同期のファイルベースのダウンロードを実行します。3 つのデータベースを設定し、次に、ファイルによってデータをダウンロードする方法を示します。この例は、参考にするだけでもかまいませんし、テキストをコピー・アンド・ペーストしてサンプルを実行することもできます。

サンプル用のデータベースの作成

次のコマンドは、この例で使用される統合データベース、リモート・データベース、ファイル定義データベースの 3 つのデータベースを作成します。

```
dbinit tcons.db
dbinit tremote.db
dbinit tfdef.db
```

次のコマンドは、この 3 つのデータベースを起動し、統合データベースへの接続に使用する Mobile Link のデータ・ソース名を作成します。

```
dbeng11 -n tfdef_eng tfdef.db
dbeng11 -n tcons_eng tcons.db
dbeng11 -n tremote_eng tremote.db
dbdsn -y -w tfbd_demo -c "eng=tcons_eng;dbf=tcons.db;uid=DBA;
  pwd=sql;astart=off;astop=off"
```

Interactive SQL を開き、*tcons.db* に接続して、Mobile Link 設定スクリプトを実行します。次に例を示します。

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

Mobile Link サーバを起動します。

```
start mlsrv11 -v+ -c "dsn=tfbd_demo" -zu+ -ot tcons.txt
```

タイムスタンプの例で使用する統合データベースの設定

この例では、統合データベースには T1 というテーブルが 1 つあります。統合データベースに接続すると、次のコードを実行してテーブル T1 を作成できます。

```
CREATE TABLE T1 (
 pk  INTEGER PRIMARY KEY,
 c1  INTEGER,
 last_modified TIMESTAMP DEFAULT TIMESTAMP
);
```

次のコードは、最小限の数のスクリプトで構成される normal というスクリプト・バージョンを定義します。このスクリプト・バージョンは、ファイルベースのダウンロードを「使用しない」同期に使用されます。

```
CALL ml_add_table_script( 'normal', 'T1',
  'upload_insert',
  'INSERT INTO T1( pk, c1) VALUES( {ml r.pk}, {ml r.c1} )' );

CALL ml_add_table_script( 'normal', 'T1',
  'upload_update',
  'UPDATE T1 SET c1 = {ml r.c1} WHERE pk = {ml r.pk} ' );

CALL ml_add_table_script( 'normal', 'T1',
  'upload_delete',
  'DELETE FROM T1 WHERE pk = {ml r.pk}' );

CALL ml_add_table_script( 'normal', 'T1',
  'download_cursor',
  'SELECT pk, c1 FROM T1
    WHERE last_modified >= {ml s.last_table_download}' );
```

次のコードは、すべてのサブスクリプションの世代番号を 1 に設定します。世代番号は、統合データベースが消失または破損し、アップロードが必要となった場合に使用すると便利です。

```
CREATE PROCEDURE begin_pub (
      INOUT generation_num integer,
      IN    username       varchar(128),
      IN    pubname        varchar(128) )
BEGIN
 SET generation_num = 1;
END;

CALL ml_add_connection_script( 'normal',
   'begin_publication',
   '{ call begin_pub(
       {ml s.generation_number},
       {ml s.username},
       {ml s.publication_name},
       {ml s.last_publication_upload},
       {ml s.last_publication_download} ) }' );

COMMIT;
```

次のコードは、filebased というスクリプト・バージョンを定義します。このスクリプト・バージョンは、ファイルベースのダウンロードの作成に使用されます。

```
CALL ml_add_connection_script( 'filebased',
   'begin_publication',
   '{ call begin_pub(
       {ml s.generation_number},
       {ml s.username},
       {ml s.publication_name} ) }' );

CALL ml_add_table_script( 'filebased', 'T1',
   'download_cursor',
   'SELECT pk, c1 FROM T1
       WHERE last_modified >= {ml s.last_table_download}' );
```

次のコードは、最後の 5 日間に発生したすべての変更がダウンロード・ファイルに追加されるように最終ダウンロード時刻を設定します。最後の 5 日間に作成されたどのダウンロード・ファイ

ルも取得していないリモートは、通常の同期を実行しないと、これよりも後のファイルベースのダウンロードを適用することはできません。

```
CREATE PROCEDURE ModifyLastDownloadTimestamp(
      INOUT last_download_timestamp TIMESTAMP,
      IN    ml_username             VARCHAR(128) )
BEGIN
   SELECT dateadd( day, -5, CURRENT TIMESTAMP )
   INTO last_download_timestamp;
END;

CALL ml_add_connection_script( 'filebased',
  'modify_last_download_timestamp',
  'CALL ModifyLastDownloadTimestamp(
    {ml s.last_download}, {ml s.username} )' );

COMMIT;
```

タイムスタンプベースの同期で使用するリモート・データベースの作成

この例では、リモート・データベースにも T1 というテーブルが 1 つあります。リモート・データベースに接続した後、次の SQL コマンドを実行して、テーブル T1、パブリケーション P1、ユーザ U1 を作成します。また、このコードは P1 に対する U1 のサブスクリプションも作成します。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
   TABLE T1
);

CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

次のコードは、sp_hook_dbmlsync_validate_download_file ストアド・プロシージャを定義します。このストアド・プロシージャは、文字列 "ok" が埋め込まれていないダウンロード・ファイルの適用を防止します。

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
   DECLARE udata varchar(256);

   SELECT value
   INTO udata
   FROM #hook_dict
   WHERE name = 'user data';

   IF udata <> 'ok' THEN
     UPDATE #hook_dict
     SET value = 'FALSE'
     WHERE name = 'apply file';
   END IF;
END
```

タイムスタンプベースの同期で使用するファイル定義データベースの作成

次のコードは、タイムスタンプベースの同期で使用するファイル定義データベースを定義します。また、このコードは、テーブル、パブリケーション、ユーザ、そのパブリケーションに対するユーザのサブスクリプションを作成します。

```
CREATE TABLE T1 (
 pk INTEGER PRIMARY KEY,
 c1 INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
);

CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;
```

初期同期の準備

ダウンロード・ファイルを適用できるようにするために新しいリモート・データベースを準備するには、通常の同期を実行するか、dbmlsync -bg オプションを使用してダウンロード・ファイルを作成します。この例では、-bg の使用方法を示します。

次のコードは、統合データベースの filebased_init というスクリプト・バージョンを定義します。このスクリプト・バージョンには、1 つの begin_publication スクリプトがあります。

```
CALL ml_add_table_script(
  'filebased_init', 'T1', 'download_cursor',
  'SELECT pk, c1 FROM T1' );

CALL ml_add_connection_script(
  'filebased_init',
  'begin_publication',
  '{ call begin_pub(
      {ml s.generation_number},
      {ml s.username},
      {ml s.publication_name} ) }' );

COMMIT;
```

次のコマンド・ラインでは、filebased_init というスクリプト・バージョンと -bg オプションを使用して初期ダウンロード・ファイルを作成、適用します。

```
dbmlsync -c "uid=DBA;pwd=sql;eng=tfdef_eng;dbf=tfdef.db"
  -v+ -e "sv=filebased_init" -bc tfile1.df -be ok -bg
  -ot tfdef1.txt

dbmlsync -c "uid=DBA;pwd=sql;eng=tremote_eng;dbf=tremote.db"
  -v+ -ba tfile1.df -ot tremote.txt
```

タイムスタンプベースの同期のファイルベースのダウンロードを実行する

統合データベースに接続し、ファイルベースのダウンロードで同期される次のようなデータをいくつか挿入します。

```
INSERT INTO T1(pk, c1) VALUES( 1, 1 );
INSERT INTO T1(pk, c1) VALUES( 2, 4 );
```



```
INSERT INTO T1(pk, c1) VALUES( 3, 9 );
commit;
```

次のコマンド・ラインは、新しいデータを含むダウンロード・ファイルを作成します。

```
dbmlsync -c
  "uid=DBA;pwd=sql;eng=tfdef_eng;dbf=tfdef.db"
  -v+ -e "sv=filebased" -bc tfile2.df -be ok -ot tfdef2.txt
```

次のコマンド・ラインは、ダウンロード・ファイルをリモート・データベースに適用します。

```
dbmlsync -c "uid=DBA;pwd=sql;eng=tremote_eng;dbf=tremote.db"
  -v+ -ba tfile2.df -ot tfdef3.txt
```

これで、リモート・データベースに変更が適用されました。Interactive SQL を開いてリモート・データベースに接続し、次に示す SQL 文を実行して、リモート・データベースにデータがあることを確認します。

```
SELECT * FROM T1
```

タイムスタンプベースの同期のクリーンアップ

次のコマンドは、3 つのデータベース・サーバをすべて停止してから、ファイルを消去します。

```
del tfile1.df
mlstop -h -w
dbstop -y -c "eng=tfdef_eng; uid=DBA; pwd=sql"
dbstop -y -c "eng=tcons_eng; uid=DBA; pwd=sql"
dbstop -y -c "eng=tremote_eng; uid=DBA; pwd=sql"
dberase -y tfdef.db
dberase -y tcons.db
dberase -y tremote.db
```

Mobile Link イベント

この項では、Mobile Link イベントに対するスクリプトを記述する方法について説明します。

同期スクリプトの作成

目次

同期スクリプトの概要

同期スクリプトを作成して、統合データベースの Mobile Link システム・テーブルに格納したりそこで参照することで、同期処理を制御できます。スクリプトは、SQL、Java、または .NET で作成できます。

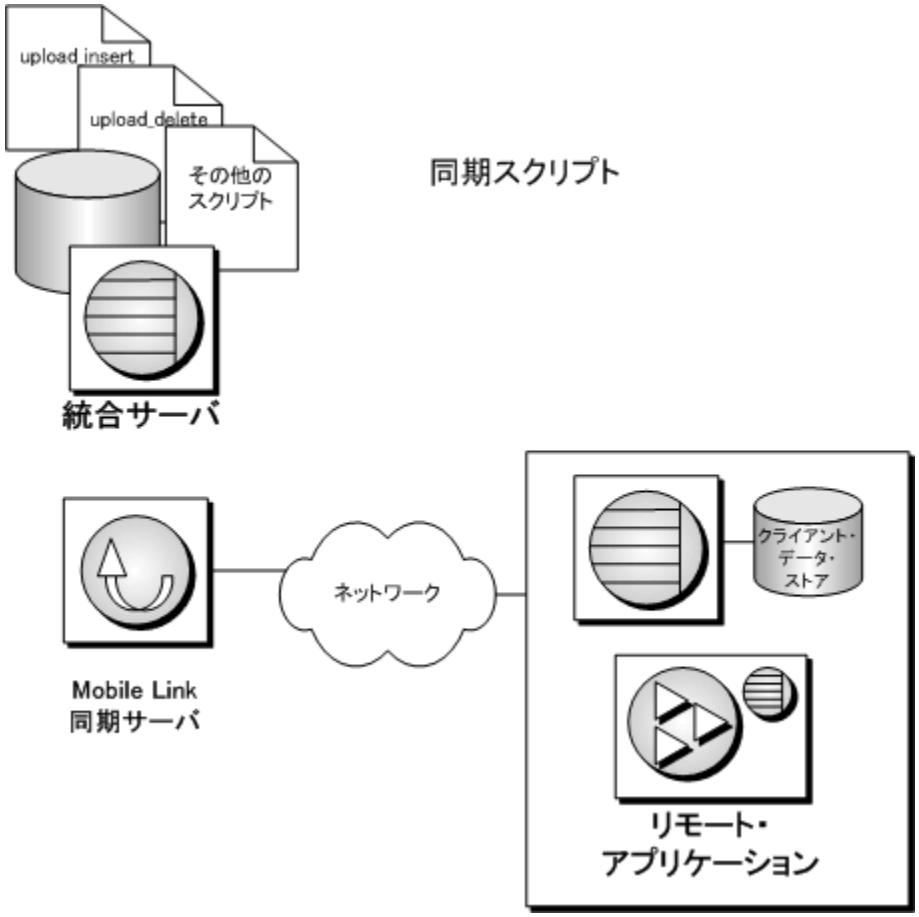
「Mobile Link 同期論理」は、同期スクリプトを使って指定されます。スクリプトでは次の内容を定義します。

- リモート・データベースからアップロードしたデータを、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

スクリプトは個別の文またはストアド・プロシージャ・コールです。統合データベースに格納されるか、統合データベースで参照されます。スクリプトを統合データベースに追加するには、Sybase Central またはシステム・プロシージャを使用できます。

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。

同期中は、Mobile Link サーバがスクリプトを読み込み、統合データベースに対してスクリプトを実行します。



同期処理には複数の手順があります。各手順は、ユニークなイベントによって識別されます。これらのイベントのいずれかに対応するスクリプトを作成することによって、同期処理を制御します。スクリプトは、特定のイベントで特定の動作を行う必要がある場合にのみ作成します。Mobile Link サーバは、イベントが発生するとそれに対応するスクリプトを実行します。特定のイベントに対してスクリプトを定義していない場合、Mobile Link サーバは単に次の手順に進みます。

たとえば、begin_upload_rows というイベントを考えてみます。スクリプトを作成して、このイベントに関連付けることができます。Mobile Link サーバは、このスクリプトを、必要となった時点で初めて読み込み、同期のアップロード・フェーズ中に実行します。スクリプトを作成しなかった場合、Mobile Link サーバは直ちに次の手順、つまり、アップロードされたローを処理する手順に移ります。

テーブル・スクリプトと呼ばれるスクリプトは、イベントだけでなく、リモート・データベースの特定のテーブルにも対応します。Mobile Link サーバは、いくつかのタスクをテーブル単位で実行します (たとえばローのダウンロード)。同じイベントでもアプリケーション・テーブルが異なれば、複数のスクリプトに対応できます。または、いくつかのアプリケーション・テーブルに対しては複数のスクリプトを定義し、他のアプリケーション・テーブルに対しては何も定義しないこともできます。

イベントの概要については、「同期処理」 『Mobile Link - クイック・スタート』を参照してください。

作成可能な各スクリプトの説明については、「同期イベント」 359 ページを参照してください。

スクリプトは、SQL、Java、または .NET で作成できます。この章の内容はすべての種類のスクリプトに適用されますが、主に SQL で同期スクリプトを作成する方法について説明します。

SQL、Java、.NET の説明と比較については、「サーバ側の同期論理の作成オプション」 『Mobile Link - クイック・スタート』を参照してください。

.NET でのスクリプト作成については、「.NET での同期スクリプトの作成」 617 ページを参照してください。

Java でのスクリプト作成については、「Java による同期スクリプトの作成」 553 ページを参照してください。

同期スクリプトを実装する方法については、「同期の方法」 131 ページを参照してください。

簡単な同期スクリプト

Mobile Link には利用可能なイベントがたくさんありますが、その各イベントに対してスクリプトを作成する必要はありません。簡単な同期モデルの場合、必要なスクリプトはわずかです。

テーブルから各リモート・データベースにすべてのローをダウンロードすると、CustDB サンプル・アプリケーションの ULProduct テーブルと同期します。この場合、リモート・データベースでの追加は許可されません。この簡単な形の同期は、1 つのスクリプトで実装できます。この例では、1 つのイベントだけに関連するスクリプトを使用します。

各同期中にダウンロードするローを制御する Mobile Link イベントは、download_cursor イベントと呼ばれます。カーソル・スクリプトには、SELECT 文が必要です。Mobile Link サーバは、このクエリを使用してカーソルを定義します。download_cursor スクリプトの場合、カーソルによって、リモート・データベース内の特定の 1 テーブルにダウンロードするローが選択されます。

CustDB サンプル・アプリケーションには、このサンプル・アプリケーションにある ULProduct テーブルに対応する download_cursor スクリプトが 1 つあり、次のクエリから構成されています。

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

このクエリは結果セットを生成します。クライアントには、この結果セットを構成するローがダウンロードされます。この場合は、テーブルのすべてのローがダウンロードされます。

Mobile Link サーバでは、ULProduct アプリケーション・テーブルへローを送信することを認識しています。これは、このスクリプトが download_cursor イベントと ULProduct テーブルの両方に対応するような方法で統合データベースに格納されているからです。Sybase Central ではこのような対応付けが可能です。

この例では、クエリによって同じ ULProduct という名前の統合テーブルからデータが選択されます。この名前が同じである必要はありません。クエリを書き換えることで、統合データベース内の任意の単一または複数のテーブルのデータを ULProduct アプリケーション・テーブルにダウンロードするようにできます。

より複雑な同期スクリプトを作成することもできます。たとえば、最近修正されたローだけをダウンロードするスクリプトや、リモート・データベースごとに異なる情報を提供するスクリプトを作成できます。

スクリプトと同期処理

各スクリプトは、同期処理の特定のイベントに対応します。特定の動作を行う必要がある場合にのみ、スクリプトを作成します。不要なイベントは、定義しないでおくことができます。

同期処理を大きく 2 つに分けると、アップロードされた情報の処理と、ダウンロードするためのローの準備があります。

Mobile Link サーバは、各スクリプトを、必要となった時点で初めて読み込み、準備します。以後は、イベントが呼び出されるたびにスクリプトが実行されます。

イベントの順序

Mobile Link イベントの完全な順序については、「Mobile Link イベントの概要」 361 ページを参照してください。

アップロード処理の詳細については、「ローをアップロードするスクリプトの作成」 348 ページを参照してください。

ダウンロード処理の詳細については、「ローをダウンロードするスクリプトの作成」 351 ページを参照してください。

注意

- Mobile Link テクノロジによって、複数のクライアントを一度に同期させることができます。この場合、各クライアントは別々の接続を使用して統合データベースにアクセスします。
- 1 つの接続で複数の同期要求を処理できるので、begin_connection イベントと end_connection イベントは特定の同期に依存しません。これらのスクリプトには、パラメータがありません。これらは、接続レベルのスクリプトの例です。
- イベントの中には、各同期に対して 1 回だけ呼び出され、1 つのパラメータを持つものがあります。このパラメータは、同期処理中の Mobile Link クライアントをユニークに識別するユーザ名です。これらも、接続レベルのスクリプトの例です。
- 各テーブルが同期されるたびに 1 回ずつ呼び出されるイベントもあります。これらのイベントに対応するスクリプトは、テーブル・レベルのスクリプトと呼ばれます。これらのスクリプトには、2 つのパラメータがあります。パラメータの 1 つは、同期関数の呼び出しで提供されるユーザ名で、もう 1 つは、同期されるリモート・データベース内のテーブル名です。

  各テーブルは専用のテーブル・スクリプトを持つことができますが、いくつかのテーブルで共有されるテーブル・レベルのスクリプトを作成することもできます。
- begin_synchronization など、一部のイベントは接続レベルとテーブル・レベルの両方で発生します。これらのイベントに対しては、接続スクリプトとテーブル・スクリプトの両方を作成できます。
- 同期処理がどのように複数のトランザクションに分散されるかについては、COMMIT 文が参考になります。
- エラーは、同期処理のあらゆる時点で発生する可能性がある特殊なイベントです。エラーは、次のスクリプトを使用して処理します。

  ```
  handle_error( error_code, error_message, user_name, table_name )
  ```


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」 359 ページを参照してください。

スクリプトの種類

同期スクリプトは、接続全体または指定したテーブルに適用できます。

- **接続レベル・スクリプト**　接続専用または同期専用の、どのリモート・テーブルにも依存しないアクションを実行します。より複雑な同期スキームを実行する際は、このスクリプトを他のスクリプトと組み合わせて使用します。

  「接続スクリプト」 334 ページを参照してください。

- **テーブルレベル・スクリプト**　1 つの同期と特定の 1 つのリモート・テーブル専用のアクションを実行します。競合解決などのより複雑な同期スキームを実行する場合は、このスクリプトを他のスクリプトと組み合わせて使用します。

  「テーブル・スクリプト」 334 ページを参照してください。

接続スクリプト

接続レベルのスクリプトは、特定のテーブルに関連付けられていない高いレベルのイベントを制御します。これらのイベントは、各同期の処理中に必要な全般的なタスクを実行するときに使用します。

接続スクリプトは、接続と切断に関連するアクションや、アップロード処理やダウンロード処理の開始と終了などの、同期レベルのイベントのアクションを制御します。一部の接続スクリプトには、関連するテーブル・スクリプトがあります。これらの接続スクリプトは、テーブルが同期されているかどうかに関係なく、いつでも呼び出されます。

接続レベルのスクリプトは、特定のイベントで特定のアクションを実行する必要がある場合にのみ作成します。少数のイベント用のスクリプトだけを作成すれば済む場合もあります。あらゆるイベントに対する Mobile Link サーバのデフォルト・アクションは、何もアクションを実行しない設定になっています。簡単な同期スキームの中には、接続スクリプトが必要でないものもあります。

ml_global スクリプト・バージョン

同じスクリプトを何回も定義しないで済むように、接続レベル・スクリプトを 1 回定義して、再使用できます。これを行うには、ml_global と呼ばれるスクリプト・バージョンを定義します。

「ml_global スクリプト・バージョン」 342 ページを参照してください。

テーブル・スクリプト

テーブル・スクリプトによって、ローのアップロードの開始や終了、競合の解決、ダウンロードするローの選択など、特定のテーブルの同期に関する特定のイベントでのアクションを実行できます。

テーブルの同期スクリプトは、統合データベースのあらゆるテーブル (またはテーブルの組み合わせ) を参照できます。この機能を使用して、1 つまたは複数の統合テーブルに格納されたデー

タを特定のリモート・テーブルに入れたり、1 つのリモート・テーブルからアップロードされたデータを統合データベースの複数のテーブルに格納したりできます。

テーブル名は一致しなくてもよい

リモート・データベースでのテーブル名と統合データベースでのテーブル名は、同じである必要はありません。Mobile Link サーバは、ml_table システム・テーブルでリモート・テーブル名を検索し、テーブルに対応するスクリプトを特定します。

スクリプトのパラメータ

同期スクリプトのほとんどは、Mobile Link サーバからパラメータを受け取ることができます。各スクリプトで使用できるパラメータの詳細については、「同期イベント」 359 ページを参照してください。

次のいずれかの方法によって、SQL スクリプトでパラメータを指定できます。

- 疑問符
- 名前付きスクリプト・パラメータ

疑問符によって表されるスクリプト・パラメータ

疑問符を使ってパラメータを表すのは ODBC 規則です。Mobile Link SQL スクリプトで疑問符を使用するには、SQL スクリプトで各パラメータに対して 1 つ疑問符を置きます。Mobile Link サーバが、各疑問符をパラメータ値に置き換えます。パラメータ値への置き換えは、スクリプト内でパラメータが定義されている順序に従って行われます。

パラメータは、「同期イベント」 359 ページで指定された順に置いてください。一部のパラメータは省略可能です。パラメータが省略可能になるのは、後続のパラメータを指定しない場合に限られます。たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を必ず使用してください。

名前付きスクリプト・パラメータ

Mobile Link には、スクリプトで疑問符の代わりに使用できる名前付きパラメータが用意されています。名前付きパラメータには、次のような利点があります。

- 使用可能なパラメータの任意のサブセットを任意の順序で指定できます。
- in/out パラメータを除き、スクリプト内で同じ名前付きパラメータを複数回指定できます。
- 名前付きパラメータを使用する場合には、スクリプトでリモート ID を指定できます。スクリプトでリモート ID を指定できるのは、この方法だけです。
- 独自の名前付きパラメータを作成できます。「ユーザ定義の名前付きパラメータ」 338 ページを参照してください。

1 つのスクリプト内で名前付きパラメータと疑問符を混在させることはできません。

Mobile Link 名前付きパラメータには 4 種類あります。名前付きパラメータを指定するには、次のように種類をプレフィクスとして付ける必要があります。

名前付きパラメータの種類	プレフィクス	例
システム・パラメータ	s.	{ml s.remote_id}
ロー・パラメータ (カラムの名前。カラム名にスペースが含まれる場合は二重引用符または角カッコで囲む)。	r.	{ml r.cust_id} {ml r."Column name"}

名前付きパラメータの種類	プレフィクス	例
古いロー・パラメータ (プレイメージのカラムの値を指定するために upload_update スクリプトのみで使用。カラム名にスペースが含まれる場合は、二重引用符または角カッコで囲む)。	o.	{ml o.cust_name} {ml o."Column name"}
認証パラメータ。「認証パラメータ」 339 ページを参照してください。	a.	{ml a.1}
ユーザ定義のパラメータ。「ユーザ定義の名前付きパラメータ」 338 ページを参照してください。	ui.	{ml ui.varname}

スクリプト・パラメータを名前で参照するには、**{ml** *parameter* **}** のように、パラメータを中カッコで囲み、前に ml を付けます。たとえば、**{ml s.action_code}** のように指定します。中カッコによる表記は ODBC 規則です。

便宜上、Mobile Link スクリプト・パラメータと同じ名前のスキーマ名がコード・セクション内に含まれていないかぎり、中カッコには大きなコード・セクションを含めることができます。たとえば、次の upload_insert スクリプトはどちらも有効で同じ内容を表しています。

```
INSERT INTO t ( id, c0 ) VALUES( {ml r.id}, {ml r.c0} )
```

および

```
{ml  INSERT INTO t ( id, c0 ) VALUES( r.id, r.c0 ) }
```

注意
Mobile Link の**同期モデル作成ウィザード**でリモート・データベース内のカラムを生成していない場合に名前付きロー・パラメータを使用するには、ml_add_column システム・プロシージャを使用してカラム情報を統合データベースに格納する必要があります。「ml_add_column system procedure」 700 ページを参照してください。

スクリプト・パラメータのコメント

次の形式のコメントが認識されます。

- 二重ハイフン・プレフィクス (--)
- 二重スラッシュ・プレフィクス (//)
- ブロック・コメント (/* */)

初めの 2 つの形式を使用すると、行の最後までのスクリプト・テキストが無視されます。3 つ目の形式を使用すると、/* プレフィクスと */ サフィックスの間にあるすべてのスクリプト・テキストが無視されます。ブロック・コメントはネストできません。

その他のタイプのベンダ固有コメントは認識されないため、名前付きパラメータへの参照のコメントには使用しないでください。

ユーザ定義の名前付きパラメータ

独自のパラメータを定義することもできます。独自のパラメータは、ユーザ定義の変数を使用できない RDBMS に特に便利です。

ユーザ定義のパラメータは、最初に参照されるときに定義され、NULL に設定されます。パラメータには、ui とピリオド (ui.) のプレフィクスを付ける必要があります。ユーザ定義のパラメータは 1 つの同期が終わるまで値が維持され、別の同期が開始されるときに NULL に設定されます。ユーザ定義のパラメータは in/out です。

ユーザ定義のパラメータは通常、テーブルに格納しないでステータス情報にアクセスするために使用します。テーブルに格納するには、複雑なジョインが必要です。

例

たとえば、var1 という変数を custom_value に設定する MyCustomProc というストアド・プロシージャを作成するとします。

```
CREATE PROCEDURE MyCustomProc(
  IN username VARCHAR (128), INOUT var1 VARCHAR (128)
)
begin
 SET var1 = 'custom_value';
end
```

次の begin_connection スクリプトでは、ユーザ定義のパラメータ var1 を定義し、値を custom_value に設定しています。

```
CALL ml_add_connection_script (
  'version1',
  'begin_synchronization',
  '{call MyCustomProc( {ml s.username}, {ml ui.var1} )}' );
```

次の begin_upload スクリプトでは、値が custom_value の var1 を参照しています。

```
CALL ml_add_connection_script (
  'version1',
  'begin_upload',
  'update SomeTable set some_column = 123 where some_other_column = {ml ui.var1}' );
```

最初のパラメータを in/out に定義する MyPFDProc という別のストアド・プロシージャがあるとします。次の prepare_for_download スクリプトでは、var1 の値を pfd_value に変更しています。

```
CALL ml_add_connection_script (
  'version1',
  'prepare_for_download',
  '{call MyPFDProc( {ml ui.var1} )}' );
```

次の begin_download スクリプトでは、値が pfd_value の var1 を参照しています。


```
CALL ml_add_connection_script (
  'version1',
  'begin_download',
  'insert into SomeTable values( {ml s.username}, {ml ui.var1} )' );
```

認証パラメータ

Mobile Link スクリプトでは、認証パラメータは、{ml a.1} のように先頭に文字 a のプレフィクスが付いた名前付きパラメータです。パラメータは 1 から始まり、上限が 255 の数値です。値は Mobile Link クライアントから送信されます。

authenticate_* スクリプトで使用すると、認証パラメータによって認証情報が渡されます。

認証パラメータは他のすべてのイベント (begin_connection と end_connection を除く) で Mobile Link クライアントからの情報を渡すために使用できます。この方法は、通常ならテーブルを作成し、データを移植する必要があるような処理を行うのに便利です。

SQL Anywhere リモートでは、dbmlsync -ap オプションを指定して情報を渡します。Ultra Light リモートでは、auth_parms と num_auth_parms を指定して情報を渡します。

参照

- 「スクリプトのパラメータ」 336 ページ
- dbmlsync：「-ap オプション」 『Mobile Link - クライアント管理』
- Ultra Light：「Authentication Parameters 同期パラメータ」 『Ultra Light データベース管理とリファレンス』と「Number of Authentication Parameters パラメータ」 『Ultra Light データベース管理とリファレンス』

例

Ultra Light リモート・データベースでは、ul_synch_info 構造体の num_auth_parms フィールドと auth_parms フィールドを使用して、パラメータを渡します。num_auth_parms は、パラメータの数で、0 ～ 255 の値になります。auth_parms は、文字列の配列へのポインタです。文字列がプレーン・テキストとして表示されるのを防ぐため、文字列はパスワードと同じ方法で送信されます。num_auth_parms が 0 の場合、auth_parms は NULL に設定します。次に、Ultra Light でパラメータを渡す例を示します。

```
ul_char * Params[ 3 ] = { UL_TEXT( "param1" ),
  UL_TEXT( "param2" ), UL_TEXT( "param3" ) };

...
info.num_auth_parms = 3;
info.auth_parms = Params;
```

SQL Anywhere のリモート・データベースでは、dbmlsync -ap オプションを使用して、認証パラメータをカンマで区切られたリストで渡します。たとえば、次のコマンド・ラインは 3 つのパラメータを渡します。

```
dbmlsync -ap "param1,param2,param3"
```

サーバでは、スクリプトは送信された順序で参照します。この例では、authenticate_parameters スクリプトは以下のようになります。

```
CALL my_auth_parm (
  {ml s.authentication_status},
  {ml s.remote_id},
  {ml s.username},
  {ml a.1},
  {ml a.2},
  {ml a.3}
)
```


スクリプト・バージョン

スクリプトは、「スクリプト・バージョン」と呼ばれるグループに分けられます。バージョンを指定することによって、アップロードの処理やダウンロードの準備に使用する同期スクリプト・セットを Mobile Link クライアントで選択できます。

統合データベースにスクリプト・バージョンを追加する方法については、「スクリプト・バージョンの追加」 342 ページを参照してください。

スクリプト・バージョンの使用方法

スクリプト・バージョンを使用すると、スクリプトを異なる環境で実行されるスクリプト・セットに編成できます。この機能によって柔軟性が生まれ、特に次のような場合に便利です。

- **アプリケーションのカスタマイズ**　さまざまなリモート・ユーザからの情報を処理するために、異なるスクリプト・セットを使用します。たとえば、組織のマネージャが自分が持っているデータベースを同期させるときに使用するスクリプト・セットを、他の人が使用するスクリプト・セットとは別に作成できます。これと同じ機能を 1 つのスクリプト・セットで実現することもできますが、より複雑なスクリプトになってしまいます。
- **アプリケーションのアップグレード**　データベース・アプリケーションをアップグレードする場合、新しいスクリプトが必要になることがあります。これは、新しいバージョンのアプリケーションではデータ処理方法が異なる場合があるからです。リモート・データベースが変更される場合はほとんど、新しいスクリプトが必要になります。通常は、すべてのユーザを同時にアップグレードすることはできません。Mobile Link クライアントは、同期中に新しいスクリプト・セットの使用を要求できます。古いスクリプトと新しいスクリプトの両方がサーバ上で共存できるため、使用するアプリケーションのバージョンに関わらず、すべてのユーザが同期できます。
- **複数のアプリケーションの管理**　1 つの Mobile Link サーバが 2 つの完全に異なるアプリケーションを同期しなければならない場合があります。たとえば、販売アプリケーションを使用する従業員がいる一方で、在庫管理用に設計されたアプリケーションが必要な従業員もいます。2 つのアプリケーションで異なるデータ・セットが必要な場合は、各アプリケーションにつき 1 つのバージョンとなるよう、2 つのバージョンの同期スクリプトを作成できます。
- **スクリプト・バージョンのプロパティの設定**　.NET または Java 同期論理のクラスから、参照可能なスクリプト・バージョンのプロパティを設定できます。「ml_add_property システム・プロシージャ」 713 ページを参照してください。

バージョン名の割り当て

スクリプト・バージョン名は、文字列です。統合データベースにスクリプトを追加するときに、この名前を指定します。たとえば、ml_add_connection_script ストアド・プロシージャと ml_add_table_script 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場合、スクリプト・バージョン名が 1 つ目のパラメータになります。また、Sybase Central を使用してスクリプトを追加する場合は、バージョン名を入力するように指示されます。

スクリプト・バージョンには、**ml_sis_1** や **ml_qa_1** という名前は使用できません。これらの名前は、Mobile Link によって内部的に使用されています。

警告
スクリプト・バージョンの名前は、**ml_** で始めないでください。ml_ で始まるスクリプト・バージョンは内部用に予約されています。

同期のバージョンの指定

同期が開始されるときにリモート・サイトでスクリプト・バージョンが指定されていない場合、同期は失敗します。

ml_global スクリプト・バージョン

他のスクリプト・バージョンとは使い方が異なる、**ml_global** と呼ばれるスクリプト・バージョンを作成できます。ml_global というスクリプト・バージョンを作成する場合、一度定義すると、関連付けられた接続スクリプトがすべての同期で自動的に使用されます。ml_global をスクリプト・バージョンとして明示的に指定することはありません。

ml_global スクリプト・バージョンでスクリプトを定義した後、同期対象として指定したスクリプト・バージョンの同じイベントに対してスクリプトを定義した場合には、指定したスクリプト・バージョンが使用されます。ml_global スクリプト・バージョンのスクリプトが使用されるのは、同期中のプライマリ・スクリプト・バージョンでそのスクリプトが定義されていない場合だけです。

ml_global スクリプト・バージョンには、接続レベル・スクリプトだけを含めることができます。スクリプト・バージョンを 1 つしか使用していない場合、ml_global スクリプト・バージョンは不要で、役に立たない可能性もあります。

スクリプト・バージョンの追加

すべてのスクリプトはスクリプト・バージョンに関連付けられています。Sybase Central の管理モードで作業している場合、統合データベースにバージョン名を追加してから、接続スクリプトを追加してください。システム・プロシージャを使ってスクリプトを追加する場合には、新しいバージョン名がスクリプトによって自動的に追加されます。Sybase Central モデル・モードでは、1 つのスクリプト・バージョンだけが使用可能で、デフォルトでモデルと同じ名前が付けられます。

「スクリプト・バージョン」 341 ページを参照してください。

♦ データベースにスクリプト・バージョンを追加するには、次の手順に従います (Sybase Central の管理モードの場合)。

1. Sybase Central で **[接続] - [Mobile Link 11 に接続]** を選択し、統合データベースに接続します。
2. **[バージョン]** フォルダを右クリックし、**[ファイル] - [新規] - [バージョン]** を選択します。
3. **スクリプト・バージョン作成ウィザード**の指示に従います。

♦ データベースからスクリプト・バージョンを削除するには、次の手順に従います (Sybase Central の管理モードの場合)。

1. Sybase Central で **[接続] - [Mobile Link 11 に接続]** を選択し、統合データベースに接続します。
2. **[バージョン]** フォルダをクリックします。
3. 右ウィンドウ枠で、バージョン名を右クリックし、**[削除]** を選択します。
4. **[はい]** をクリックします。

♦ データベースにスクリプト・バージョンを追加するには、次の手順に従います (システム・プロシージャの場合)。

- スクリプト・バージョンは、接続スクリプトまたはテーブル・スクリプトを追加するのと同じ操作で追加できます。

  詳細については、「スクリプトを追加または削除するためのシステム・プロシージャ」 698 ページを参照してください。

必要なスクリプト

Mobile Link サーバを実行する場合には、特定のスクリプトが必要です。必要なスクリプトは、双方向同期、アップロード専用の同期、ダウンロード専用の同期のどれを行っているかによって決まります。

双方向またはアップロード専用の同期の場合、Mobile Link では次のテーブル・スクリプトの少なくとも 1 つが必要です。

- upload_delete
- upload_insert
- upload_new_row_insert
- upload_old_row_insert
- upload_update
- または、ダイレクト・ロー・ハンドリングでアップロードを処理している場合には、handle_UploadData 接続イベント用のスクリプトが必要。

双方向またはダウンロード専用の同期の場合、Mobile Link では同期のすべてのテーブルでダウンロード・テーブル・スクリプト (download_cursor または download_delete_cursor) が必要です。または、ダイレクト・ロー・ハンドリングでダウンロードを処理している場合には、handle_DownloadData 接続スクリプトを指定する必要があります。このスクリプトを空にして、他のイベントでダウンロードを処理できます。

デフォルトでは、必要なスクリプトがない場合、同期はアボートされます。Mobile Link サーバの -fr オプションを使用して、この動作を無効にできます。

「-fr オプション」 72 ページを参照してください。

スクリプトの追加と削除

同期モデル作成ウィザードを使用すると、モデルの配備時にスクリプトが自動的に統合データベースに追加されます。

Sybase Central モデル・モード以外で同期スクリプトを作成する場合は、統合データベース内の Mobile Link システム・テーブルにそのスクリプトを追加してください。SQL スクリプトの場合には、スクリプト全体が Mobile Link システム・テーブルに保存されます。Java または .NET スクリプトの場合には、メソッド名がシステム・テーブルに登録されます。スクリプトの保存方法とメソッド名の保存方法は、ほぼ同じです。

「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

Sybase Central を使用している場合は、データベースに同期バージョンを追加してから、個々のスクリプトを追加してください。「スクリプト・バージョンの追加」 342 ページを参照してください。

♦ 接続スクリプトを追加するには、次の手順に従います (Sybase Central の管理モードの場合)。

1. **[接続] - [Mobile Link 11 に接続]** を選択し、統合データベースに接続します。
2. **[接続スクリプト]** を右クリックし、**[新規] - [接続スクリプト]** を選択します。
3. **接続スクリプトの作成ウィザード**の指示に従います。

♦ 接続スクリプトを削除するには、次の手順に従います (Sybase Central の管理モードの場合)。

1. **[接続] - [Mobile Link 11 に接続]** を選択し、統合データベースに接続します。
2. **[接続スクリプト]** を展開します。
3. 接続スクリプトを右クリックして、**[削除]** を選択します。
4. **[はい]** をクリックします。

♦ 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 (Sybase Central の管理モードの場合)。

1. **[接続] - [Mobile Link 11 に接続]** を選択し、統合データベースに接続します。
2. **[同期テーブル]** を展開します。
3. テーブルを右クリックして、**[新規] - [テーブル・スクリプト]** を選択します。
4. **テーブル・スクリプト作成ウィザード**の指示に従います。

♦ テーブル・スクリプトを削除するには、次の手順に従います (Sybase Central の管理モードの場合)。

1. **[接続] - [Mobile Link 11 に接続]** を選択し、統合データベースに接続します。
2. **[同期テーブル]** を展開します。

3. テーブルを展開します。
4. テーブル・スクリプトを右クリックして、**[削除]** を選択します。
5. **[はい]** をクリックします。

♦ すべてのタイプのスクリプトを追加または削除するには、次の手順に従います (システム・プロシージャの場合)。

- 統合データベースの設定時にインストールされるストアド・プロシージャを使用して、スクリプトを統合データベースに追加、または統合データベースから削除できます。

  スクリプトを追加または削除できるストアド・プロシージャについては、以下の項目を参照してください。

  - 「ml_add_connection_script システム・プロシージャ」 702 ページ
  - 「ml_add_table_script システム・プロシージャ」 716 ページ
  - 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ
  - 「ml_add_dnet_table_script システム・プロシージャ」 704 ページ
  - 「ml_add_java_connection_script システム・プロシージャ」 705 ページ
  - 「ml_add_java_table_script システム・プロシージャ」 706 ページ

スクリプトの直接挿入

ほとんどの場合、ストアド・プロシージャまたは Sybase Central を使用してスクリプトをシステム・テーブルに挿入することをおすすめします。ただし、INSERT 文を使用してスクリプトを直接挿入することが必要な場合もあります。たとえば、旧バージョンの RDBMS では、長さの制限があって、ストアド・プロシージャを使用するのが難しい場合があります。

Mobile Link システム・テーブルの完全な説明については、「Mobile Link サーバのシステム・テーブル」 733 ページを参照してください。

スクリプトを直接挿入するのに必要な INSERT 文のフォーマットは、ストアド・プロシージャ ml_add_connection_script と ml_add_table_script のソース・コードにあります。これらのストアド・プロシージャのソース・コードは、Mobile Link 設定スクリプトにあります。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。設定スクリプトはすべて *install-dir¥MobiLink¥setup* にあります。ファイル名は次のとおりです。

統合データベース	設定ファイル
Adaptive Server Enterprise	*syncase.sql*
IBM DB2 メインフレーム	*syncd2m.sql* または *syncd2m_jcl.sql*
IBM DB2 LUW	*syncdb2.sql*
Microsoft SQL Server	*syncmss.sql*

統合データベース	設定ファイル
MySQL	*syncmys.sql*
Oracle	*syncora.sql*
SQL Anywhere	*syncsa.sql*

スクリプトの無視

統合データベースに upload_insert、upload_update、upload_delete スクリプトがないテーブルに対するデータの挿入、更新、削除がアップロード・ストリームに含まれる場合、またはテーブルのダウンロード・スクリプト (download_cursor スクリプトと download_delete_cursor スクリプト) がない場合、Mobile Link サーバは次のいずれかの処理を行います。

- Mobile Link サーバが -fr を指定して起動されなかった場合は、見つからないスクリプトについてエラーを通知し、同期を中断する
- Mobile Link サーバが -fr を指定して起動された場合は、データの不整合に関する警告メッセージを表示する

Mobile Link サーバ・コマンド・オプション -zwd を指定すると、この警告メッセージを抑制することができますが、このオプションにより、すべての同期テーブルに関する警告メッセージが抑制されてしまいます。

Mobile Link サーバは、プレフィクス --{ml_ignore} が含まれるすべての接続スクリプトやテーブル・スクリプトについて、別の扱いをします。Mobile Link サーバでは、これらのスクリプトが意図的に無視するスクリプトとして認識されます。より厳密には、upload_insert、upload_update、upload_delete スクリプトのある同期テーブルに対するデータの挿入、更新、削除がアップロード・ストリームに含まれていて、これらのスクリプトでプレフィクス --{ml_ignore} が指定されている場合、Mobile Link サーバは統合データベースに対してこれらのスクリプトを実行しません。さらに、-fr オプションを指定してサーバが起動されたかどうかに関係なく、エラーや警告のメッセージを表示しないで同期を続けます。

このロジックはダウンロードにも当てはまります。ただし、意図的な無視されたスクリプトや、実際の upload_delete (upload_insert または upload_update) スクリプトがない同期テーブルに対するデータの削除 (挿入または更新) がアップロード・ストリームに含まれる場合、このテーブルに対する意図的に無視されたスクリプトや実際の upload_insert または upload_update スクリプトが存在する場合でも、Mobile Link サーバは同期を中断したり警告メッセージを表示したりします。

ローをアップロードするスクリプトの作成

Mobile Link サーバに、リモート・データベースから受信したアップロード・ストリーム・データを処理する方法を通知するには、アップロード・スクリプトを定義します。リモート・データベースで更新、挿入、または削除するローを処理する個別のスクリプトを作成します。簡単な実装によって、対応するアクション (更新、挿入、削除) を統合データベースで実行できます。

Mobile Link サーバは、データを 1 つのトランザクションでアップロードします。アップロード処理の説明については、「アップロード中のイベント」 368 ページを参照してください。

.NET 同期論理でローをアップロードする方法については、「ローのアップロードまたはダウンロード」 630 ページを参照してください。

注意

- 各リモート・テーブルの begin_upload スクリプトと end_upload スクリプトには、更新される個々のローには依存しない論理が保持されます。
- アップロードは、1 ローずつの挿入、更新、削除から構成されます。通常、これらのアクションは、upload_insert、upload_update、upload_delete の各スクリプトを使用して実行されます。
- SQL Anywhere クライアント用のアップロードの準備において、dbmlsync ユーティリティは、正常に行われた最後の同期よりも後に書き込まれたすべてのトランザクション・ログにアクセスする必要があります。「トランザクション・ログ・ファイル」 『Mobile Link - クライアント管理』を参照してください。
- Mobile Link クライアント・バージョン 9.0 以前を使用してリモート・データベースを同期する場合、または upload_insert イベント、upload_new_row_insert イベント、upload_old_row_insert イベントでのプレースホルダとして名前付きパラメータの代わりに疑問符を使用する場合、Mobile Link サーバは、リモート・データベースで定義されたときのテーブルのカラム順序を使用します。イベント文のカラム順序は、リモート・データベースで定義されたときのカラム順序と一致する必要がありますが、統合データベースのテーブルおよびカラム名は、リモート・データベースのテーブル名およびカラム名と異なっていてもかまいません。

  次の INSERT 文は、リモート・データベースで emp_id よりも前に emp_name が定義された場合にのみ使用されます。

```
INSERT INTO emp (emp_name, emp_id)
VALUES (?, ?);
```

upload_insert スクリプトの作成

Mobile Link サーバは、アップロードの処理中にこのイベントを使用して、リモート・データベースに挿入されたローを処理します。

次の INSERT 文は、upload_insert スクリプトで使用されます。

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


注意

- upload_new_row_insert イベントと upload_old_row_insert イベントは、remote_id と user_name を追加のパラメータとして受け入れます。これらのパラメータは、テーブルの完全なカラム・リストの前に記述されている必要があります。

参照

- 「ローをアップロードするスクリプトの作成」 348 ページ
- 「upload_insert テーブル・イベント」 530 ページ

upload_update スクリプトの作成

Mobile Link サーバは、アップロードの処理中にこのイベントを使用して、リモート・データベースで更新されたローを処理します。次の UPDATE 文は、upload_update 文の使用方法を示しています。

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

注意

- Mobile Link クライアント・バージョン 9.0 以前を使用してリモート・データベースを同期する場合、およびプレースホルダとして名前付きパラメータの代わりに疑問符を使用する場合は、パラメータの数を次のいずれかにすることができます。
  - ○ 非プライマリ・キー・カラムの数 + プライマリ・キー・カラムの数。
  - ○ 2 * (非プライマリ・キー・カラムの数 + プライマリ・キー・カラムの数)。

  カラム順序は、非プライマリ・キー・カラムを先に、次のどちらかのカラムを後にして構成してください。
  - ○ プライマリ・キー・カラム。
  - ○ すべてのカラム。

参照

- 「ローをアップロードするスクリプトの作成」 348 ページ
- 「upload_update テーブル・イベント」 548 ページ

upload_delete スクリプトの作成

Mobile Link サーバは、アップロードの処理中にこのイベントを使用して、リモート・データベースから削除されたローを処理します。次の文は、upload_delete 文の使用方法を示しています。

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

注意

- Mobile Link クライアント・バージョン 9.0 以前を使用してリモート・データベースを同期する場合、およびプレースホルダとして名前付きパラメータの代わりに疑問符を使用する場合は、パラメータの数を次のいずれかにしてください。
  - プライマリ・キー・カラムの数。
  - すべてのカラムの数。

参照

- 「ローをアップロードするスクリプトの作成」 348 ページ
- 「upload_delete テーブル・イベント」 523 ページ

upload_fetch スクリプトの作成

upload_fetch スクリプトは、統合データベースのテーブルにカーソルを定義する SELECT 文です。このカーソルは、リモート・データベースから更新されたものとして受信したローの古い値と、統合データベースにある値を比較するために使用します。これによって、upload_fetch スクリプトは更新の処理中に競合を識別します。

同期テーブルが、次のように定義されている場合を考えてみます。

```
CREATE TABLE uf_example (
 pk1 integer NOT NULL,
 pk2  integer NOT NULL,
 val  varchar(200),
 PRIMARY KEY( pk1, pk2 ));
```

この場合、このテーブルに対して考えられる upload_fetch スクリプトは次のようになります。

```
SELECT pk1, pk2, val
FROM uf_example
WHERE pk1 = {ml r.pk1} and pk2 = {ml r.pk2}
```

「upload_fetch テーブル・イベント」 525 ページを参照してください。

Mobile Link サーバが、統合データベース内の競合のチェック対象となる 1 つのローを正確に識別するには、upload_fetch スクリプト内にクエリの WHERE 句が必要となります。

ローをダウンロードするスクリプトの作成

ダウンロード・トランザクション時に各テーブルの処理に使用できるスクリプトは、2 つあります。挿入と更新を実行する download_cursor スクリプトと、削除を実行する download_delete_cursor スクリプトです。

これらのスクリプトは、SELECT 文か、結果セットを返すプロシージャの呼び出しのどちらかです。Mobile Link サーバは、スクリプトの結果セットをリモート・データベースにダウンロードします。Mobile Link クライアントは自動的に、download_cursor スクリプトの結果セットに基づいてローを挿入または更新し、download_delete_cursor イベントに基づいてローを削除します。

ストアド・プロシージャの使用の詳細については、「ストアド・プロシージャ・コールからの結果セットのダウンロード」 167 ページを参照してください。

Mobile Link サーバは、データを 1 つのトランザクションでダウンロードします。ダウンロード処理の説明については、「ダウンロード中のイベント」 370 ページを参照してください。

注意

- アップロードと同様、ダウンロードも接続イベントで開始、終了します。他のイベントは、テーブル・レベルのイベントです。
- SendDownloadAck 設定を ON に変更した場合、サーバの動作は、使用しているダウンロード確認モードによって異なります。ブロッキング・ダウンロード確認の場合、ダウンロードの確認をクライアントから受け取らなかった場合、統合データベースではダウンロード・トランザクション全体がロールバックされます。非ブロッキング・ダウンロード確認の場合、ダウンロード・トランザクションはコミットされますが、ダウンロード・タイムスタンプ更新スクリプトとダウンロード確認スクリプトは確認を受信するまで実行されません。

  デフォルトでは、SendDownloadAck は OFF に設定されています。

  「SendDownloadACK (sa) 拡張オプション」 『Mobile Link - クライアント管理』、「Send Download Acknowledgement 同期パラメータ」 『Ultra Light データベース管理とリファレンス』、「-nba オプション」 76 ページ、「nonblocking_download_ack 接続イベント」 495 ページ、「publication_nonblocking_download_ack 接続イベント」 500 ページを参照してください。
- 各リモート・テーブルの begin_download スクリプトと end_download スクリプトには、更新される個々のローには依存しない論理が保持されます。
- タイムスタンプベースのダウンロードの場合は、last_download_timestamp パラメータを指定して、最後の同期以降の変更のみがダウンロードされるようにします。たとえば、download_cursor または download_delete_cursor SQL スクリプトには次の行を挿入できます。

  ```
  WHERE Customer.last_modified >= {ml s.last_table_download}
  ```

  「スクリプトでの最終ダウンロード時刻の使用」 134 ページを参照してください。
- ダウンロードでは、挿入と更新が区別されません。download_cursor イベントに対応するスクリプトは、ダウンロードされるローを定義する SELECT 文です。クライアントは、ローが存在するかどうかを調べ、適切な挿入操作または更新操作を実行します。
- 参照整合性違反を避けるために必要であれば、ダウンロード処理の最後にクライアントは自動的にローを削除します。

警告
以前の展開によって作成されたシャドー・テーブルを同期しないでください (たとえば、*_mod* または *_del* で終わるテーブルを同期しないでください)。これらのテーブルが必要になるのは、変更または削除されたローを統合データベースで追跡する場合のみです。

「参照整合性と同期」『Mobile Link - クイック・スタート』を参照してください。

download_cursor スクリプトの作成

統合データベースからリモート・データベースに情報をダウンロードするには、download_cursor スクリプトを作成します。このスクリプトは、変更をダウンロードするリモート・データベースの各テーブルに 1 つ作成してください。他のスクリプトを使用するとダウンロード処理をカスタマイズできますが、それらは必要ありません。

- 各 download_cursor スクリプトには、SELECT 文か、SELECT 文を含むプロシージャの呼び出しが必要です。Mobile Link サーバは、SELECT 文を使用して統合データベース内でカーソルを定義します。
- スクリプトでは、対応するリモート・データベース内のテーブルのカラムに対応するすべてのカラムを選択します。統合データベース内のカラムは、対応するリモート・データベースのカラムとは異なる名前にできますが、互換性のある型にしてください。

例

次のスクリプトは、従業員情報を格納しているリモート・テーブルの download_cursor スクリプトとして機能します。Mobile Link サーバは、この SQL 文を使用してダウンロード・カーソルを定義します。このスクリプトによって、すべての従業員についての情報がダウンロードされます。

```
SELECT emp_id, emp_fname, emp_lname
FROM employee;
```

Mobile Link サーバは、スクリプトへ特定のパラメータを渡します。これらのパラメータを使用するには、名前付きパラメータを使用するか、SQL 文に疑問符を挿入できます。後者の場合、Mobile Link サーバは、パラメータ値に置き換えてから、統合データベースに対して文を実行します。次のスクリプトは、名前付きパラメータの使用方法を示しています。

```
CALL ml_add_table_script(
  'Lab',
  'ULOrder',
  'download_cursor',
  'SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant, o.notes, o.status
      FROM ULOrder o
      WHERE o.last_modified >= {ml s.last_table_download}
          AND o.emp_name = {ml s.username}' )
```

注意

- ローの値は、単一のテーブルまたは複数のテーブル間のジョインから選択できます。
- スクリプト自体にリモート・テーブルの名前を入れる必要はありません。リモート・テーブル名は、統合データベースのテーブル名と同じである必要はありません。リモート・テーブ

ル名は、Mobile Link システム・テーブル ml_table 内のエントリによって示されます。Sybase Central では、リモート・テーブルがそのスクリプトとともにリストされます。

- リモート・テーブルのローには、emp_id、emp_fname、emp_lname の値を入れてください。リモート・カラムは、名前が異なっていてもかまいませんが、上記の順にしてください。リモート・データベース内のカラムは、リファレンス・データベース内のカラムと同じ順になります。
- カーソル・スクリプトでは、リモート・データベースで定義されている順序に従ってカラムを選択してください。統合データベースでカラム名やテーブル構造が異なる場合、リモート・データベース (リファレンス・データベースと同等) に対して正しい順序でカラムを選択してください。カラムは、SELECT 文内の順序に基づいてリモート・データベース内のカラムに割り当てられます。
- Ultra Light アプリケーションを構築する場合は、Ultra Light アプリケーションの各テーブルに対して Ultra Light ジェネレータがサンプル・ダウンロード・スクリプトを作成します。そして、これらのサンプル・スクリプトをリファレンス・データベースに挿入します。サンプル・スクリプトでは、統合データベースにアプリケーションと同じテーブルが含まれていることを仮定しています。統合データベースの設計が異なる場合は、サンプル・スクリプトを修正してください。その場合でも、これらのスクリプトを基にすることができます。
- download_cursor スクリプトには、すべてのカラムをリモート・データベースで定義されたときと同じ順序で含めてください。

参照

- 「download_cursor テーブル・イベント」 418 ページ
- 「リモート・データベース間でのローの分割」 139 ページ
- 「download_delete_cursor スクリプトの作成」 353 ページ

download_delete_cursor スクリプトの作成

リモート・データベースからローを削除するには、download_delete_cursor スクリプトを作成します。このスクリプトは、同期中に削除するローを含む、リモート・データベースの各テーブルに対して 1 つ作成してください。

統合データベースからのローの削除のみを実行し、リモート・データベースからローを消去することはできません。削除されたローのプライマリ・キーを download_delete_cursor で選択できるようにするため、そのローのプライマリ・キーを追跡する必要があります。こうするには、以下の 2 つの一般的な方法があります。

- **論理削除**　統合データベースのローを物理的に削除しないでください。その代わり、ローが有効であるかどうかを追跡するステータス・カラムを使用します。このようにすると、download_delete_cursor を簡略化できます。ただし、ステータス・カラムを認識して使用できるように、download_cursor とその他のアプリケーションを修正しなければならない場合があります。削除の時刻を保持する最終変更カラムが存在し、各リモートの最終ダウンロード時刻の追跡を行っている場合は、すべてのリモートのダウンロード時刻が削除の時刻よりも早ければ、ローを物理的に削除できます。

- **シャドー・テーブル**　削除に関する追跡を実行する各テーブルには、テーブルのプライマリ・キーを保持するカラムとタイムスタンプを保持するカラムの 2 つを含むシャドー・テーブルを作成します。ローが削除されたときにプライマリ・キーとタイムスタンプをシャドー・テーブルに挿入するトリガを作成します。このようにすると、download_delete_cursor はこのシャドー・テーブルから選択できるようになります。論理削除と同様に、すべてのリモート・データベースによって対応するデータがダウンロードされれば、シャドー・テーブルからローを削除できます。

Mobile Link サーバは、統合データベースからプライマリ・キー値を選択し、それらをリモート・データベースに渡すことによって、リモート・データベースのローを削除します。値がリモート・データベース内のプライマリ・キーの値と一致する場合、そのローを削除します。

- 各 download_delete_cursor スクリプトには、SELECT 文か、結果セットを返すストアド・プロシージャの呼び出しが必要です。Mobile Link サーバは、SELECT 文を使用して統合データベース内でカーソルを定義します。
- SELECT 文では、リモート・データベース内のテーブルのプライマリ・キー・カラムに対応するすべてのカラムを選択します。統合データベース内のカラムは、対応するリモート・データベースのカラムとは異なる名前にできますが、互換性のある型にしてください。
- 値は、対応するカラムがリモート・データベース内で定義されている順序に従って選択します。この順序は、テーブルの作成に使用される CREATE TABLE 文のカラム順と同じですが、プライマリ・キーを定義する文中のカラム順とは異なります。
- 親レコードを削除すると、子レコードも自動的に削除されます。

  子レコードの削除の詳細については、「参照整合性と同期」 『Mobile Link - クイック・スタート』を参照してください。

各 download_delete_cursor スクリプトでは、対応するリモート・テーブルのプライマリ・キーに存在するすべてのカラム値を選択する必要がありますが、その他のすべてのカラムを選択することも可能です。この機能は、古いクライアントとの互換性を保つためだけにあります。追加のカラムを選択すると、データベース・サーバが取得しなければならないデータが増え、効率が悪くなります。古いクライアント以外については、Mobile Link サーバが追加の値を即座に破棄します。追加の値は、古いクライアントにのみダウンロードされます。

テーブルから全ローを削除

Mobile Link はすべての NULL を含むローの download_delete_cursor を検出すると、リモート・データベース内のデータをすべて削除します。download_delete_cursor 内の NULL の数は、プライマリ・キー・カラムの数またはテーブル内のカラムの総数です。

たとえば、次の download_delete_cursor SQL スクリプトは、2 つのプライマリ・キー・カラムがあるテーブル内のすべてのローを削除します。この例は、SQL Anywhere、Adaptive Server Enterprise、Microsoft SQL Server のデータベースに適用されます。

```
SELECT NULL, NULL
```

IBM DB2 LUW と Oracle の統合データベースでは、ダミー・テーブルを指定して NULL を選択してください。IBM DB2 LUW 7.1 では、次の構文を使用できます。

```
SELECT NULL, NULL FROM SYSIBM.SYSDUMMY1
```


Oracle 統合データベースでは、次の構文を使用できます。

```
SELECT NULL, NULL FROM DUAL
```

例

次の例は、従業員情報を格納しているリモート・テーブルに対する download_delete_cursor スクリプトです。Mobile Link サーバは、この SQL 文を使用して削除カーソルを定義します。このスクリプトは、スクリプトの実行時に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すべての従業員の情報を削除します。

```
SELECT emp_id
FROM employee
```

download_delete_cursor は、パラメータ last_download と ml_username を受け入れます。次のスクリプトは、各パラメータを使用して選択範囲を絞る方法を示しています。

```
SELECT order_id
FROM ULOrder
WHERE last_modified >= {ml s.last_table_download}
 AND status = 'Approved'
 AND user_name = {ml s.username}
```

注意
一部の統合データベースでは、適切なデータ型にキャストする必要があります。「CAST 関数 [データ型変換]」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

上記の例は、従業員数の多い組織では効率的ではない場合があります。リモート・データベースにあるローだけを選択すると、削除処理をより効率的に行えます。たとえば、最近新しくマネージャになった人だけを選択すると、ロー数を制限できます。また、クライアント・アプリケーションにロー自体を削除させるという方法もあります。この方法は、不必要なローをルールによって識別できる場合のみ可能です。たとえば、有効期限を示すタイムスタンプがローに含まれている場合があります。次回の同期時にこれらの削除がアップロードされるのを停止するには、STOP SYNCHRONIZATION DELETE 文を使用してからローを削除します。他の削除を通常の方法で同期する場合は、この後ですぐに START SYNCHRONIZATION DELETE を実行してください。

注意

- download_delete_cursor スクリプトには、プライマリ・キー・カラムをリモート・データベースで定義されたときと同じ順序で含めてください。

参照

効率良くローを削除するには、すべての Mobile Link クライアントに組み込まれている参照整合性検査を使用します。「参照整合性と同期」 『Mobile Link - クイック・スタート』を参照してください。

download_delete_cursor スクリプトの使用の詳細については、以下を参照してください。

- 「download_cursor テーブル・イベント」 418 ページ
- 「download_delete_cursor テーブル・イベント」 422 ページ
- 「削除の処理」 161 ページ
- 「削除同期の一時停止」 162 ページ
- 「STOP SYNCHRONIZATION DELETE 文 [Mobile Link]」『SQL Anywhere サーバ - SQL リファレンス』
- 「リモート・データベース間でのローの分割」 139 ページ
- 「スナップショットを使った同期」 137 ページ

エラーを処理するスクリプトの作成

Mobile Link サーバがスクリプト内のオペレーションを実行しているとき、そのオペレーションに失敗すると、同期スクリプトのエラーが発生します。DBMS は、エラーの内容を示す SQLCODE を Mobile Link サーバに返します。統合データベースの各 DBMS は、独自の SQLCODE 値を持っています。

エラーが発生すると、handle_error イベントが呼び出されます。エラーを処理するには、このイベントに対応する接続スクリプトを定義してください。このスクリプトには、Mobile Link サーバから、エラーの性質とコンテキストについての情報を提供するいくつかのパラメータが渡され、エラーへの対処方法を指示する出力値が要求されます。

エラー処理アクション

エラー処理スクリプト内で指定できるアクションには次のものがあります。

- 別のテーブルにエラーのログを取る
- Mobile Link サーバに、エラーを無視して続行、同期のロールバック、同期をロールバックして Mobile Link サーバを停止、のいずれかを指示する
- 電子メール・メッセージを送信する

詳細については、「handle_error 接続イベント」 469 ページを参照してください。

エラーのレポート

エラーによって同期処理の正常な進行が中断されるので、エラーとその解決方法の記録を継続的に作成することは困難です。report_error スクリプトは、このタスクを実行する手段を提供します。Mobile Link サーバはエラーが発生すると常に、このスクリプトを実行します。handle_error スクリプトが定義されている場合は、レポート・スクリプトに先立って即座にそれが実行されます。

report_error スクリプトのパラメータは、handle_error スクリプトのパラメータと同じです。report_error スクリプトではアクション・コードを変更できない点だけが異なります。アクション・コードの値は、handle_error スクリプトによって返されます。したがって、エラー処理問題のデバッグにこのスクリプトを使用できます。

このスクリプトは通常、後で参照できるように、時間や日付などといった他のデータとともに値をテーブルに記録する INSERT 文から成ります。このデータが失われないようにするために、Mobile Link サーバは常にこのスクリプトを別のトランザクションで実行し、スクリプトが完了すると変更を自動的にコミットします。

「report_error 接続イベント」 502 ページを参照してください。

例

次の report_error スクリプトは 1 つの INSERT 文で構成されています。このスクリプトは、現在の日付と時刻とともにスクリプトのパラメータをテーブルに追加します。スクリプトはこの変更をコミットしません。通常、Mobile Link サーバが自動的に行ってくれるからです。

```
INSERT INTO errors
VALUES(
  CURRENT DATE,
  {ml s.action_code},
  {ml s.error_code},
  {ml s.error_message},
  {ml s.username},
  {ml s.table} );
```

1 つの SQL 文で複数のエラーが処理される場合

ODBC では、1 つの SQL 文につき複数のエラーの処理が可能で、RDBMS にはこの機能を利用しているものがあります。たとえば、Microsoft SQL Server では、1 文で 2 つのエラーが発生することがあります。1 番目は実際のエラーで、通常 2 番目は現在の文が終了した理由を説明する情報メッセージです。

1 つの SQL 文で複数のエラーが発生した場合、1 エラーにつき 1 つの handle_error スクリプトが呼び出されます。Mobile Link サーバは、最も厳しいアクション・コード (つまり、一番大きな番号) を使用して、どのアクションを取るかを決定します。同じことが handle_error スクリプトにも適用されます。

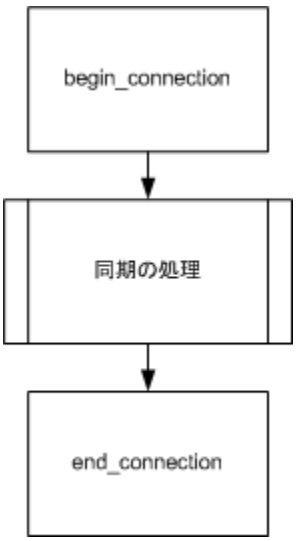
handle_error スクリプト自体で SQL エラーが発生した場合は、デフォルトのアクション・コード (3000) と見なされます。

同期イベント

目次

Mobile Link イベントの概要

同期要求が発生し、Mobile Link サーバが、新しい接続を作成することを決定すると、begin_connection イベントが呼び出され、同期が始まります。



同期に続いて、接続が接続プールに配置され、再度 Mobile Link は同期要求を待ちます。end_connection イベントが呼び出された後、接続は最終的に接続プールから削除されます。ただし、同じバージョンに対する同期要求を再び受信した場合、Mobile Link はその次に受けた同期要求を同じ接続で処理します。現在の同期に影響するイベントがいくつかあります。

トランザクション

各同期内で、次のトランザクションが発生する可能性があります。各トランザクションは省略可能です。

- 認証
- 同期の開始
- アップロード

  dbmlsync -tu オプションを使用して複数のアップロード・トランザクションを指定できます。
- ダウンロードの準備
- ダウンロード
- 同期の終了
- 非ブロッキング・ダウンロード確認

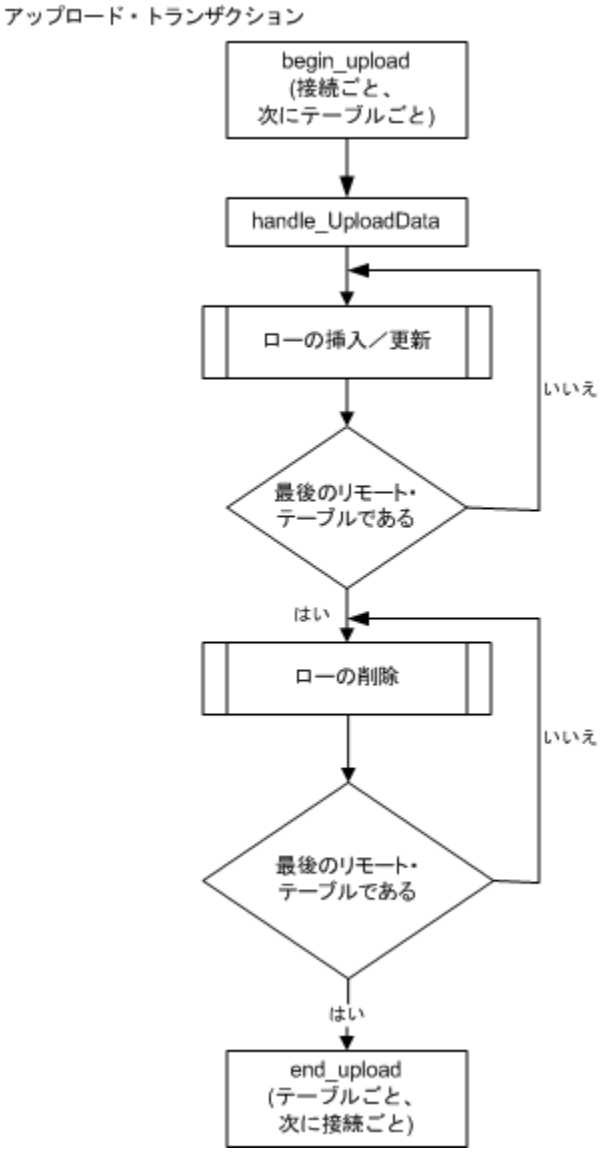
さらに、2 つの接続トランザクションが含まれることがあります。接続の開始トランザクションは、接続が確立された直後に発生し、接続の終了トランザクションは、接続が終了したときに発生します。

同期の主要段階は、アップロード・トランザクションとダウンロード・トランザクションです。アップロードとダウンロードのトランザクションに含まれるイベントについては、以下で概説します。

アップロード・トランザクション

アップロード・トランザクションは、リモート・データベースからアップロードされた変更を適用します。

begin_upload イベントは、アップロード・トランザクションの開始にマークを付けます。アップロード・トランザクション処理は 2 段階になっています。まず、すべてのリモート・テーブルに対する挿入と更新がアップロードされ、次にすべてのリモート・テーブルに対する削除がアップロードされます。



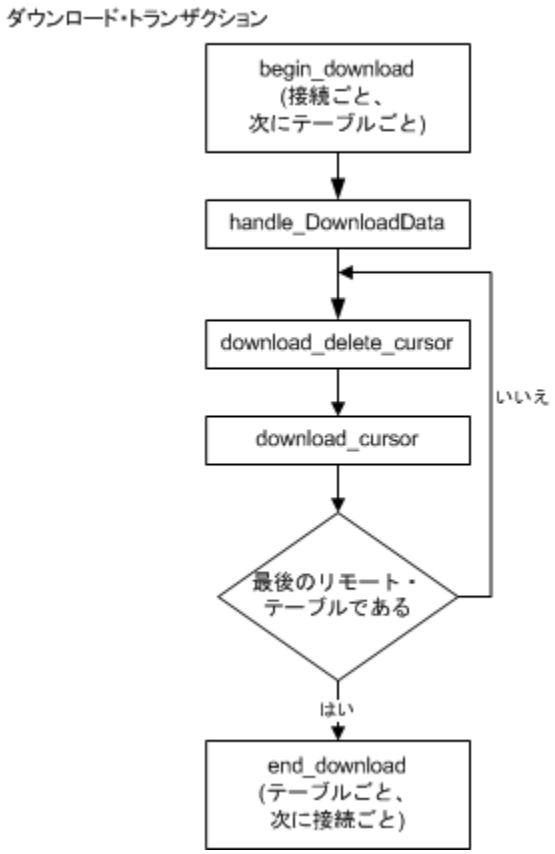
end_upload イベントは、アップロード・トランザクションの終了にマークを付けます。

「ローをアップロードするスクリプトの作成」 348 ページを参照してください。

ダウンロード・トランザクション

ダウンロード・トランザクションは、統合データベースからローをフェッチします。ダウンロード・トランザクションは、begin_download イベントで始まります。

ダウンロード・トランザクション処理は 2 段階になっています。まず最初に各テーブルに対する削除データがダウンロードされ、次に更新／挿入ロー (アップサート) がダウンロードされます。end_download イベントによって、ダウンロード・トランザクションが終了します。



「ローをダウンロードするスクリプトの作成」 351 ページを参照してください。

非ブロッキング・ダウンロード確認トランザクション

非ブロッキング・ダウンロード確認トランザクションは、Mobile Link が非ブロッキング・ダウンロード確認モードであり、かつダウンロード確認が受信された場合にのみ実行されます。このトランザクションには、2 つの目的があります。このトランザクションでは、publication_nonblocking_download_ack スクリプトと nonblocking_download_ack スクリプトが実行されます。これらのスクリプトは、ダウンロード・ステータスの追跡を容易にします。次に、このトランザクション中に Mobile Link システム・テーブルのダウンロード・タイムスタンプが更新されます。

このトランザクションは、ターゲット同期に対する他のイベントと同じデータベース接続では実行されません。つまり、このトランザクションでは接続レベルの変数が参照されません。

擬似コード内のイベントの概要

次の疑似コードは、イベント (イベントと同名のスクリプト) が呼び出される順序の概要を示します。この Mobile Link イベント・モデルの説明は、エラーのない完全な同期 (アップロード専用またはダウンロード専用ではない) を想定しています。

注意

- ほとんどの場合、指定したイベントのスクリプトを定義していないと、デフォルト・アクションは何も実行しません。
- begin_connection イベントと end_connection イベントは、「接続レベルのイベント」です。これらのイベントは特定の同期に依存せず、パラメータがありません。
- 各テーブルが同期されるたびに 1 回ずつ呼び出されるイベントもあります。これらのイベントに対応するスクリプトは、「テーブル・レベルのスクリプト」と呼ばれます。

  各テーブルは専用のテーブル・スクリプトを持つことができますが、いくつかのテーブルで共有されるテーブル・レベルのスクリプトを作成することもできます。
- begin_synchronization など、一部のイベントは接続レベルとテーブル・レベルの両方で発生します。これらのイベントに対しては、接続スクリプトとテーブル・スクリプトの両方を作成できます。
- 同期処理がどのように複数のトランザクションに分散されるかについては、COMMIT 文が参考になります。
- データベース・エラーは、同期処理中のどの時点でも発生する可能性があります。データベース・エラーは、handle_error スクリプトまたは handle_odbc_error スクリプトを使用して処理されます。

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。

Mobile Link の完全なイベント・モデル

```
-------------------------------------------------------
MobiLink complete event model.

Legend:
- // This is a comment.
- <name>
    The pseudo code for <name> is listed separately
    in a later section, under a banner:
        ------------------------
        name
        ------------------------
- VariableName <- value
    Assign the given value to the given variable name.
    Variable names are in mixed case.
- event_name
    If you have defined a script for the given event name,
    it is invoked.
-------------------------------------------------------

CONNECT to consolidated database
begin_connection_autocommit
begin_connection
```

```
COMMIT
for each synchronization request with
    the same script version {
  <synchronize>
}
end_connection
COMMIT
DISCONNECT from consolidated database

-------------------------------------------------------
synchronize
-------------------------------------------------------

<authenticate>
<begin_synchronization>
<upload>
<prepare_for_download>
<download>
<end_synchronization>

-------------------------------------------------------
authenticate
-------------------------------------------------------

Status <- 1000
UseDefaultAuthentication <- TRUE
if( authenticate_user script is defined ) {
  UseDefaultAuthentication <- FALSE
  TempStatus <- authenticate_user
  if( TempStatus > Status ) {
    Status <- TempStatus
  }
}
if( authenticate_user_hashed script is defined ) {
  UseDefaultAuthentication <- FALSE
  TempStatus <- authenticate_user_hashed
  if( TempStatus > Status ) {
    Status <- TempStatus
  }
}
  if( authenticate_parameters script is defined )
 {
    TempStatus <- authenticate_parameters
    if( TempStatus > Status ) {
      Status <- TempStatus
  }
if( UseDefaultAuthentication ) {
  if( the user exists in the ml_user table ) {
    if( ml_user.hashed_password column is not NULL ) {
      if( password matches ml_user.hashed_password ) {
        Status <- 1000
      } else {
        Status <- 4000
      }
    } else {
      Status <- 1000
    }
  } else if( -zu+ was on the command line ) {
    Status <- 1000
  } else {
    Status <- 4000
  }
}
if( Status >= 3000 ) {
```



```
  // Abort the synchronization.
} else {
  // UserName defaults to MobiLink user name
  // sent from the remote.
  if( modify_user script is defined ) {
    UserName <- modify_user
    // The new value of UserName is later passed to
    // all scripts that expect the MobiLink user name.
  }
}
COMMIT

-------------------------------------------------------
begin_synchronization
-------------------------------------------------------

begin_synchronization   // Connection event.
for each table being synchronized {
    begin_synchronization    // Call the table level script.
}
for each publication being synchronized {
  begin_publication
}
COMMIT

-------------------------------------------------------
end_synchronization
-------------------------------------------------------

for each publication being synchronized {
  if( begin_publication script was called ) {
    end_publication
  }
}
for each table being synchronized {
  if( begin_synchronization table script was called ) {
    end_synchronization // Table event.
  }
}
if( begin_synchronization connection script was called ) {
  end_synchronization     // Connection event.
}
for each table being synchronized {
synchronization_statistics // Table event.
}
synchronization_statistics // Connection event.
for each table being synchronized {
  time_statistics // Table event.
}
time_statistics // Connection event.

COMMIT
```

アップロード処理の詳細については、「アップロード中のイベント」 368 ページを参照してください。

ダウンロード処理の詳細については、「ダウンロード中のイベント」 370 ページを参照してください。

アップロード中のイベント

次の擬似コードは、アップロード・イベントとアップロード・スクリプトがどのように呼び出されるかを示します。

これらのイベントは、完全なイベント・モデルのアップロードのロケーションで発生します。「Mobile Link イベントの概要」 361 ページを参照してください。

アップロードの概要

```
-------------------------------------------------------
upload
-------------------------------------------------------

begin_upload // Connection event
for each table being synchronized {
  begin_upload // Table event
}
  handle_UploadData
  for each table being synchronized {
    begin_upload_rows
    for each uploaded INSERT or UPDATE for this table {
      if( INSERT ) {
        <upload_inserted_row>
      }
      if( UPDATE ) {
        <upload_updated_row>
      }
    }
    end_upload_rows
  }
  for each table being synchronized IN REVERSE ORDER {
    begin_upload_deletes
    for each uploaded DELETE for this table {
      <upload_deleted_row>
    }
    end_upload_deletes
  }

For each table being synchronized {
  if( begin_upload table script is called ) {
    end_upload // Table event
  }
}
if( begin_upload connection script was called ) {
  end_upload // Connection event

  for each table being synchronized {
    upload_statistics  // Table event.
  }
    upload_statistics  // Connection event.

  COMMIT
```

挿入のアップロード

```
-------------------------------------------------------
<upload_inserted_row>
-------------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.
```



```
  ConflictsAreExpected <- (
      upload_new_row_insert script is defined
    or upload_old_row_insert script is defined
    or resolve_conflict script is defined )
  if( upload_insert script is defined ) {
    upload_insert
  } else if( ConflictsAreExpected
      and upload_update script is not defined
      and upload_insert script is not defined
      and upload_delete script is not defined ) {
      // Forced conflict.
      upload_new_row_insert
      resolve_conflict
  } else {
      // Ignore the insert.
  }
```

更新のアップロード

```
-------------------------------------------------------
upload_updated_row
-------------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.
// - Both the old (original) and new rows are uploaded for
//   each update.

  ConflictsAreExpected <- (
      upload_new_row_insert script is defined
    or upload_old_row_insert script is defined
    or resolve_conflict script is defined )
  Conflicted <- FALSE
  if( upload_update script is defined ) {
    if( ConflictsAreExpected
      and upload_fetch script is defined ) {
      FETCH using upload_fetch INTO current_row
      if( current_row <> old row ) {
        Conflicted <- TRUE
      }
    }
    if( not Conflicted ) {
      upload_update
    }
  } else if( upload_update script is not defined
      and upload_insert script is not defined
      and upload_delete script is not defined ) {
      // Forced conflict.
      Conflicted <- TRUE
  }
  if( ConflictsAreExpected and Conflicted ) {
    upload_old_row_insert
    upload_new_row_insert
    resolve_conflict
  }
```

削除のアップロード

```
-------------------------------------------------------
upload_deleted_row
-------------------------------------------------------
// NOTES:
// - Only table scripts for the current table are involved.
```

```
ConflictsAreExpected <- (
    upload_new_row_insert script is defined
  or upload_old_row_insert script is defined
  or resolve_conflict script is defined )
if( upload_delete is defined ) {
  upload_delete
} else if( ConflictsAreExpected
  and upload_update script is not defined
  and upload_insert script is not defined
  and upload_delete script is not defined ) {
  // Forced conflict.
  upload_old_row_insert
  resolve_conflict
} else {
  // Ignore this delete.
}
```

ダウンロード中のイベント

次の疑似コードは、ダウンロード・イベント (イベントと同名のスクリプト) が呼び出される順序の概要を示します。

これらのイベントは、「Mobile Link イベントの概要」 361 ページに示す完全なイベント・モデルのダウンロードのロケーションで発生します。

```
-------------------------------------------------------
prepare_for_download
-------------------------------------------------------

modify_last_download_timestamp
fetch the next download timestamp from consolidated
prepare_for_download
if( modify_last_download_timestamp script is defined
    or prepare_for_download script is defined ) {
    COMMIT
}

-------------------------------------------------------
download
-------------------------------------------------------

begin_download // Connection event.
for each table being synchronized {
   begin_download // Table event.
}
   handle_DownloadData
   for each table being synchronized {
     begin_download_deletes
     for each row in download_delete_cursor {
       if( all primary key columns are NULL ) {
         send TRUNCATE to remote
       } else {
         send DELETE to remote
       }
     }
     end_download_deletes
     begin_download_rows
     for each row in download_cursor {
       send INSERT ON EXISTING UPDATE to remote
     }
```



```
    end_download_rows
  }
  modify_next_last_download_timestamp
  for each table being synchronized {
    if( begin_download table script is called ) {
      end_download // Table event
    }
}
if( begin_download connect script is called ) {
  end_download // Connection event
}
  for each table being synchronized {
    download_statistics   // Table event.
  }
    download_statistics   // Connection event.

COMMIT
```

注意

- ダウンロード通知を要求していてもダウンロードの確認をクライアントから受け取らなかった場合、統合データベースではダウンロード・トランザクション全体がロールバックされます。

  SQL Anywhere リモートについては、「SendDownloadACK (sa) 拡張オプション」『Mobile Link - クライアント管理』を参照してください。Ultra Light リモートについては、「Send Download Acknowledgement 同期パラメータ」『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。

- ダウンロード・ストリームでは、挿入と更新が区別されません。download_cursor イベントに対応するスクリプトは、ダウンロードされるローを定義する SELECT 文です。クライアントは、ローが存在するかどうかを調べ、適切な挿入操作または更新操作を実行します。

- ダウンロード処理の最後に、クライアントは自動的に参照整合性に違反するローを削除します。

  「参照整合性と同期」『Mobile Link - クイック・スタート』を参照してください。

authenticate_file_transfer 接続イベント

mlfiletransfer ユーティリティまたは MLFileTransfer メソッドを使用してファイル転送のカスタム認証を実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.file_authentication_code	INTEGER。必須。これは INOUT パラメータです。認証の全体の成功状況を示します。 この値が 1000 ～ 1999 の場合、ファイル転送は許可されます。この値が 2000 ～ 2999 の場合、ファイル転送は許可されません。	1
s.filename	VARCHAR(128)。認証の対象となる、転送されているファイルの名前を示す省略可能なパラメータです。パスを含めないでください。ファイルは、mlsrv11 -ftr オプションを使用して指定したルート転送ディレクトリまたは自動的に作成されるサブディレクトリに配置してください。	2
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	3

備考

Mobile Link サーバはこのイベントを実行してから、mlfiletransfer ユーティリティまたは MLFileTransfer メソッドを使用したファイル転送を許可します。ユーザが通常の認証を使用して認証した後、このイベントが実行されます。このスクリプトが定義されていない場合、ファイル転送は許可されます。

MLFileTransfer メソッドは Ultra Light クライアントだけで使用できます。

参照

- 「スクリプトの追加と削除」 345 ページ
- 「-ftr オプション」 73 ページ
- 「Mobile Link ファイル転送ユーティリティ (mlfiletransfer)」 『Mobile Link - クライアント管理』
- Ultra Light：「Mobile Link ファイル転送の使用」 『Ultra Light データベース管理とリファレンス』
- 「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』

authenticate_parameters 接続イベント

ユーザ ID とパスワード以外の認証に使用できる、リモートからの値を受信します。この値を使用して、各同期を任意にカスタマイズすることもできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.authentication_status	INTEGER。これは INOUT パラメータです。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
a.*N* (1 つ以上)	VARCHAR(128)。たとえば、a.1 a.2 のような名前付きパラメータを指定できます。	3...

パラメータの説明

- **authentication_status** authentication_status パラメータは必須です。認証の全体の成功状況を示します。次のいずれかの値に設定されます。

戻り値	authentication_status	説明
V <= 1999	1000	認証に成功しました。
1999 < V <= 2999	2000	認証に成功しましたが、パスワードの有効期限がもうすぐ切れます。
2999 < V <= 3999	3000	認証に失敗しました。パスワードの有効期限が切れています。
3999 < V <= 4999	4000	認証に失敗しました。

戻り値	authentication_status	説明
4999 < V <= 5999	5000	認証に失敗しました。ユーザがすでに同期中です。
5999 < V	4000	戻り値が 5999 より大きい場合、その値は 4000 (認証に失敗) として解釈されます。

- **username** このパラメータは、Mobile Link ユーザ名です。VARCHAR(128)。
- **remote_ID** Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。

  「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』を参照してください。
- **remote_parameters** リモート・パラメータの数が予期される数と一致しないと、エラーの原因となります。また、パラメータがクライアントから送信されたときにこのイベントのスクリプトがない場合にもエラーが発生します。

備考

SQL Anywhere のクライアントからも Ultra Light のクライアントからも、文字列またはパラメータを文字列の形式で送信できます。これにより、ユーザ ID とパスワード以外の認証も可能になります。また、パラメータの値に基づいて同期をカスタマイズでき、それも前同期フェーズで認証中にカスタマイズできます。

Mobile Link 同期サーバは、各同期の開始時にこのイベントを実行します。このイベントは、authenticate_user イベントと同じトランザクションで実行されます。

このイベントを使用して、組み込み Mobile Link 認証メカニズムを、カスタム・メカニズムに置き換えられます。使用している DBMS の認証メカニズムを使用したり、Mobile Link 組み込みメカニズムには存在しない機能を実装したりできます。

authenticate_user スクリプトまたは authenticate_user_hashed スクリプトが呼び出されてエラーを返すと、このイベントは呼び出されません。

authenticate_parameters イベント用の SQL スクリプトは、ストアド・プロシージャとして実装してください。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「認証パラメータ」 339 ページ
- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』
- 「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』
- 「カスタム・ユーザ認証」 『Mobile Link - クライアント管理』
- 「authenticate_user 接続イベント」 377 ページ
- 「authenticate_user_hashed 接続イベント」 382 ページ
- 「begin_synchronization 接続イベント」 402 ページ
- dbmlsync：「-ap オプション」 『Mobile Link - クライアント管理』
- Ultra Light：「Authentication Parameters 同期パラメータ」 『Ultra Light データベース管理とリファレンス』と「Number of Authentication Parameters パラメータ」 『Ultra Light データベース管理とリファレンス』

例

Ultra Light リモート・データベースでは、ul_synch_info 構造体の num_auth_parms フィールドと auth_parms フィールドを使用して、パラメータを渡します。num_auth_parms は、パラメータの数で、0 ～ 255 の値になります。auth_parms は、文字列の配列へのポインタです。文字列がプレーン・テキストとして表示されるのを防ぐため、文字列はパスワードと同じ方法で送信されます。num_auth_parms が 0 の場合、auth_parms は NULL に設定します。次に、Ultra Light でパラメータを渡す例を示します。

```
ul_char * Params[ 3 ] = { UL_TEXT( "param1" ),
  UL_TEXT( "param2" ), UL_TEXT( "param3" ) };

...
info.num_auth_parms = 3;
info.auth_parms = Params;
```

SQL Anywhere のリモート・データベースでは、dbmlsync -ap オプションを使用して、パラメータをカンマで区切られたリストで渡します。たとえば、次のコマンド・ラインは 3 つのパラメータを渡します。

```
dbmlsync -ap "param1,param2,param3"
```

この例では、authenticate_parameters スクリプトは以下のようになります。

```
CALL my_auth_parm (
  {ml s.authentication_status},
  {ml s.remote_id},
  {ml s.username},
  {ml a.1},
  {ml a.2},
  {ml a.3}
)
```


authenticate_user 接続イベント

カスタム・ユーザ認証を実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.authentication_status	INTEGER。これは INOUT パラメータです。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.password	VARCHAR(128)。認証に使用するパスワード。ユーザがパスワードを入力しない場合、この値は NULL です。	3
s.new_password	VARCHAR(128)。ユーザ・パスワードをリセットするために使用している場合は、新しいパスワード。ユーザがパスワードを変更しない場合、この値は NULL です。	4

デフォルトのアクション

Mobile Link 組み込みユーザ認証メカニズムを使用します。

備考

Mobile Link 同期サーバは、各同期の開始時にこのイベントを実行します。このイベントは、begin_synchronization トランザクションの前にトランザクション内で実行されます。

このイベントを使用して、組み込み Mobile Link 認証メカニズムを、カスタム・メカニズムに置き換えられます。使用している DBMS の認証メカニズムを使用したり、Mobile Link 組み込みメカニズムには存在しない機能 (パスワードの有効期限やパスワードの最小長など) を実装したりできます。

authenticate_user イベントで使用するパラメータは、次のとおりです。

- **authentication_status** authentication_status パラメータは必須です。認証の全体の成功状況を示します。次のいずれかの値に設定されます。

戻り値	authentication_status	説明
V <= 1999	1000	認証に成功しました。
1999 < V <= 2999	2000	認証に成功しました。パスワードの有効期限がもうすぐ切れます。
2999 < V <= 3999	3000	認証に失敗しました。パスワードの有効期限が切れています。
3999 < V <= 4999	4000	認証に失敗しました。
4999 < V <= 5999	5000	認証に失敗しました。ユーザがすでに同期中です。
5999 < V	4000	戻り値が 5999 より大きい場合、その値は 4000 として解釈されます。

- **username** Mobile Link ユーザ名を示す省略可能なパラメータです。

  「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』を参照してください。

- **remote_id** Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。

- **password** 認証に使用するパスワードを示す省略可能なパラメータです。ユーザがパスワードを入力しない場合、この値は NULL です。

- **new_password** 新しいパスワードを示す省略可能なパラメータです。ユーザがパスワードを変更しない場合、この値は NULL です。

authenticate_user イベント用の SQL スクリプトは、ストアド・プロシージャとして実装してください。

2 つの認証スクリプトを両方とも定義し、両方のスクリプトが異なる authentication_status コードを返す場合は、大きい方の値が使用されます。

authenticate_user スクリプトは、すべての認証スクリプトとともに、トランザクション内で実行されます。このトランザクションは、常にコミットを実行します。

LDAP サーバ、IMAP サーバ、POP3 サーバを使用する認証を簡素化するために、authenticate_user イベントに使用できる事前に定義されたスクリプトがあります。

「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。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』
- 「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』
- 「カスタム・ユーザ認証」 『Mobile Link - クライアント管理』
- 「外部サーバに対する認証」 『Mobile Link - クライアント管理』
- 「authenticate_user_hashed 接続イベント」 382 ページ
- 「authenticate_parameters 接続イベント」 374 ページ
- 「modify_user 接続イベント」 492 ページ
- 「begin_synchronization 接続イベント」 402 ページ

SQL の例

一般的な authenticate_user スクリプトは、ストアド・プロシージャの呼び出しです。呼び出しの中のパラメータ順は、上記の順序と同じでなければなりません。次の例では、ml_add_connection_script を使用して、my_auth というストアド・プロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script(
  'ver1', 'authenticate_user', 'call my_auth ( {ml s.username} )'
)
```

たとえば、次の SQL Anywhere ストアド・プロシージャは認証にユーザ名のみ使用し、パスワードのチェックは行いません。このプロシージャは、指定されたユーザ名が ULEmployee テーブルにある従業員 ID の 1 つであるかどうかだけを確認します。

```
CREATE PROCEDURE my_auth( in @user_name varchar(128) )
BEGIN
 IF EXISTS
 ( SELECT * FROM ulemployee
  WHERE emp_id = @user_name )
 THEN
  MESSAGE 'OK' type info to client;
  RETURN 1000;
 ELSE
  MESSAGE 'Not OK' type info to client;
  RETURN 4000;
 END IF
END
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、authenticateUser という Java メソッドを authenticate_user イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_java_connection_script(
  'ver1', 'authenticate_user',
  'ExamplePackage.ExampleClass.authenticateUser'
)
```

次に示すのは、サンプルの Java メソッド authenticateUser です。このメソッドは、ユーザのパスワードをチェックし、必要に応じてパスワードを変更する Java メソッドを呼び出します。

```
public String authenticateUser(
  ianywhere.ml.script.InOutInteger authStatus,
  String user,
  String pwd,
  String newPwd )
  throws java.sql.sqlException {
  // A real authenticate_user handler would
  // handle more authentication code states.
  _curUser = user;
  if( checkPwd( user, pwd ) ) {
    // Authentication successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( changePwd( user, pwd, newPwd ) ) {
        // Authentication OK and password change OK.
        // Use custom code.
        authStatus.setValue( 1001 );
      } else {
        // Authentication OK but password
        // change failed. Use custom code.
        java.lang.System.err.println( "user: "
         + user + " pwd change failed!" );
        authStatus.setValue( 1002 );
      }
    } else {
      authStatus.setValue( 1000 );
    }
  } else {
    // Authentication failed.
    authStatus.setValue( 4000 );
  }
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、AuthUser という .NET メソッドを authenticate_user 接続イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_dnet_connection_script(
  'ver1', 'authenticate_user',
  'TestScripts.Test.AuthUser'
)
```

次に示すのは、サンプルの .NET メソッド AuthUser です。このメソッドは、ユーザのパスワードをチェックし、必要に応じてパスワードを変更する .NET メソッドを呼び出します。

```
public string AuthUser(
  ref int authStatus,
  string user,
  string pwd,
  string newPwd ) {
  // A real authenticate_user handler would
  // handle more authentication code states.
  _curUser = user;
  if( CheckPwd( user, pwd ) ) {
    // Authentication successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( ChangePwd( user, pwd, newPwd ) ) {
        // Authentication OK and password change OK.
```



```
        // Use custom code.
        authStatus = 1001;
      } else {
        // Authentication OK but password
        // change failed. Use custom code.
        System.Console.WriteLine( "user: "
         + user + " pwd change failed!" );
        authStatus = 1002;
      }
    } else {
      authStatus = 1000 ;
    }
  } else {
    // Authentication failed.
    authStatus = 4000;
  }
  return ( null );
}
```

.NET において C# で作成された authenticate_user スクリプトの詳しい例については、「.NET 同期のサンプル」 634 ページを参照してください。

authenticate_user_hashed 接続イベント

カスタム・ユーザ認証メカニズムを実装します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.authentication_status	INTEGER。これは INOUT パラメータです。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.hashed_password	BINARY(20)。ユーザがパスワードを入力しない場合、この値は NULL です。	3
s.hashed_new_password	BINARY(20)。ユーザのパスワードを変更するためにこのイベントを使用していない場合、この値は NULL です。	4

デフォルトのアクション

Mobile Link 組み込みユーザ認証メカニズムを使用します。

備考

このイベントは authenticate_user と同じですが、パスワードの部分のみ異なります。パスワードは、ml_user.hashed_password カラムに格納されたものと同じように、ハッシュされた形式となります。パスワードをハッシュされた形式で渡すことにより、セキュリティが向上します。

一方向ハッシュが使用されます。一方向ハッシュはパスワードを使用し、それを各パスワードで (基本的に) ユニークなバイト・シーケンスに変換します。一方向ハッシュでは、統合データベースに実際のパスワードを保存せずにパスワードの認証を行えます。

このスクリプトは、ユーザの認証シーケンス中に複数回呼び出すことができます。

authenticate_user と authenticate_user_hashed を両方とも定義し、両方のスクリプトが異なる authentication_status コードを返す場合は、大きい方の値が使用されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』
- 「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』
- 「カスタム・ユーザ認証」 『Mobile Link - クライアント管理』
- 「authenticate_user 接続イベント」 377 ページ
- 「authenticate_parameters 接続イベント」 374 ページ

SQL の例

一般的な authenticate_user_hashed スクリプトは、ストアド・プロシージャの呼び出しです。呼び出しの中のパラメータ順は、上記の順序と同じでなければなりません。次の例では、ml_add_connection_script を呼び出して、my_auth というストアド・プロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script(
  'ver1', 'authenticate_user_hashed',
  'call my_auth (
     {ml s.authentication_status},
     {ml s.username},
     {ml s.hashed_password})'
)
```

次の SQL Anywhere のストアド・プロシージャは、認証にユーザ名とパスワードの両方を使用します。このプロシージャは、指定されたユーザ名が ULEmployee テーブルにある従業員 ID の 1 つであるかどうかだけを確認します。プロシージャは、Employee テーブルには hashed_pwd という名前の binary(20) のカラムがあることを前提としています。

```
CREATE PROCEDURE my_auth(
 inout @authentication_status integer,
 in @user_name varchar(128),
 in @hpwd binary(20) )
BEGIN
 IF EXISTS
 ( SELECT * FROM ulemployee
   WHERE emp_id = @user_name
     and hashed_pwd = @hpwd )
 THEN
   message 'OK' type info to client;
   RETURN 1000;
 ELSE
   message 'Not OK' type info to client;
   RETURN 4000;
 END IF
END
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、authUserHashed という Java メソッドを authenticate_user_hashed イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1', 'authenticate_user_hashed',
  'ExamplePackage.ExampleClass.authUserHashed')
```

次に示すのは、サンプルの Java メソッド authUserHashed です。このメソッドは、ユーザのパスワードをチェックし、必要に応じてパスワードを変更する Java メソッドを呼び出します。

```
public String authUserHashed(
  ianywhere.ml.script.InOutInteger authStatus,
  String user,
  byte pwd[],
  byte newPwd[] )
  throws java.sql.SQLException {
  // A real authenticate_user_hashed handler
  // would handle more auth code states.
  _curUser = user;
  if( checkPwdHashed( user, pwd ) ) {
    // Authorization successful.
    if( newPwd != null ) {
      // Password is being changed.
      if( changePwdHashed( user, pwd, newPwd ) ) {
        // Authorization OK and password change OK.
        // Use custom code.
        authStatus.setValue( 1001 );
      } else {
        // Auth OK but password change failed.
        // Use custom code
        java.lang.System.err.println( "user: " + user
          + " pwd change failed!" );
        authStatus.setValue( 1002 );
      }
      } else {
        authStatus.setValue( 1000 );
      }
    } else {
      // Authorization failed.
      authStatus.setValue( 4000 );
    }
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、AuthUserHashed という .NET メソッドを authenticate_user_hashed 接続イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_dnet_connection_script(
   'ver1',
   'authenticate_user_hashed',
   'TestScripts.Test.AuthUserHashed'
)
```

次に示すのは、サンプルの .NET メソッド AuthUserHashed です。

```
public string AuthUserHashed(
  ref int authStatus,
  string user,
  byte[] pwd,
  byte[] newPwd ) {
  // A real authenticate_user_hashed handler
  // would handle more auth code states.
  _curUser = user;
  if( CheckPwdHashed( user, pwd ) ) {
    // Authorization successful.
    if( newPwd != null ) {
```



```
      // Password is being changed.
      if( ChangePwdHashed( user, pwd, newPwd ) ) {
      // Authorization OK and password change OK.
      // Use custom code.
      authStatus = 1001;
    } else {
      // Auth OK but password change failed.
      // Use custom code
      System.Console.WriteLine( "user: " + user
        + " pwd change failed!" );
      authStatus = 1002;
    }
      } else {
        authStatus = 1000;
      }
    } else {
      // Authorization failed.
      authStatus = 4000;
    }
    return ( null );
}
```

begin_connection 接続イベント

Mobile Link サーバが統合データベース・サーバに接続するときに呼び出されます。

パラメータ

なし

デフォルトのアクション

なし

備考

Mobile Link 同期は、同期要求を受け取ると、要求に応じて接続を開きます。アプリケーションが Mobile Link サーバへの接続を形成または再形成すると、Mobile Link サーバはその同期の間、データベース・サーバへの接続を一時的に 1 つ割り付けます。Mobile Link サーバがプールからの接続を使用している場合、このイベントは呼び出されない可能性があります。

注意
このスクリプトは、通常は Java または .NET では使用されません。これは、データベース変数の代わりにこのクラス・インスタンスのメンバ変数を使用し、メンバをコンストラクタで準備するためです。

参照

- 「スクリプトの追加と削除」 345 ページ
- 「end_connection 接続イベント」 431 ページ
- 「-cn オプション」 58 ページ
- 「-w オプション」 108 ページ

SQL の例

次の SQL スクリプトは、SQL Anywhere 統合データベースで動作します。これで 2 つの変数が作成されます。1 つは last_download タイムスタンプ、もう 1 つは従業員 ID の変数です。

```
CALL ml_add_connection_script(
   'custdb',
   'begin_connection',
   'create variable @LastDownload timestamp;
   create variable @EmployeeID integer;')
```


begin_connection_autocommit 接続イベント

オートコミットをオンにします。

パラメータ

なし

デフォルトのアクション

オートコミットはオフです。

備考

Mobile Link サーバが統合データベースに接続するときにエラーが発生した場合、アップロードとダウンロードをロールバックできるように、オートコミットをオフにします。

しかし、Adaptive Server Enterprise の統合データベースを使用している場合、オートコミットがオンでなければ、テンポラリ・テーブルを作成するなどの DDL 機能は実行できません。Adaptive Server Enterprise の統合データベースを使用している場合、begin_connection_autocommit イベントで DDL コマンドを実行します。イベントが終了すると、オートコミットがオフになります。

begin_connection_autocommit スクリプトは、繰り返し可能なように作成します。エラーまたはデッドロックが発生した場合、Mobile Link サーバがスクリプトをリトライする必要があるからです (スクリプトのロールバックはできません)。

このイベントは、イベントに対するスクリプトが定義されている場合のみ実行されます。

参照

- 「スクリプトの追加と削除」 345 ページ

begin_download 接続イベント

Mobile Link サーバがダウンロードの準備を開始する直前に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_download	TIMESTAMP。同期テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2

デフォルトのアクション

なし

備考

Mobile Link サーバは、ダウンロード情報を処理する最初の手順としてこのイベントを実行します。ダウンロード情報は 1 つのトランザクションで処理されます。このイベントは、このトランザクションで最初に実行されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_download 接続イベント」 433 ページ
- 「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」 134 ページ

SQL の例

次の例では、ml_add_connection_script を呼び出して、SetDownloadParameters というストアド・プロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script (
  'Lab',
```



```
  'begin_download',
  'CALL SetDownloadParameters( {ml s.last_table_download}, {ml s.username} )' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginDownloadConnection という Java メソッドを begin_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'example_ver',
  'begin_download',
  'ExamplePackage.ExampleClass.beginDownloadConnection' )
```

次に示すのは、サンプルの Java メソッド beginDownloadConnection です。このメソッドは、以前に設定された JDBC 同期を使用して削除テーブルを準備する Java メソッド (prepDeleteTables) を呼び出します。

```
public String beginDownloadConnection(
  Timestamp ts,
  String user )
  throws java.sql.SQLException {
  prepDeleteTables ( _syncConn, ts, user );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginDownload という .NET メソッドを begin_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'begin_download',
  'TestScripts.Test.BeginDownload'
)
```

次に示すのは、サンプルの .NET メソッド BeginDownload です。このメソッドは、以前に設定された JDBC 同期を使用して削除テーブルを準備する .NET メソッド (prepDeleteTables) を呼び出します。

```
public string BeginDownload(
  DateTime timestamp,
  string user ) {
  prepDeleteTables ( _syncConn, ts, user );
  return ( null );
}
```

begin_download テーブル・イベント

挿入、更新、削除のダウンロードを準備する直前に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3

デフォルトのアクション

なし

備考

Mobile Link サーバは、特定のテーブルのダウンロード情報を準備する最初の手順として、このイベントを実行します。ダウンロード情報は、専用トランザクションで準備されます。このイベントの実行が、このトランザクションで最初のテーブル固有のアクションとなります。

リモート・データベースのテーブルごとに、begin_download スクリプトを 1 つ指定できます。このスクリプトは、テーブルが同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_download テーブル・イベント」 436 ページ
- 「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」 134 ページ

SQL の例

次の Mobile Link システム・プロシージャ procedure ml_add_table_script の呼び出しは BeginTableDownload プロシージャを呼び出します。これは SQL Anywhere 11 統合データベース用の構文です。

```
CALL ml_add_table_script(
  'version1',
  'Leads',
  'begin_download',
  'CALL BeginTableDownLoad(
    {ml s.last_table_download},
    {ml s.username},
    {ml s.table} )' );
```

次の SQL 文は BeginTableDownload プロシージャを作成します。

```
CREATE PROCEDURE BeginTableDownload(
  LastDownload timestamp,
  MLUser varchar(128),
  TableName varchar(128) )
BEGIN
  EXECUTE IMMEDIATE 'update ' || TableName ||
' set last_download_check = CURRENT TIMESTAMP
  WHERE Owner = ' ||MLUser;
END
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginDownloadTable という Java メソッドを begin_download テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_download',
  'ExamplePackage.ExampleClass.beginDownloadTable' )
```

次に示すのは、サンプルの Java メソッド beginDownloadTable です。このメソッドは、後のメソッド呼び出しで使用するために現在のテーブルの名前を保存します。

```
public String beginDownloadTable(
  Timestamp ts,
  String user,
  String table ) {
  _curTable = table;
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginTableDownload という .NET メソッドを begin_download テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1', 'table1', 'begin_download',
  'TestScripts.Test.BeginTableDownload'
)
```

次に示すのは、サンプルの .NET メソッド BeginTableDownload です。このメソッドは、後のメソッド呼び出しで使用するために現在のテーブルの名前を保存します。

```
public string BeginTableDownload(
  DateTime timestamp,
  string user,
  string table ) {
  _curTable = table;
  return ( null );
}
```

begin_download_deletes テーブル・イベント

リモート・データベース内の指定したテーブルから削除するローのリストをフェッチする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3

デフォルトのアクション

なし

備考

このイベントは、リモート・データベースの指定したテーブルから削除するローのリストをフェッチする直前に実行されます。

リモート・データベースのテーブルごとに、begin_download_deletes スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_download_rows テーブル・イベント」 396 ページ
- 「end_download_rows テーブル・イベント」 442 ページ
- 「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」 134 ページ

SQL の例

リモート・データベース上のデータ量を最小限に抑えるために、このイベントを使用して、download_delete_cursor の実行時に削除するデータにフラグを設定できます。次の例では、リモート・デバイスの 10 週以上経過した売上データに、削除対象を示すフラグを設定します。次の例は、SQL Anywhere 11 データベースで使用できます。

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginDownloadDeletes ストアド・プロシージャを begin_download_deletes イベントに割り当てます。

```
CALL ml_add_table_script (
 'ver1',
 'Leads',
 'begin_download_deletes',
 'CALL BeginDownloadDeletes (
   {ml s.last_table_download},
   {ml s.username},
   {ml s.table})' );
```

次の SQL 文は BeginDownloadDeletes ストアド・プロシージャを作成します。

```
CREATE PROCEDURE BeginDownloadDeletes(
  LastDownload timestamp,
  MLUser varchar(128),
  TableName varchar(128) )
BEGIN
  execute immediate 'update ' || TableName ||
  ' set delete_flag = 1 where
  days(creation_time, CURRENT DATE) > 70 and Owner = '
  || MLUser;
END;
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginDownloadDeletes という Java メソッドを begin_download_deletes テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'begin_download_deletes',
   'ExamplePackage.ExampleClass.beginDownloadDeletes' )
```

サンプルの Java メソッド beginDownloadDeletes は、後のメソッド呼び出しで使用するために現在のテーブルの名前を保存します。

```
public String beginDownloadDeletes (
  Timestamp ts,
  String user,
  String table ) {
  _curTable = table;
  return ( null );
}
```


.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginDownloadDeletes という .NET メソッドを begin_download_deletes テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script (
  'ver1', 'table1', 'begin_download_deletes',
  'TestScripts.Test.BeginDownloadDeletes'
)
```

サンプルの .NET メソッド BeginDownloadDeletes は、後のメソッド呼び出しで使用するために現在のテーブルの名前を保存します。

```
public string BeginDownloadDeletes(
  DateTime timestamp,
  string user,
  string table ) {
  _curTable = table;
  return ( null );
}
```

begin_download_rows テーブル・イベント

リモート・データベース内の指定したテーブルで挿入または更新するローのリストをフェッチする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3

デフォルトのアクション

なし

備考

このイベントは、リモート・データベース内の指定したテーブルで挿入または更新されるローのストリームをフェッチする直前に、実行されます。

リモート・データベースのテーブルごとに、begin_download_rows スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_download_deletes テーブル・イベント」 393 ページ
- 「end_download_deletes テーブル・イベント」 439 ページ
- 「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」 134 ページ

SQL の例

begin_download_rows テーブル・イベントを使用すると、このテーブルでダウンロードする必要がなくなったローにフラグを設定できます。次の例では、7 日以上経過した売上データをアーカイブします。

次の Mobile Link システム・プロシージャ・コールは、begin_download_rows イベント用の BeginDownloadRows ストアド・プロシージャを登録します。

```
CALL ml_add_table_script(
  'version1',
  'Leads',
  'begin_download_rows',
  'CALL BeginDownloadRows (
    {ml s.last_table_download},
    {ml s.username},
    {ml s.table})' ); )
```

次の SQL 文は BeginDowloadRows ストアド・プロシージャを作成します。

```
CREATE PROCEDURE BeginDownloadRows (
  LastDownload timestamp, MLUser varchar(128),
  TableName varchar(128) )
BEGIN
  execute immediate 'update ' || TableName ||
  ' set download_flag = 0 where
  days(creation_time, CURRENT DATE) > 7 and Owner = '
  || MLUser;
END;
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginDownloadRows という Java メソッドを begin_download_rows テーブル・イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'begin_download_rows',
   'ExamplePackage.ExampleClass.beginDownloadRows' )
```

次に示すのは、サンプルの Java メソッド beginDownloadRows です。このメソッドは、Mobile Link が実行するテーブルとユーザを使用して UPDATE 文を生成します。

```
public String beginDownloadRows(
  Timestamp ts,
  String user,
  String table ) {
  return( "update " + table + " set download_flag = 0 "
   + " where days(creation_time, CURRENT DATE) > 7 " +
   " and Owner = '" + user + "'" );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginDownloadRows という .NET メソッドを begin_download_rows テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1', 'table1', 'begin_download_rows',
  'TestScripts.Test.BeginDownloadRows'
)
```

次に示すのは、サンプルの .NET メソッド BeginDownloadRows です。このメソッドは、Mobile Link が実行するテーブルとユーザを使用して UPDATE 文を生成します。

```
public string BeginDownloadRows(
  DateTime timestamp,
  string user,
  string table ) {
  return( "update " + table + " set download_flag = 0 "
   + " where days(creation_time, CURRENT DATE) > 7 " +
   " and Owner = '" + user + "'" );
}
```


begin_publication 接続イベント

同期しているパブリケーションに関する有用な情報を提供します。このスクリプトを使って、ファイル・ベースのダウンロードの世代番号を管理できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトのパラメータ名	説明	順序
s.generation_number	INTEGER。これは INOUT パラメータです。使用している配備環境でファイル・ベースのダウンロードを使用しない場合、このパラメータは無視できます。デフォルトは 1 です。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.publication_name	VARCHAR(128)。パブリケーションの名前。	3
s.last_publication_upload	TIMESTAMP。このパブリケーションが最後に正常にアップロードされた時刻。	4
s.last_publication_download	TIMESTAMP。パブリケーションの最後のダウンロード時刻。	5
s.subscription_id	VARCHAR(128)。サブスクリプション ID。	6

デフォルトのアクション

デフォルトの世代番号は 1 です。このイベントにスクリプトが定義されていない場合、リモートに送信される世代番号は必ず 1 になります。

備考

このイベントを使って、現在同期されているパブリケーションに基づいて、同期論理を設計できます。このイベントは、begin_synchronization イベントと同じトランザクションで呼び出され、begin_synchronization イベントの後で呼び出されます。このイベントは、パブリケーションが同期するたびに 1 回呼び出されます。

このイベントは、使用されるパブリケーションに基づいてダウンロードされるものに影響を与えることができます。たとえば、優先度パブリケーション (PriorityPub) と全テーブル用のパブリケーション (AllTablesPub) の双方の一部であるテーブルを考えます。begin_publication イベント

のスクリプトは、Java クラスまたは SQL 変数やパッケージにパブリケーション名を保存できます。ダウンロード・スクリプトは、パブリケーションが PriorityPub と同期するか AllTablesPub と同期するかにより、異なった動作ができます。

Ultra Light リモートが UL_SYNC_ALL を使用して同期されている場合、このイベントは 'unknown' という名前で 1 回呼び出されます。

世代番号

generation_number パラメータは、特にファイル・ベースのダウンロード用です。世代番号の出力値は、begin_synchronization スクリプトから end_synchronization スクリプトへ渡されます。generation_number の意味は、現在の同期がダウンロード・ファイルを作成するために使用されているか、現在の同期にアップロードが含まれているかによって異なります。

ファイルベースのダウンロードでは、世代番号を使って、ダウンロードの前にアップロードを強制的に行います。世代番号は、ダウンロード・ファイルに保存されます。アップロードを持つ同期中に、パブリケーションに対するサブスクリプションごとに 1 つの世代番号が出力されます。この番号はアップロード確認でリモート・データベースへ送信され、SYSSYNC.generation_number に保存されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_publication 接続イベント」 445 ページ
- 「Mobile Link ファイルベースのダウンロード」 305 ページ
- 「Mobile Link の世代番号」 313 ページ
- 「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」 134 ページ

SQL の例

同期されるパブリケーションごとに情報を記録する必要がある場合があります。次の例では、ml_add_connection_script を呼び出して、RecordPubSync というストアド・プロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script(
 'version1',
 'begin_publication',
 '{CALL RecordPubSync(
    {ml s.generation_number},
    {ml s.username},
    {ml s.publication_name},
    {ml s.last_publication_upload},
    {ml s.last_publication_download},
    {ml s.subscription_id} )}' );
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginPublication という Java メソッドを begin_publication 接続イベント用のスクリプトとして登録します。


```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_publication',
 'ExamplePackage.ExampleClass.beginPublication' )
```

次に示すのは、サンプルの Java メソッド beginPublication です。このメソッドは後で使用する各パブリケーションの名前を保存します。

```
public String beginPublication(
  ianywhere.ml.script.InOutInteger generation_number,
  String user,
  String pub_name,
  Timestamp last_publication_upload,
  Timestamp last_download ) {
  _publicationNames[ _numPublications++ ] = pub_name;
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginPub という .NET メソッドを begin_publication 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'begin_publication',
 'TestScripts.Test.BeginPub'
)
```

次に示すのは、サンプルの .NET メソッド BeginPub です。このメソッドは後で使用する各パブリケーションの名前を保存します。

```
public string BeginPub(
  ref int generation_number,
  string user,
  string pub_name,
  DateTime last_publication_upload,
  DateTime last_download ) {
  _publicationNames[ _numPublications++ ] = pub_name;
  return ( null );
}
```

begin_synchronization 接続イベント

同期処理の準備中にアプリケーションが Mobile Link サーバに接続する時点で、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1

デフォルトのアクション

なし

備考

同期を準備するアプリケーションが Mobile Link サーバとの接続を形成した直後に、Mobile Link サーバがこのイベントを実行します。このイベントは、アップロード・トランザクションの前に、別のトランザクションで実行されます。

begin_synchronization スクリプトは統計値の管理に便利です。これは、エラーや競合が発生しても end_synchronization スクリプトが起動されるので、アップロード・トランザクションがロールバックされている間は、統計値のようにデータが維持されるためです。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_synchronization 接続イベント」 448 ページ
- 「begin_synchronization テーブル・イベント」 404 ページ
- 「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』

SQL の例

username の値を後続のスクリプトで何度も参照する場合は、その値をテンポラリ・テーブルまたは変数に格納できます。

```
CALL ml_add_connection_script (
 'version1',
```



```
  'begin_synchronization',
  'set @EmployeeID = {ml s.username}' );
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginSynchronizationConnection という Java メソッドを begin_synchronization 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_synchronization',
 'ExamplePackage.ExampleClass.beginSynchronizationConnection'
)
```

次に示すのは、サンプルの Java メソッド beginSynchronizationConnection です。このメソッドは後で使用する同期ユーザの名前を保存します。

```
public String beginSynchronizationConnection(
  String user ) {
  _curUser = user;
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginSync という .NET メソッドを begin_synchronization 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script( 'ver1',
  'begin_synchronization',
  'TestScripts.Test.BeginSync'
)
```

次に示すのは、サンプルの .NET メソッド BeginSync です。このメソッドは後で使用する同期ユーザの名前を保存します。

```
public string BeginSync(
  string user ) {
  _curUser = user;
  return ( null );
}
```

begin_synchronization テーブル・イベント

同期処理の準備中にアプリケーションが Mobile Link サーバに接続する時点で、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

デフォルトのアクション

なし

備考

同期を準備するアプリケーションが Mobile Link サーバとの接続を形成すると、Mobile Link サーバは、begin_synchronization 接続レベルのイベントを実行してから、このイベントを実行します。

リモート・データベースのテーブルごとに、begin_synchronization スクリプトを 1 つ指定できます。このイベントは、テーブルが同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_synchronization テーブル・イベント」 451 ページ
- 「begin_synchronization 接続イベント」 402 ページ
- 「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』

SQL の例

begin_synchronization テーブル・イベントを使って、特定のテーブルの同期を設定します。次の SQL スクリプトは、同期中にローを保存するためのテンポラリ・テーブルを作成するスクリプトを登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
 'begin_synchronization',
 'CREATE TABLE #sales_order (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
    order_date      date NOT NULL,
    fin_code_id char(2) NULL,
    region      char(7) NULL,
    sales_rep       integer NOT NULL,
    PRIMARY KEY (id),
)' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginSynchronizationTable という Java メソッドを begin_synchronization テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
 'ver1',
 'table1',
 'begin_synchronization',
 'ExamplePackage.ExampleClass.beginSynchronizationTable' )
```

次に示すのは、サンプルの Java メソッド beginSynchronizationTable です。このメソッドは、このインスタンスに含まれるテーブル名のリストに現在のテーブル名を追加します。

```
public String beginSynchronizationTable(
  String user,
  String table ) {
  _tableList.add( table );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginTableSync という .NET メソッドを begin_synchronization テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script (
  'ver1',
  'table1',
  'begin_synchronization',
  'TestScripts.Test.BeginTableSync' )
```

次に示すのは、サンプルの .NET メソッド BeginTableSync です。このメソッドは、このインスタンスに含まれるテーブル名のリストに現在のテーブル名を追加します。

```
public string BeginTableSync(
  string user,
  string table ) {
```

```
    _tableList.Add( table );
    return ( null );
  }
```


begin_upload 接続イベント

Mobile Link サーバがアップロードされた挿入、更新、削除のストリーム処理を開始する直前に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1

デフォルトのアクション

なし

備考

Mobile Link サーバは、アップロードした情報を処理する最初の手順としてこのイベントを実行します。アップロード情報は 1 つのトランザクションで処理されます。このイベントは、このトランザクションで最初に実行されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_upload 接続イベント」 454 ページ
- 「begin_upload テーブル・イベント」 409 ページ
- 「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』

SQL の例

begin_upload 接続イベントを使って、ローをアップロードする前に行う必要がある手順を実行できます。次の SQL スクリプトは、sales_order テーブルの矛盾処理のために新旧のローの値を保存するテンポラリ・テーブルを作成します。この例は SQL Anywhere 統合データベースで動作します。

```
CALL ml_add_connection_script(
 'version1',
 'begin_upload',
```

```
'CREATE TABLE #sales_order_conflicts (
  id          integer NOT NULL default autoincrement,
  cust_id     integer NOT NULL,
  order_date      date NOT NULL,
  fin_code_id char(2) NULL,
  region      char(7) NULL,
  sales_rep       integer NOT NULL,
  new_value       char(1) NOT NULL,
  PRIMARY KEY (id) )' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginUploadConnection という Java メソッドを begin_upload 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'begin_upload',
 'ExamplePackage.ExampleClass.beginUploadConnection ' )
```

次に示すのは、サンプルの Java メソッド beginUploadConnection です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String beginUploadConnection( String user ) {
  java.lang.System.out.println(
    "Starting upload for user: " + user );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、BeginUpload という .NET メソッドを begin_upload 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'begin_upload',
  'TestScripts.Test.BeginUpload'
)
```

次の C# サンプルは、後のイベントで使用するために現在のユーザ名を保存します。

```
public string BeginUpload( string curUser ) {
  user = curUser;
  return ( null );
}
```


begin_upload テーブル・イベント

Mobile Link サーバがアップロードされた挿入、更新、削除のストリーム処理を開始する直前に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

デフォルトのアクション

なし

備考

Mobile Link サーバは、アップロードした情報を処理する最初の手順としてこのイベントを実行します。アップロード情報は別のトランザクションで処理されます。このイベントの実行が、このトランザクションで最初のテーブル固有のアクションとなります。

リモート・データベースのテーブルごとに、begin_upload スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_upload テーブル・イベント」 456 ページ
- 「begin_upload 接続イベント」 407 ページ
- 「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』

SQL の例

リモートからローをアップロードする場合は、変更内容を中間テーブルに入れて手動で処理できます。このイベントでは、グローバルなテンポラリ・テーブルを設定できます。

```
CALL ml_add_table_script(
 'version1',
 'Leads',
 'begin_upload',
 'INSERT INTO T_Leads
  SELECT * FROM Leads
  WHERE Owner = @EmployeeID' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginUploadTable という Java メソッドを begin_upload テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload',
  'ExamplePackage.ExampleClass.beginUploadTable'
 )
```

次に示すのは、サンプルの Java メソッド beginUploadTable です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String beginUploadTable(
  String user,
  String table ) {
  java.lang.System.out.println("Beginning to process upload for: " + table);
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginTableUpload という .NET メソッドを begin_upload テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_upload',
  'TestScripts.Test.BeginTableUpload'
)
```

次に示すのは、サンプルの .NET メソッド BeginTableUpload です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string BeginTableUpload(
  string user,
  string table ) {
  System.Console.WriteLine("Beginning to process upload for: " + table);
```



```
  return ( null );
}
```

begin_upload_deletes テーブル・イベント

リモート・データベース内の指定のテーブルから削除されたローをアップロードする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

デフォルトのアクション

なし

備考

このイベントは、2 番目のパラメータで指定したクライアント・テーブルでローを削除した結果生じる変更を適用する直前に、発生します。

リモート・データベースのテーブルごとに、begin_upload_deletes スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_upload_deletes テーブル・イベント」 459 ページ
- 「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』

SQL の例

begin_upload_deletes 接続イベントは、特定のテーブルの挿入と更新をアップロードした後で、そのテーブルの削除をアップロードする前に行う必要のある手順を実行するために使用します。次の SQL スクリプトは、アップロード中に一時的に削除を保存するためのテンポラリ・テーブルを作成します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
 'begin_upload_deletes',
 'CREATE TABLE #sales_order_deletes (
   id          integer NOT NULL default autoincrement,
   cust_id     integer NOT NULL,
   order_date  date NOT NULL,
   fin_code_id char(2) NULL,
   region      char(7) NULL,
   sales_rep   integer NOT NULL,
   PRIMARY KEY (id) )' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginUploadDeletes という Java メソッドを begin_upload_deletes テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload_deletes',
  'ExamplePackage.ExampleClass.beginUploadDeletes' )
```

次に示すのは、サンプルの Java メソッド beginUploadDeletes です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String beginUploadDeletes(
  String user,
  String table )
  throws java.sql.SQLException {
  java.lang.System.out.println( "Starting upload
    deletes for table: " + table );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginUploadDeletes という .NET メソッドを begin_upload_deletes テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_upload_deletes',
  'TestScripts.Test.BeginUploadDeletes'
)
```

次に示すのは、サンプルの .NET メソッド BeginUploadDeletes です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string BeginUploadDeletes(
  string user,
  string table ) {
  System.Console.WriteLine(
   "Starting upload deletes for table: " + table );
  return ( null );
}
```


begin_upload_rows テーブル・イベント

リモート・データベース内の指定したテーブルから挿入と更新をアップロードする直前に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

デフォルトのアクション

なし

備考

このイベントは、2 番目のパラメータで指定したクライアント・テーブルに対する挿入と削除から生じる変更を適用する直前に、発生します。

リモート・データベースのテーブルごとに、begin_upload_rows スクリプトを 1 つ指定できます。このスクリプトは、テーブルが実際に同期されている場合にのみ呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_upload_rows テーブル・イベント」 462 ページ
- 「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』

SQL の例

begin_upload_rows 接続イベントを使って、特定のテーブルの挿入と更新をアップロードする前に行う必要がある手順を実行します。次のスクリプトは、統合データベースで Inventory テーブルへの挿入と更新を準備するストアド・プロシージャを呼び出します。

```
CALL ml_add_table_script(
 'MyCorp 1.0',
 'Inventory',
 'begin_upload_rows',
 'CALL PrepareForUpserts()' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、beginUploadRows という Java メソッドを begin_upload_rows テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload_rows',
  'ExamplePackage.ExampleClass.beginUploadRows' )
```

次に示すのは、サンプルの Java メソッド beginUploadRows です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String beginUploadRows(
  String user,
  String table )
  throws java.sql.SQLException {
  java.lang.System.out.println(
    "Starting upload rows for table: " +
    table + " and user: " + user );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、BeginUploadRows という .NET メソッドを begin_upload_rows テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'begin_upload_rows',
  'TestScripts.Test.BeginUploadRows'
)
```

次に示すのは、サンプルの .NET メソッド BeginUploadRows です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string BeginUploadRows(
  string user,
  string table ) {
  System.Console.WriteLine(
```



```
    "Starting upload rows for table: " +
    table + " and user: " + user );
   return ( null );
}
```

download_cursor テーブル・イベント

ダウンロードして、リモート・データベースで挿入または更新するローを選択するためのカーソルを定義します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2

デフォルトのアクション

なし

備考

Mobile Link サーバは読み込み専用のカーソルを開き、それを使用してリモート・データベースにダウンロードするローのリストをフェッチします。このスクリプトには、適切な SELECT 文を含めてください。

リモート・データベースのテーブルごとに、download_cursor スクリプトを 1 つ指定できます。

Ultra Light クライアントに対する同期のダウンロード処理のパフォーマンスを最適化するには、プライマリ・キー値の範囲がデバイスで指定されている現在のローの外側にある場合に、ダウンロード・カーソル内のローをプライマリ・キー順に並べてください。たとえば、大きいリファレンス・テーブルをダウンロードする場合は、このような最適化による利点が得られます。

各 download_cursor スクリプトには、SELECT 文か、SELECT 文を含むプロシージャの呼び出しが必要です。Mobile Link サーバは、SELECT 文を使用して統合データベース内でカーソルを定義します。

スクリプトでは、対応するリモート・データベース内のテーブルのカラムに対応するすべてのカラムを選択します。統合データベース内のカラムは、対応するリモート・データベースのカラムとは異なる名前にできますが、互換性のある型にしてください。

カラムは、対応するカラムがリモート・データベース内で定義されている順序に従って選択します。

download_cursor ではカスケード削除ができることに注意してください。そのため、データベースからレコードを削除できます。

不要なローのダウンロードを防ぐために、download_cursor スクリプトの WHERE 句に次の行を追加してください。

```
AND last_table_download > '1900/1/1'
```

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

ダウンロード・パフォーマンスに影響を与える大量の更新を行っているので、download_cursor スクリプトで READPAST テーブル・ヒントの使用を検討している場合は、代わりにダウンロードのスナップショット・アイソレーションの使用を検討してください。READPAST テーブル・ヒントは、download_cursor スクリプトで使用すると問題を引き起こす可能性があります。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、READPAST ヒントによってローが失われ、1 つのローが一度もリモート・データベースにダウンロードされなくなる可能性があります。次に例を示します。

- 1 つのローが統合データベースに追加され、コミットされます。そのローの last_modified カラムは、昨日の日時になっています。
- 同じローが更新されますが、コミットはされません。
- last_download の値が先週の日時になっているリモート・データベースと同期されます。
- download_cursor スクリプトは READPAST を使用してローを選択しようとして、そのローをスキップします。
- ローを更新したトランザクションはロールバックされます。リモートに対する次の最終ロード時間は今日に進められます。

この時点から、このローは更新されないかぎりダウンロードされません。回避方法として、modify_next_last_download_timestamp スクリプトを実装して、最終ダウンロード時刻を一番最初に開いたトランザクションの開始時間に設定する方法が考えられます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「ローをダウンロードするスクリプトの作成」 351 ページ
- 「download_cursor スクリプトの作成」 352 ページ
- 「リモート・データベース間でのローの分割」 139 ページ
- 「download_delete_cursor テーブル・イベント」 422 ページ
- 「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」 134 ページ
- 「FROM 句」 『SQL Anywhere サーバ - SQL リファレンス』の「Mobile Link 同期で READPAST を使用」

SQL の例

次の例は Oracle インストール環境の場合を示していますが、文はサポートされているすべてのデータベースに対して有効です。この例は、前回データをダウンロードした後に変更されたローのうち、emp_name カラム内のユーザ名と一致するローをすべてダウンロードします。

```
CALL ml_add_table_script(
  'Lab',
  'ULOrder',
  'download_cursor',
  'SELECT order_id,
    cust_id,
    prod_id,
    emp_id,
    disc,
    quant,
    notes,
    status
   FROM ULOrder
   WHERE last_modified >= {ml s.last_table_download}
    AND emp_name = {ml s.username}' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、downloadCursor という Java メソッドを download_cursor テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
   'ver1',
   'ULCustomer',
   'download_cursor',
   'ExamplePackage.ExampleClass.downloadCursor ' )
```

次に示すのは、サンプルの Java メソッド downloadCursor です。このメソッドは、last_modified カラムが最終ダウンロード時刻より遅い場合に、SQL 文をダウンロード・ローに返します。

```
public String downloadCursor(
  java.sql.Timestamp ts,
  String user ) {
  return( "SELECT cust_id, cust_name FROM ULCustomer
           WHERE last_modified >= ' "
   + ts + " ' ");
}
```


.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、DownloadCursor という .NET メソッドを download_cursor テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'download_cursor',
 'TestScripts.Test.DownloadCursor'
)
```

次に示すのは、サンプルの .NET メソッド DownloadCursor です。このメソッドは、テンポラリ・テーブルに *rows.txt* というファイルの内容を設定します。その結果、Mobile Link によって、テンポラリ・テーブルのローがリモート・データベースに送信されます。これは SQL Anywhere 統合データベース用の構文です。

```
public string DownloadCursor(
  DateTime ts,
  string user ) {
  DBCommand stmt   = curConn.CreateCommand();
  StreamReader input  = new StreamReader( "rows.txt" );
  string sql    = input.ReadLine();
  stmt.CommandText = "DELETE FROM dnet_dl_temp";
  stmt.ExecuteNonQuery();
  while( sql != null ){
    stmt.CommandText = "INSERT INTO dnet_dl_temp VALUES " + sql;
    stmt.ExecuteNonQuery();
    sql = input.ReadLine();
  }
  return( "SELECT * FROM dnet_dl_temp" );
}
```


download_delete_cursor テーブル・イベント

リモート・データベースで削除するローを選択するためのカーソルを定義します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2

デフォルトのアクション

なし

備考

Mobile Link サーバは読み込み専用のカーソルを開き、リモート・データベースにダウンロードして挿入または削除するローのリストを、そのカーソルを使用してフェッチします。このスクリプトには、リモート・データベース内のテーブルから削除されるローのプライマリ・キー値を返す SELECT 文を含めてください。

リモート・データベースのテーブルごとに、download_delete_cursor スクリプトを 1 つ指定できます。

テーブル内の 1 つ以上のローのプライマリ・キー・カラムで download_delete_cursor が NULL の場合、Mobile Link は、リモートにテーブル内のデータをすべて削除するように命令します。「テーブルから全ローを削除」 354 ページを参照してください。

統合データベースから削除されたローは、download_delete_cursor イベントにより定義された結果セットには表示されないため、リモート・データベースから自動的に削除されないことに注意してください。リモート・データベースから削除されるローを識別するには、ローを非アクティブとして識別するカラムを統合データベース・テーブルに追加する方法があります。

不要なローのダウンロードを防ぐために、download_delete_cursor スクリプトの WHERE 句に次の行を追加してください。

```
AND last_modified > '1900/1/1'
```

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

download_delete_cursor で READPAST テーブル・ヒントを使用すると、問題が発生する可能性があります。詳細については、download_cursor イベントを参照してください。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「download_cursor テーブル・イベント」 418 ページ
- 「ローをダウンロードするスクリプトの作成」 351 ページ
- 「リモート・データベース間でのローの分割」 139 ページ
- 「download_delete_cursor スクリプトの作成」 353 ページ
- 「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」 134 ページ
- 「FROM 句」 『SQL Anywhere サーバ - SQL リファレンス』の「Mobile Link 同期で READPAST を使用」

SQL の例

この例は Contact の例から抜粋したもので、*Samples¥MobiLink¥Contact¥build_consol.sql* にあります。この例は、このユーザが前回データをダウンロードした後に変更があった顧客 (Customer.last_modified >= {ml s.last_table_download}) と、次のいずれかに該当する顧客をリモート・データベースから削除します。

- 同期中のユーザに属していない顧客 (SalesRep.username != {ml s.username})
- 統合データベース内で非アクティブのマークが付いている顧客 (Customer.active = 0)

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'download_delete_cursor',
 'SELECT cust_id FROM Customer key join SalesRep
  WHERE Customer.last_modified >= {ml s.last_table_download} AND
   ( SalesRep.username != {ml s.username} OR Customer.active = 0 )')
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、downloadDeleteCursor という Java メソッドを download_delete_cursor イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'download_delete_cursor',
  'ExamplePackage.ExampleClass.downloadDeleteCursor' )
```

次に示すのは、サンプルの Java メソッド downloadDeleteCursor です。このメソッドは、ダウンロード削除カーソル用の SQL を生成する Java メソッドを呼び出します。

```
public String downloadDeleteCursor(
  Timestamp ts,
  String user ) {
  return( getDownloadCursor( _curUser, _curTable ) );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、DownloadDeleteCursor という .NET メソッドを download_delete_cursor テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'download_delete_cursor',
  'TestScripts.Test.DownloadDeleteCursor'
)
```

次に示すのは、サンプルの .NET メソッド DownloadDeleteCursor です。このメソッドは、ダウンロード削除カーソル用の SQL を生成する .NET メソッドを呼び出します。

```
public string DownloadDeleteCursor(
  DateTime timestamp,
  string user ) {
  return( GetDownloadCursor( _curUser, _curTable ) );
}
```


download_statistics 接続イベント

ダウンロード操作の同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。SYNCHRONIZATION USER 定義に指定した Mobile Link ユーザ名。	1
s.warnings	INTEGER。発行された警告の数。	2
s.errors	INTEGER。処理済みのエラーを含め、発生したエラーの数。	3
s.fetched_rows	INTEGER。download_cursor スクリプトによってフェッチされたローの数。	4
s.deleted_rows	INTEGER。download_delete_cursor スクリプトによってフェッチされたローの数。	5
s.filtered_rows	INTEGER。deleted_rows パラメータから実際にリモートに送信されたローの数。これには、アップロードされた値のダウンロード・フィルタが反映されます。	6
s.bytes	INTEGER。ダウンロードとしてリモートに送信されたバイト数。	7

デフォルトのアクション

なし

備考

download_statistics イベントを使用すると、任意のユーザについてダウンロード統計を収集できます。ダウンロード・トランザクション終了時のコミット直前に、download_statistics 接続スクリプトが呼び出されます。

注意
コマンド・ラインによっては、すべての警告やエラーのログが取られるとは限らないため、実際の警告数とエラー数が、ログを取られた警告数やエラー数より多くなる場合があります。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「download_statistics テーブル・イベント」 428 ページ
- 「upload_statistics 接続イベント」 538 ページ
- 「upload_statistics テーブル・イベント」 543 ページ
- 「synchronization_statistics 接続イベント」 511 ページ
- 「synchronization_statistics テーブル・イベント」 514 ページ
- 「time_statistics 接続イベント」 517 ページ
- 「time_statistics テーブル・イベント」 520 ページ
- 「Mobile Link モニタ」 183 ページ
- 「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』

SQL の例

次の例は、同期の統計を download_audit というテーブルに挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'download_statistics',
 'INSERT INTO download_audit(
   user_name,
   warnings,
   errors,
   deleted_rows,
   fetched_rows,
   download_rows,
   bytes )
  VALUES (
   {ml s.username},
   {ml s.warnings},
   {ml s.errors},
   {ml s.fetched_rows},
   {ml s.deleted_rows},
   {ml s.filtered_rows},
   {ml s.bytes})')
```

監査テーブルに重要な統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、downloadStatisticsConnection という Java メソッドを download_statistics イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'download_statistics',
  'ExamplePackage.ExampleClass.downloadStatisticsConnection' )
```

次に示すのは、サンプルの Java メソッド downloadStatisticsConnection です。このメソッドは、フェッチしたローの数を Mobile Link メッセージ・ログに出力します (フェッチしたローの数を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String downloadStatisticsConnection(
  String user,
  int warnings,
  int errors,
  int fetchedRows,
  int deletedRows,
  int bytes ) {
  java.lang.System.out.println(
   "download connection stats fetchedRows: "
   + fetchedRows );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、DownloadStats という .NET メソッドを download_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'download_statistics',
  'TestScripts.Test.DownloadStats'
)
```

次に示すのは、サンプルの .NET メソッド DownloadStats です。このメソッドは、フェッチしたローの数を Mobile Link メッセージ・ログに出力します (フェッチしたローの数を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string DownloadStats(
  string user,
  int warnings,
  int errors,
  int deletedRows,
  int fetchedRows,
  int downloadRows,
  int bytes ) {
  System.Console.WriteLine(
   "download connection stats fetchedRows: "
   + fetchedRows );
  return ( null );
}
```

download_statistics テーブル・イベント

ダウンロード操作の同期統計をテーブル別に追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。SYNCHRONIZATION USER 定義に指定した Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.warnings	INTEGER。発行された警告の数。	3
s.errors	INTEGER。処理済みのエラーを含め、発生したエラーの数。	4
s.fetched_rows	INTEGER。download_cursor スクリプトによってフェッチされたローの数。	5
s.deleted_rows	INTEGER。download_delete_cursor スクリプトによってフェッチされたローの数。	6
s.filtered_rows	INTEGER。(6) から実際にリモートに送信されたローの数。これには、アップロードされた値のダウンロード・フィルタが反映されます。	7
s.bytes	INTEGER。ダウンロードとしてリモートに送信されたバイト数。	8

デフォルトのアクション

なし

備考

download_statistics イベントを使用すると、任意のユーザとテーブルについて、そのテーブルに適用されるダウンロードの統計を収集できます。ダウンロード・トランザクション終了時のコミット直前に、download_statistics テーブル・スクリプトが呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「download_statistics 接続イベント」 425 ページ
- 「upload_statistics 接続イベント」 538 ページ
- 「upload_statistics テーブル・イベント」 543 ページ
- 「synchronization_statistics 接続イベント」 511 ページ
- 「synchronization_statistics テーブル・イベント」 514 ページ
- 「time_statistics 接続イベント」 517 ページ
- 「time_statistics テーブル・イベント」 520 ページ
- 「Mobile Link モニタ」 183 ページ
- 「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』

SQL の例

次の例は、同期の統計を download_audit というテーブルに挿入します。監査テーブルに重要な統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'download_statistics',
 'INSERT INTO download_audit (
   user_name,
   table, warnings,
   errors,
   deleted_rows,
   fetched_rows,
   download_rows,
   bytes)
  VALUES (
   {ml s.username},
   {ml s.table},
   {ml s.warnings},
   {ml s.errors},
   {ml s.fetched_rows},
   {ml s.deleted_rows},
   {ml s.filtered_rows},
   {ml s.bytes})')
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、downloadStatisticsTable という Java メソッドを download_statistics テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
```

```
  'table1',
  'download_statistics',
  'ExamplePackage.ExampleClass.downloadStatisticsTable' )
```

次に示すのは、サンプルの Java メソッド downloadStatisticsTable です。このメソッドは、このテーブルの統計を Mobile Link メッセージ・ログに出力します (テーブルの統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String downloadStatisticsTable(
  String user,
  String table,
  int warnings,
  int errors,
  int fetchedRows,
  int deletedRows,
  int bytes ) {
  java.lang.System.out.println( "download table stats "
   + "table: " + table + "bytes: " + bytes );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、DownloadTableStats という .NET メソッドを download_statistics テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'download_statistics',
 'TestScripts.Test.DownloadTableStats'
)
```

次に示すのは、サンプルの .NET メソッド DownloadTableStats です。このメソッドは、このテーブルの統計を Mobile Link メッセージ・ログに出力します (テーブルの統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string DownloadTableStats(
  string user,
  string table,
  int warnings,
  int errors,
  int deletedRows,
  int fetchedRows,
  int downloadRows,
  int bytes ) {
  System.Console.WriteLine( "download table stats "
   + "table: " + table + "bytes: " + bytes );
  return ( null );
}
```


end_connection 接続イベント

停止準備中、または接続プールから接続が削除されるとき、Mobile Link サーバが統合データベース・サーバとの接続を閉じる直前に、任意の文を処理します。

パラメータ

なし

デフォルトのアクション

なし

備考

Mobile Link サーバと統合データベース・サーバ間の接続を閉じる直前に、end_connection スクリプトを使用して、選択したアクションを実行できます。

このスクリプトは通常、begin_connection スクリプトによって起動されたすべてのアクションを完了し、取得されていたリソースをすべて解放するために使用されます。

参照

- 「begin_connection 接続イベント」 386 ページ
- 「スクリプトの追加と削除」 345 ページ

SQL の例

次の SQL スクリプトは、begin_connection スクリプトが作成したテンポラリ・テーブルを削除します。これは SQL Anywhere 統合データベース用の構文です。厳密に言うと、このテーブルは明示的に削除する必要はありません。接続が切断されるときに SQL Anywhere が自動的に削除します。テンポラリ・テーブルを明示的に削除する必要があるかどうかは、統合データベースのタイプによります。

```
CALL ml_add_connection_script(
 'version 1.0',
 'end_connection',
 'DROP TABLE #sync_info' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endConnection という Java メソッドを end_connection イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_connection',
  'ExamplePackage.ExampleClass.endConnection' )
```

次に示すのは、サンプルの Java メソッド endConnection です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String endConnection() {
  java.lang.System.out.println( "Ending connection." );
```

```
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndConnection という .NET メソッドを end_connection 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_connection',
  'TestScripts.Test.EndConnection'
)
```

次に示すのは、サンプルの .NET メソッド EndConnection です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string EndConnection() {
  System.Console.WriteLine( "Ending connection." );
  return ( null );
}
```


end_download 接続イベント

Mobile Link サーバがダウンロード・データの準備を完了した直後に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_download	TIMESTAMP。同期テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2

デフォルトのアクション

なし

備考

すべてのローのダウンロード後に、Mobile Link サーバはこのスクリプトを実行します。ブロッキング・ダウンロード確認を使用している場合、受信の確認の受信後に、スクリプトが実行されます。ダウンロード情報は 1 つのトランザクションで処理されます。このスクリプトの実行は、このトランザクションの最後の非統計アクションです。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_download 接続イベント」 388 ページ
- 「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」 134 ページ

SQL の例

次の例は、end_download 接続スクリプトの、考えられる用途の 1 つを示します。ULEmpCust テーブルには action カラムが含まれています。次のスクリプトは、このカラムの値を使用して、レコードをリモート・データベースから削除します。

```
CALL ml_add_connection_script(
 'ver1',
 'end_download',
 'DELETE FROM ULEmpCust ec
  WHERE ec.emp_id = {ml s.username} AND action = "D"')
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endDownloadConnection という Java メソッドを end_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_download',
  'ExamplePackage.ExampleClass.endDownloadConnection' )
```

次に示すのは、サンプルの Java メソッド endDownloadConnection です。ULEmpCust テーブルには action カラムが含まれています。次のスクリプトは、このカラムの値を使用して、レコードをリモート・データベースから削除します。また、ダウンロードが終了する前に、現在の Mobile Link 接続 (以前に保存した Mobile Link 接続) を使用して更新を実行します。これは SQL Anywhere 統合データベース用の SQL 構文です。

```
public String endDownloadConnection(
  Timestamp ts,
  String user )
  throws java.sql.SQLException {
  String del_sql = "DELETE FROM ULEmpCust ec " +
    "WHERE ec.emp_id = '" + user + "' " +
    "AND action = 'D' ";
  execUpdate( _syncConn, del_sql );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndDownload という .NET メソッドを end_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'end_download',
 'TestScripts.Test.EndDownload' )
```

次に示すのは、サンプルの .NET メソッド EndDownload です。ULEmpCust テーブルには action カラムが含まれています。次のスクリプトは、このカラムの値を使用して、レコードをリモート・データベースから削除します。また、ダウンロードが終了する前に、現在の Mobile Link 接続 (以前に保存した Mobile Link 接続) を使用して更新を実行します。これは SQL Anywhere 統合データベース用の SQL 構文です。


```
public string EndDownload(
  DateTime timestamp,
  string user ) {
  string del_sql = "DELETE FROM ULEmpCust ec " +
   "WHERE ec.emp_id = '" + user + "' " +
   "AND action = 'D' ";
   execUpdate( _syncConn, del_sql );
  return ( null );
}
```

end_download テーブル・イベント

Mobile Link サーバがダウンロードされた挿入、更新、削除のストリームの準備を終了した直後に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3

デフォルトのアクション

なし

備考

すべてのローのダウンロードと受信確認の受信後に、Mobile Link サーバはこのスクリプトを実行します。ダウンロード情報は、別のトランザクションで準備されます。このスクリプトの実行が、このトランザクションで最後のテーブル固有の非統計アクションとなります。

リモート・データベースのテーブルごとに、end_download スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_download テーブル・イベント」 390 ページ
- 「end_download 接続イベント」 433 ページ
- 「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」 134 ページ

SQL の例

end_download テーブル・イベントを使って、特定のテーブルをダウンロードした後で行う必要がある手順を実行します。次の SQL Anywhere の SQL スクリプトは、sales_summary テーブルのダウンロード・ローを保持するために prepare_for_download スクリプトが作成したテンポラリ・テーブルを削除します。

```
CALL ml_add_table_script(
 'MyCorp 1.0',
 'sales_summary',
 'end_download',
 'DROP TABLE #sales_summary_download' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endDownloadTable という Java メソッドを end_download テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script (
  'ver1',
  'table1',
  'end_download',
  'ExamplePackage.ExampleClass.endDownloadTable' )
```

次に示すのは、サンプルの Java メソッド endDownloadTable です。このメソッドは現在のテーブル・メンバ変数をリセットします。

```
public String endDownloadTable(
  Timestamp ts,
  String user,
  String table ) {
  _curTable = null;
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndTableDownload という .NET メソッドを end_download テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'end_download',
  'TestScripts.Test.EndTableDownload'
)
```

次に示すのは、サンプルの .NET メソッド EndTableDownload です。このメソッドは現在のテーブル・メンバ変数をリセットします。

```
public string EndTableDownload
  DateTime timestamp,
  string user,
  string table ) {
  _curTable = null;
  return ( null );
}
```


end_download_deletes テーブル・イベント

リモート・データベース内の指定されたテーブルから削除するローのリストを準備した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3

デフォルトのアクション

なし

備考

このスクリプトは、リモート・データベース内の指定されたテーブルから削除されるローのリストを準備した直後に実行されます。

リモート・データベースのテーブルごとに、end_download_deletes スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_download_deletes テーブル・イベント」 393 ページ
- 「end_download 接続イベント」 433 ページ
- 「begin_download_rows テーブル・イベント」 396 ページ
- 「end_download_rows テーブル・イベント」 442 ページ
- 「download_delete_cursor テーブル・イベント」 422 ページ
- 「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」 134 ページ

SQL の例

リモート・データベース上のローに削除マークを付ける必要がある場合があります。次のスクリプトは、OnRemote という統合データベースのカラムを更新します。

注意
UPDATE の WHERE 句は、download_delete_cursor イベントのスクリプトに使用される WHERE 句に一致します。

```
CALL ml_add_table_script(
 'version1',
 'Leads',
 'end_download_deletes',
 'UPDATE Leads SET OnRemote = 0
  WHERE LastModified >= {ml s.last_table_download}
   AND Owner = {ml s.username} AND DeleteFlag=1');
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endDownloadDeletes という Java メソッドを end_download_deletes テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_download_deletes',
  'ExamplePackage.ExampleClass.endDownloadDeletes' )
```

リモート・データベース上のローに削除マークを付ける必要がある場合があります。次に示すのは、サンプルの Java メソッド endDownloadDeletes です。このメソッドは、レコードがリモート・データベース上に存在しなくなったことを示すため、OnRemote という統合データベースのカラムを更新します。

注意
UPDATE の WHERE 句は、download_delete_cursor イベントのスクリプトに使用される WHERE 句に一致します。

```
public String endDownloadDeletes(
  Timestamp ts,
  String user,
```

```
  String table ) {
  return( "UPDATE Leads SET OnRemote = 0
     WHERE LastModified >= {ml s.last_table_download}
     AND Owner = {ml s.username} AND DeleteFlag=1" );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndDownloadDeletes という .NET メソッドを end_download_deletes テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'end_download_deletes',
  'TestScripts.Test.EndDownloadDeletes'
)
```

リモート・データベース上のローに削除マークを付ける必要がある場合があります。次に示すのは、サンプルの .NET メソッド EndDownloadDeletes です。このメソッドは、レコードがリモート・データベース上に存在しなくなったことを示すため、OnRemote という統合データベースのカラムを更新します。UPDATE の WHERE 句は、download_delete_cursor イベントのスクリプトに使用される WHERE 句に一致します。

```
public string EndDownloadDeletes(
  DateTime timestamp,
  string user,
  string table) {
  return( "UPDATE Leads SET OnRemote = 0
     WHERE LastModified >= {ml s.last_table_download}
     AND Owner = {ml s.username} AND DeleteFlag=1" );
}
```

end_download_rows テーブル・イベント

リモート・データベース内の指定されたテーブルで挿入または更新するローのリストを準備した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_table_download	TIMESTAMP。テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.table	VARCHAR(128)。テーブル名。	3

デフォルトのアクション

なし

備考

このスクリプトは、リモート・データベース内の指定されたテーブルで挿入または更新されるローのストリームを準備した直後に実行されます。

リモート・データベースのテーブルごとに、end_download_rows スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_download_rows テーブル・イベント」 396 ページ
- 「end_download 接続イベント」 433 ページ
- 「end_download_deletes テーブル・イベント」 439 ページ
- 「begin_download_deletes テーブル・イベント」 393 ページ
- 「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」 134 ページ

SQL の例

リモート・データベースへのダウンロードが成功したことを示すマークを、ローに付ける必要がある場合があります。次のスクリプトは、OnRemote という統合データベースのカラムを更新します。

注意
UPDATE の WHERE 句は、download_delete_cursor イベントのスクリプトに使用される WHERE 句に一致します。

```
CALL ml_add_table_script(
  'version1',
  'Leads',
  'end_download_rows',
  'UPDATE Leads SET OnRemote = 1
   WHERE LastModified >= {ml s.last_table_download}
    AND Owner = {ml s.username}
    AND DownloadFlag=1' );
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endDownloadRows という Java メソッドを end_download_rows テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_download_rows',
  'ExamplePackage.ExampleClass.endDownloadRows' )
```

次に示すのは、サンプルの Java メソッド endDownloadRows です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String endDownloadRows(
  Timestamp ts,
  String user,
  String table ) {
  java.lang.System.out.println(
   "Done downloading inserts and updates for table "
    + table );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndDownloadRows という .NET メソッドを end_download_rows テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'end_download_rows',
 'TestScripts.Test.EndDownloadRows'
)
```

次に示すのは、サンプルの .NET メソッド EndDownloadRows です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string EndDownloadRows(
  DateTime timestamp,
  string user,
  string table ) {
  System.Console.WriteLine(
   "Done downloading inserts and updates for table "
   + table );
  return ( null );
}
```


end_publication 接続イベント

同期しているパブリケーションに関する有用な情報を提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.generation_number	INTEGER。使用している配備環境でファイル・ベースのダウンロードを使用しない場合、このパラメータは無視できます。デフォルト値は 1 です。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.publication_name	VARCHAR(128)	3
s.last_publication_upload	TIMESTAMP。このパブリケーションが最後に正常にアップロードされた時刻。	4
s.last_publication_download	TIMESTAMP。パブリケーションの最後のダウンロード時刻。	5
s.subscription_id	VARCHAR(128)。サブスクリプション ID。	6

デフォルトのアクション

なし

備考

このイベントを使って、現在同期されているパブリケーションに基づいて、同期論理を設計できます。このイベントは、end_synchronization イベントと同じトランザクションで呼び出され、end_synchronization イベントの前に呼び出されます。このイベントは、パブリケーションが同期するたびに 1 回呼び出されます。

現在の同期がアップロードを正常に適用すると、last_upload パラメータにはこの最終アップロードが適用された時刻が含まれます。ブロッキング・ダウンロード確認を使用していて、現在の同期でダウンロード確認が成功すると、last_download 時刻にはこの最終ダウンロードが生成された時刻が含まれます。これは、最終ダウンロード時刻としてダウンロード・スクリプトに渡された値と同じです。

Ultra Light リモートが UL_SYNC_ALL を使用して同期されている場合、このイベントは 'unknown' という名前で 1 回呼び出されます。

世代番号

generation_number パラメータは、特にファイル・ベースのダウンロード用です。

世代番号の出力値は、begin_publication スクリプトから end_publication スクリプトへ渡されます。generation_number の意味は、現在の同期がダウンロード・ファイルを作成するために使用されているか、現在の同期にアップロードが含まれているかによって異なります。

ファイルベースのダウンロードでは、世代番号を使って、ダウンロードの前にアップロードを強制的に行います。世代番号は、ダウンロード・ファイルに保存されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_publication 接続イベント」 399 ページ
- 「Mobile Link ファイルベースのダウンロード」 305 ページ
- 「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」 134 ページ

SQL の例

同期されるパブリケーションごとに情報を記録する必要がある場合があります。次の例では、ml_add_connection_script を呼び出して、RecordPubEndSync というストアド・プロシージャにイベントを割り当てます。

```
CALL ml_add_connection_script(
  'version1',
  'end_publication',
  'CALL RecordPubEndSync(
   {ml s.generation_number},
   {ml s.username},
   {ml s.publication_name},
   {ml s.last_publication_upload},
   {ml s.last_publication_download} )' );
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endPublication という Java メソッドを end_publication 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_publication',
  'ExamplePackage.ExampleClass.endPublication' )
```

次に示すのは、サンプルの Java メソッド endPublication です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String endPublication(
  ianywhere.ml.script.InOutInteger generation_number,
  String user,
  String pub_name,
  Timestamp last_publication_upload,
  Timestamp last_publication_download ) {
  java.lang.System.out.println(
   "Finished synchronizing publication " + pub_name );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndPub という .NET メソッドを end_publication 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_publication',
  'TestScripts.Test.EndPub'
)
```

次に示すのは、サンプルの .NET メソッド EndPub です。このメソッドは、Mobile Link メッセージ・ログにメッセージを出力します (メッセージを Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string EndPub(
  ref int generation_number,
  string user,
  string pub_name,
  DateTime last_publication_upload,
  DateTime last_publication_download ) {
  System.Console.Write(
   "Finished synchronizing publication " + pub_name );
  return ( null );
}
```

end_synchronization 接続イベント

同期処理の完了時にアプリケーションを Mobile Link サーバから切断する時点で、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.synchronization_ok	INTEGER。この値は、同期が成功すると 1 に、失敗すると 0 になります。	2

デフォルトのアクション

なし

備考

同期の完了後、Mobile Link サーバはこのスクリプトを実行します。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 クライアントがダウンロードの受信の確認を返した後に、スクリプトが実行されます。

このスクリプトは、ダウンロード・トランザクションの後に、別のトランザクションで実行されます。

end_synchronization スクリプトは統計値の管理に便利です。これは、begin_synchronization スクリプトを呼び出した場合、エラーや競合が発生しても end_synchronization スクリプトが起動されるので、アップロード・トランザクションがロールバックされている間は、統計値が維持されるためです。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_synchronization 接続イベント」 402 ページ
- 「begin_synchronization テーブル・イベント」 404 ページ
- 「end_synchronization テーブル・イベント」 451 ページ
- 「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』

SQL の例

次の SQL スクリプトは、同期試行の終了時刻と同期の成功または失敗を記録するシステム・プロシージャを呼び出します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_connection_script(
 'ver1',
 'end_synchronization',
 'CALL RecordEndOfSyncAttempt(
   {ml s.username},
   {ml s.synchronization_ok} )' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endSynchronizationConnection という Java メソッドを end_synchronization イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_synchronization',
  'ExamplePackage.ExampleClass.endSynchronizationConnection'
)
```

次に示すのは、サンプルの Java メソッド endSynchronizationConnection です。このメソッドは JDBC 接続を使用して更新を実行します。これは SQL Anywhere 統合データベース用の構文です。

```
public String endSynchronizationConnection(
  String user )
  throws java.sql.SQLException {
  execUpdate( _syncConn,
   "UPDATE sync_count set count = count + 1 where user_id = '"
   + user + "' " );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndSync という .NET メソッドを end_synchronization 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_synchronization',
  'TestScripts.Test.EndSync'
)
```

次に示すのは、サンプルの .NET メソッド EndSync です。このメソッドは、テーブル sync_count を更新します。これは SQL Anywhere 統合データベース用の構文です。

```
public string EndSync(
  string user ) {
  return(
   "UPDATE sync_count set count = count + 1 where user_id = '"
   + user + "' ");
  return ( null );
}
```


end_synchronization テーブル・イベント

同期処理の完了時にアプリケーションを Mobile Link サーバから切断する時点で、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.synchronization_ok	INTEGER。この値は、同期が成功すると 1 に、失敗すると 0 になります。	3

デフォルトのアクション

なし

備考

アプリケーションが同期を終了し、Mobile Link サーバから切断しようとしているとき、Mobile Link サーバは、同じ名前の接続レベルのスクリプトの前に、このスクリプトを実行します。

リモート・データベースのテーブルごとに、end_synchronization スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_synchronization テーブル・イベント」 404 ページ
- 「end_synchronization 接続イベント」 448 ページ
- 「end_synchronization テーブル・イベント」 451 ページ
- 「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』

SQL の例

次の SQL Anywhere の SQL スクリプトは、begin_synchronization スクリプトが作成したテンポラリ・テーブルを削除します。

```
CALL ml_add_table_script(
 'ver1',
 'sales_order',
 'end_synchronization',
 'DROP TABLE #sales_order' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endSynchronizationTable という Java メソッドを end_synchronization テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_synchronization',
  'ExamplePackage.ExampleClass.endSynchronizationTable' )
```

次に示すのは、サンプルの Java メソッド endSynchronizationTable です。このメソッドは、begin_synchronization スクリプトが作成したテンポラリ・テーブルを削除する SQL 文を返します。

```
public String endSynchronizationTable(
  String user,
  String table ) {
  return( "DROP TABLE #sales_order" );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndTableSync という .NET メソッドを end_synchronization テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
 'ver1',
 'table1',
 'end_synchronization',
 'TestScripts.Test.EndTableSync'
)
```


次に示すのは、サンプルの .NET メソッド EndTableSync です。このメソッドは、begin_synchronization スクリプトが作成したテンポラリ・テーブルを削除する SQL 文を返します。

```
public string EndTableSync(
  string user,
  string table ) {
  return( "DROP TABLE #sales_order" );
}
```

end_upload 接続イベント

Mobile Link サーバがアップロードされた挿入、更新、削除の処理を完了した直後に、任意の文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1

デフォルトのアクション

なし

備考

Mobile Link サーバは、アップロードした情報を処理する最後の手順としてこのスクリプトを実行します。アップロード情報は 1 つのトランザクションで処理されます。このスクリプトは、このトランザクションで統計スクリプトの前に実行される最後のアクションです。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_upload 接続イベント」 407 ページ
- 「end_upload テーブル・イベント」 456 ページ
- 「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』

SQL の例

次の SQL Anywhere の SQL スクリプト は EndUpload ストアド・プロシージャを呼び出します。

```
CALL ml_add_connection_script(
 'ver1',
 'sales_order',
 'end_upload',
 'CALL EndUpload({ml s.username});' )
```


Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadConnection という Java メソッドを end_upload 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_upload',
  'ExamplePackage.ExampleClass.endUploadConnection' )
```

次に示すのは、サンプルの Java メソッド endUploadConnection です。このメソッドは、データベースの操作を実行するメソッドを呼び出します。

```
public String endUploadConnection( String user ) {
  // Clean up new and old tables.
  Iterator    two_iter = _tables_with_ops.iterator();
  while( two_iter.hasNext() ) {
    TableInfo cur_table = (TableInfo)two_iter.next();
    dumpTableOps( _sync_conn, cur_table );
  }
  _tables_with_ops.clear();
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、EndUpload という .NET メソッドを end_upload 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'end_upload',
  'TestScripts.Test.EndUpload'
)
```

次に示すのは、サンプルの .NET メソッド EndUpload です。このメソッドは、EndUpload ストアド・プロシージャを呼び出す SQL 文を返します。

```
public string EndUpload( string user ) {
  return ( "CALL EndUpload({ml s.username});" );
}
```

end_upload テーブル・イベント

Mobile Link サーバがアップロードされた挿入、更新、削除のストリーム処理を終了した直後に、特定のテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

パラメータ	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

デフォルトのアクション

なし

備考

Mobile Link サーバは、アップロードした情報を処理する最後の手順としてこのスクリプトを実行します。アップロード情報は別のトランザクションで処理されます。このスクリプトの実行が、このトランザクションで最後のテーブル固有のアクションになります。

リモート・データベースのテーブルごとに、end_upload スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_upload テーブル・イベント」 409 ページ
- 「end_upload 接続イベント」 454 ページ
- 「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』

SQL の例

次の Mobile Link システム・プロシージャ・コールは end_upload イベントを、ULCustomerIDPool_maintain というストアド・プロシージャに割り当てます。

```
CALL ml_add_table_script(
  'custdb',
  'ULCustomerIDPool',
  'end_upload',
  'CALL ULCustomerIDPool_maintain( username );' );
```

次の SQL 文は ULCustomerIDPool_maintain ストアド・プロシージャを作成します。

```
ALTER PROCEDURE "DBA"."ULCustomerIDPool_maintain" ( IN last_table_download TIMESTAMP, 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
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadTable という Java メソッドを end_upload テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_upload',
  'ExamplePackage.ExampleClass.endUploadTable' )
```

次に示すのは、サンプルの Java メソッド endUploadTable です。このメソッドは、渡されたテーブル名と関連する名前のテーブルに対する削除を生成します。これは SQL Anywhere 統合データベース用の構文です。

```
public String endUploadTable(
  String user,
  String table ) {
  return( "DELETE from '" + table + "_temp'" );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndUpload という .NET メソッドを end_upload テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
```

```
    'end_upload',
    'TestScripts.Test.EndUpload'
)
```

次の .NET サンプルは、テンポラリ・テーブルに挿入されたローを、スクリプトに渡されたテーブルに移動します。

```
public string EndUpload( string user, string table ) {
  DBCommand stmt = curConn.CreateCommand();
  // Move the uploaded rows to the destination table.
  stmt.CommandText = "INSERT INTO "
    + table
    + " SELECT * FROM dnet_ul_temp";
  stmt.ExecuteNonQuery();
  stmt.Close();
  return ( null );
}
```


end_upload_deletes テーブル・イベント

リモート・データベース内の指定されたテーブルからアップロードされた削除を適用した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

デフォルトのアクション

なし

備考

このスクリプトは、2 番目のパラメータで指定したリモート・テーブルでローを削除した結果生じる変更を適用した直後に実行されます。

リモート・データベースのテーブルごとに、end_upload_deletes スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「begin_upload_deletes テーブル・イベント」 412 ページ
- 「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』

SQL の例

このイベントを使用すると、中間テーブル上でアップロード中に削除されたローを処理できます。ベース・テーブルのローを中間テーブルのローと比較して、削除されたローの処理を決定できます。

次の Mobile Link システム・プロシージャ・コールは、EndUploadDeletesLeads ストアド・プロシージャを end_upload_deletes イベントに割り当てます。

```
CALL ml_add_table_script(
  'version1',
  'Leads',
  'end_upload_deletes',
  'call EndUploadDeletesLeads()');
```

次の SQL 文は EndUploadDeletes ストアド・プロシージャを作成します。

```
CREATE PROCEDURE EndUploadDeletesLeads ( )
Begin
  FOR names AS curs CURSOR FOR
  SELECT LeadID
    FROM Leads
    WHERE LeadID NOT IN (SELECT LeadID FROM T_Leads);
  DO
   CALL decide_what_to_do( LeadID )
  END FOR;
end
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadDeletes という Java メソッドを end_upload_deletes テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'end_upload_deletes',
   'ExamplePackage.ExampleClass.endUploadDeletes' )
```

次に示すのは、サンプルの Java メソッド endUploadDeletes です。このメソッドは、データベースを操作する Java メソッドを呼び出します。

```
public String endUploadDeletes(
  String user,
  String table )
  throws java.sql.SQLException {
  processUploadedDeletes( _syncConn, table );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndUploadDeletes という .NET メソッドを end_upload_deletes テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
   'ver1',
   'table1',
```



```
    'end_upload_deletes',
    'TestScripts.Test.EndUploadDeletes'
)
```

次に示すのは、サンプルの .NET メソッド EndUploadDeletes です。このメソッドは、データベースを操作する .NET メソッドを呼び出します。

```
public string EndUploadDeletes(
  string user,
  string table ) {
  processUploadedDeletes( _syncConn, table );
  return ( null );
}
```

end_upload_rows テーブル・イベント

リモート・データベース内の指定されたテーブルからアップロードされた挿入と更新を適用した直後に、そのテーブルに関連した文を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

デフォルトのアクション

なし

備考

アップロードされた情報によっては、統合データベースでローを挿入したり更新したりする必要があります。このスクリプトは、2 番目のパラメータで指定したリモート・テーブルに対する修正の結果生じる変更を適用した直後に実行されます。

リモート・データベースのテーブルごとに、end_upload_rows スクリプトを 1 つ指定できます。

参照

- 「スクリプトの追加と削除」 345 ページ
- 「begin_upload_rows テーブル・イベント」 415 ページ
- 「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』

SQL の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadRows という SQL メソッドを EndUploadRows テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_table_script(
  'version1',
  'table1',
  'end_upload_rows',
  'CALL EndUploadRows(
     { ml s.username },
     { ml s.table } )' )
```

次に示すのは、サンプルの SQL メソッド EndUploadRows です。このメソッドは、データベースを操作する SQL メソッドを呼び出します。

```
CREATE PROCEDURE EndUploadRows (
 IN user VARCHAR(128),
 IN table VARCHAR(128) ),
BEGIN
   CALL decide_what_to_do(table);
END;
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、endUploadRows という Java メソッドを end_upload_rows テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
   'ver1',
   'table1',
   'end_upload_rows',
   'ExamplePackage.ExampleClass.endUploadRows' )
```

次に示すのは、サンプルの Java メソッド endUploadRows です。このメソッドは、データベースを操作する Java メソッドを呼び出します。

```
public String endUploadRows(
  String user,
  String table )
  throws java.sql.SQLException {
  processUploadedRows( _syncConn, table );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、EndUploadRows という .NET メソッドを end_upload_rows テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'end_upload_rows',
  'TestScripts.Test.EndUploadRows'
)
```

次に示すのは、サンプルの .NET メソッド EndUploadRows です。このメソッドは、データベースを操作する .NET メソッドを呼び出します。

```
public string EndUploadRows(
  string user,
  string table ) {
  processUploadedRows( _syncConn, table );
  return ( null );
}
```


handle_DownloadData 接続イベント

ダイレクト・ロー・ハンドリングにおいて、ダウンロードするロー・セットの作成に使用されます。

パラメータ

なし

デフォルトのアクション

なし

備考

handle_DownloadData イベントを使用すると、ダイレクト・ロー・ハンドリングを使用して Mobile Link クライアントにダウンロードする操作を決定できます。

ダイレクト・ロー・ハンドリングは、Mobile Link でサポートされている統合データベース以外のデータ・ソースと同期するために使用されます。「ダイレクト・ロー・ハンドリング」 679 ページを参照してください。

ダイレクト・ダウンロードを作成するには、Java または .NET 用 Mobile Link サーバ API の DownloadData クラスと DownloadTableData クラスを使用できます。

Java の場合、DBConnectionContext の getDownloadData メソッドは、現在の同期に対する DownloadData インスタンスを返します。DownloadData はすべてのダウンロード操作をカプセル化して、リモート・クライアントに送信します。DownloadData の getDownloadTables メソッドと getDownloadTableByName メソッドを使用して DownloadTableData インスタンスを取得できます。DownloadTableData は、特定のテーブルに対するダウンロード操作をカプセル化します。getUpsertPreparedStatement メソッドを使用して、挿入と更新操作の準備文を取得できます。DownloadTableData の getDeletePreparedStatement メソッドを使用して、削除操作の準備文を取得できます。

.NET の場合、DBConnectionContext の GetDownloadData メソッドは、現在の同期に対する DownloadData インスタンスを返します。DownloadData はすべてのダウンロード操作をカプセル化して、リモート・クライアントに送信します。DownloadData の GetDownloadTables メソッドと GetDownloadTableByName メソッドを使用して DownloadTableData インスタンスを取得できます。DownloadTableData は、特定のテーブルに対するダウンロード操作をカプセル化します。GetUpsertCommand メソッドを使用して、挿入と更新操作のコマンドを取得できます。DownloadTableData の getDeleteCommand メソッドを使用して、削除操作のコマンドを取得できます。

Java の場合については、「DBConnectionContext インタフェース」 570 ページを参照してください。.NET の場合については、「DBConnectionContext インタフェース」 639 ページを参照してください。

handle_DownloadData または他の同期イベントでダウンロードを作成できます。Mobile Link にはこの柔軟性があるので、データがアップロードされたときまたは特定のイベントが発生したときのダウンロード操作を設定できます。handle_DownloadData 以外のイベントでダイレクト・ダウンロードを作成する場合は、含まれているメソッドが何も処理をしない handle_DownloadData スクリプトを作成してください。Mobile Link でダイレクト・ロー・ハンドリングを有効にするに

は、このスクリプトが定義されている必要があります。アップロード専用同期の場合を除き、Mobile Link サーバでは、少なくとも handle_DownloadData スクリプトが 1 つ定義されている必要があります。

ダイレクト・ダウンロードを handle_DownloadData 以外のイベントで作成する場合、そのイベントは begin_synchronization イベントより前または end_download イベントより後に実装できません。

注意
このイベントは SQL として実装できません。

参照

- 「ダイレクト・ロー・ハンドリング」 679 ページ
- 「ダイレクト・ダウンロードの処理」 692 ページ
- Java：「DownloadData インタフェース」 575 ページ
- Java：「DownloadTableData インタフェース」 577 ページ
- .NET：「DownloadData インタフェース」 652 ページ
- .NET：「DownloadTableData インタフェース」 654 ページ
- 「handle_UploadData 接続イベント」 477 ページ
- 「必要なスクリプト」 344 ページ
- 「スクリプトの追加と削除」 345 ページ

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handle_DownloadData 接続イベントに対して handleDownload という Java メソッドを登録します。Mobile Link 統合データベースに対してこのシステム・プロシージャを実行します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'handle_DownloadData',
 'MyPackage.MyClass.handleDownload' )
```

「ml_add_java_connection_script システム・プロシージャ」 705 ページを参照してください。

次の例は、handleDownload メソッドを使用してダウンロードを作成する方法を示します。

次のコードは、クラス・レベルの DBConnectionContext インスタンスを MobiLinkOrders クラスのコンストラクタ内で設定します。

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

public class  MobiLinkOrders{

  DBConnectionContext _cc;

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


HandleDownload メソッドでは、DBConnectionContext の getDownloadData メソッドを使用して、現在の同期に対する DownloadData インスタンスを返します。DownloadData の getDownloadTableByName メソッドは、remoteOrders テーブルの DownloadTableData インスタンスを返します。DownloadTableData の getUpsertPreparedStatement メソッドは java.sql.PreparedStatement を返します。ダウンロードに操作を追加するには、すべてのカラム値を設定して、executeUpdate メソッドを呼び出します。

次に示すのは、MobiLinkOrders クラスの handleDownload メソッドです。ここでは、remoteOrders テーブル用のダウンロードに 2 つのローを追加しています。

```
// Method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
  // Get DownloadData instance for current synchronization.
  DownloadData downloadData = _cc.getDownloadData();

  // Get a DownloadTableData instance for the remoteOrders table.
  DownloadTableData td = downloadData.getDownloadTableByName("remoteOrders");

  // Get a java.sql.PreparedStatement for upsert (update/insert) operations.
  PreparedStatement upsertPS = td.getUpsertPreparedStatement();

  // Set values for one row.
  upsertPS.setInt( 1, 2300 );
  upsertPS.setInt( 2, 100 );

  // Add the values to the download.
  int updateResult = upsertPS.executeUpdate();

  // Set values for another row.
  upsertPS.setInt( 1, 2301 );
  upsertPS.setInt( 2, 50 );
  updateResult = upsertPS.executeUpdate();

  upsertPS.close();

  // ...
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、HandleDownload という .NET メソッドを handle_DownloadData 接続イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_dnet_connection_script(
  'ver1', 'handle_DownloadData',
  'TestScripts.Test.HandleDownload'
)
```

次に示すのは、サンプルの .NET メソッド HandleDownload です。

```
using System;
using System.Data;
using System.IO;
using iAnywhere.MobiLink.Script;
using iAnywhere.MobiLink;

namespace MyScripts
{
    /// <summary>
```

```
    /// Tests that scripts are called correctly for most sync events.
    /// </summary>
    public class MobiLinkOrders
    {
  private DBConnectionContext _cc;

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

  ~MobiLinkOrders()
  {
  }

  public void handleDownload()
  {
     // Get DownloadData instance for current synchronization.
     DownloadData my_dd = _cc.GetDownloadData();

     // Get a DownloadTableData instance for the remoteOrders table.
     DownloadTableData td = my_dd.GetDownloadTableByName("remoteOrders");

     // Get an IdbCommand for upsert (update/insert) operations.
     IDbCommand upsert_stmt = td.GetUpsertCommand();

     IDataParameterCollection parameters = upsert_stmt.Parameters;

     // Set values for one row.
     parameters[ 0 ] = 2300;
     parameters[ 1 ] = 100;

     // Add the values to the download.
     int update_result = upsert_stmt.ExecuteNonQuery();

     // Set values for another row.
     parameters[ 0 ] = 2301;
     parameters[ 1 ] = 50;
     update_result = upsert_stmt.ExecuteNonQuery();

     // ...
  }

    }
}
```


handle_error 接続イベント

Mobile Link サーバで SQL エラーが発生したときに実行されます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.action_code	INTEGER。これは INOUT パラメータです。	1
s.error_code	INTEGER	2
s.error_message	TEXT	3
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	4
s.table	VARCHAR(128)。スクリプトがテーブル・スクリプトでない場合、テーブル名は NULL です。	5

デフォルトのアクション

handle_error スクリプトが定義されない場合、またはこのスクリプトが原因でエラーが発生した場合、デフォルト・アクション・コードは 3000 です。現在のトランザクションをロールバックし、現在の同期をキャンセルします。

備考

Mobile Link サーバは、現在のアクション・コードで送信します。最初は 1 回の SQL 操作によって発生したエラーのセットごとに 3000 が設定されます。通常、エラー数は SQL 操作ごとに 1 つのみですが、複数の場合もあります。この handle_error スクリプトは、セットに含まれるエラーごとに 1 回呼び出されます。最初のエラーに渡されるアクション・コードは 3000 です。以降の

呼び出しには、直前の呼び出しから返されたアクション・コードが渡されます。Mobile Link は、複数の呼び出しから返される値のうち最も大きい番号が付いた値を使用します。

スクリプト内でアクション・コードを修正し、Mobile Link に次の処理を指示する値を返すことができます。Mobile Link サーバが次に何を行うかは、アクション・コードに示されます。Mobile Link サーバは、エラーの重大度に応じてアクション・コードにデフォルト値を設定してから、このスクリプトを呼び出します。この値は、スクリプトで変更できます。スクリプトは、必ずアクション・コードを返すか設定するようにします。

アクション・コード・パラメータには、次のいずれかの値を指定します。

- **1000** 現在のローをスキップして、処理を続行します。
- **3000** 現在のトランザクションをロールバックし、現在の同期をキャンセルします。これはデフォルト・アクション・コードで、handle_error スクリプトが定義されない場合、またはこのスクリプトが原因でエラーが発生した場合に使用されます。
- **4000** 現在のトランザクションのロールバック、同期のキャンセル、Mobile Link サーバの停止を行います。

エラーの内容は、エラー・コードとメッセージで識別できます。同期の一部としてエラーが発生した場合は、ユーザ名が指定されます。それ以外の場合、この値は NULL です。

Mobile Link がアップロード・トランザクション中に挿入、更新、または削除スクリプトを処理している間、またはダウンロード・ローをフェッチしている間に ODBC エラーが発生した場合、Mobile Link サーバはこのスクリプトを実行します。別のときに ODBC エラーが発生した場合、Mobile Link サーバは report_error または report_ODBC_error スクリプトを呼び出して、同期をアボートします。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、クライアント・アプリケーションでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、同期システムの設計によって異なります。

handle_error イベント用の SQL スクリプトは、ストアド・プロシージャとして実装してください。

次のいずれかの方法で、handle_error スクリプトから値を返すことができます。

- 次のように、プロシージャの OUTPUT パラメータにアクション・パラメータを渡す。

  ```
  CALL my_handle_error( {ml s.action_code}, {ml s.error_code}, {ml s.error_message}, {ml
  s.username}, {ml s.table} )
  ```

- 次のように、プロシージャまたは関数の戻り値を介してアクション・コードを設定する。

  ```
  {ml s.action_code} = CALL my_handle_error( {ml s.error_code}, {ml s.error_message}, {ml
  s.username}, {ml s.table} )
  ```

  ほとんどの RDBMS では、RETURN 文を使用してプロシージャまたは関数からの戻り値を設定します。

CustDB サンプル・アプリケーションには、さまざまなデータベース管理環境に対するエラー・ハンドラが含まれています。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「report_error 接続イベント」 502 ページ
- 「report_odbc_error 接続イベント」 505 ページ
- 「handle_odbc_error 接続イベント」 473 ページ
- 「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』

SQL の例

次の例は SQL Anywhere 統合データベースで動作します。これによって、アプリケーションは冗長挿入を無視できます。

次の Mobile Link システム・プロシージャ・コールは、ULHandleError ストアド・プロシージャを handle_error イベントに割り当てます。

```
CALL ml_add_connection_script(
   'ver1',
   'handle_error',
   'CALL ULHandleError(
     {ml s.action_code},
     {ml s.error_code},
     {ml s.error_message},
     {ml s.username},
     {ml s.table} )' )
```

次の SQL 文は ULHandleError ストアド・プロシージャを作成します。

```
CREATE PROCEDURE ULHandleError(
  INOUT action integer,
  IN error_code integer,
  IN error_message varchar(1000),
  IN user_name varchar(128),
  IN table_name varchar(128) )
BEGIN
 -- -196 is SQLE_INDEX_NOT_UNIQUE
 -- -194 is SQLE_INVALID_FOREIGN_KEY
 IF error_code = -196 or error_code = -194 then
    -- ignore the error and keep going
    SET action = 1000;
 ELSE
    -- abort the synchronization
    SET action = 3000;
 END IF;
END
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handleError という Java メソッドを handle_error 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'handle_error',
   'ExamplePackage.ExampleClass.handleError' )
```

次に示すのは、サンプルの Java メソッド handleError です。このメソッドは、渡されたデータに基づいてエラーを処理します。また、エラーの結果生じるエラー・コードも判別します。

```
public String handleError(
  ianywhere.ml.script.InOutInteger actionCode,
  int errorCode,
  String errorMessage,
  String user,
  String table ) {
  int newAC;
  if( user == null ) {
    newAC = handleNonSyncError( errorCode,
    errorMessage ); }
  else if( table == null ) {
    newAC = handleConnectionError( errorCode,
    errorMessage, user ); }
  else {
    newAC = handleTableError( errorCode,
    errorMessage, user, table );
  }
  // Keep the most serious action code.
  if( actionCode.getValue() < newAC ) {
    actionCode.setValue( newAC );
  }
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、HandleError という .NET メソッドを handle_error 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'handle_error',
  'TestScripts.Test.HandleError' )
```

次に示すのは、サンプルの .NET メソッド HandleError です。

```
public string HandleError() (
  ref int actionCode,
  int errorCode,
  string errorMessage,
  string user,
  string table ) {
  int new_ac;
  if( user == null ) {
    new_ac = HandleNonSyncError( errorCode,
    errorMessage ); }
  else if( table == null ) {
    new_ac = HandleConnectionError( errorCode,
    errorMessage, user ); }
  else {
    new_ac = HandleTableError( errorCode,
    errorMessage, user, table );
  }
  // Keep the most serious action code.
  if( actionCode < new_ac ) {
    actionCode = new_ac;
  }
}
```


handle_odbc_error 接続イベント

ODBC ドライバ・マネージャによってトリガされたエラーを Mobile Link サーバが検出すると、実行されます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.action_code	INTEGER。これは INOUT パラメータです。	1
s.ODBC_state	VARCHAR(5)	2
s.error_message	TEXT	3
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	4
s.table	VARCHAR(128)	5

デフォルトのアクション

Mobile Link サーバは、デフォルトのアクション・コードを選択します。スクリプト内でアクション・コードを修正し、Mobile Link に次の処理を指示する値を返すことができます。アクション・コード・パラメータには、次のいずれかの値を指定します。

- **1000** 現在のローをスキップして、処理を続行します。
- **3000** 現在のトランザクションをロールバックし、現在の同期をキャンセルします。これはデフォルト・アクション・コードで、handle_error スクリプトが定義されない場合、またはこのスクリプトが原因でエラーが発生した場合に使用されます。
- **4000** 現在のトランザクションのロールバック、同期のキャンセル、Mobile Link サーバの停止を行います。

備考

Mobile Link がアップロード・トランザクション中に挿入、更新、または削除スクリプトを処理している間、またはダウンロード・ローをフェッチしている間に発生し、ODBC ドライバ・マネージャによってフラグが設定されたエラーを検出した場合、Mobile Link サーバはこのスクリプトを実行します。別のときに ODBC エラーが発生した場合、Mobile Link サーバは report_error または report_ODBC_error スクリプトを呼び出して、同期をアボートします。

エラーの内容は、エラー・コードで識別できます。

Mobile Link サーバが次に何を行うかは、アクション・コードに示されます。Mobile Link サーバは、エラーの重大度に応じてアクション・コードにデフォルト値を設定してから、このスクリプトを呼び出します。この値は、スクリプトで変更できます。スクリプトは、必ずアクション・コードを返すか設定するようにします。

handle_odbc_error スクリプトは、handle_error スクリプトと report_error スクリプトの後、report_odbc_error スクリプトの前に呼び出されます。

エラー処理スクリプトがどちらか 1 つだけ定義されている場合、そのスクリプトからの戻り値によってエラー処理が決定されます。エラー処理スクリプトが両方とも定義されている場合、Mobile Link サーバは数値の一番大きいアクション・コードを使用します。handle_error と handle_ODBC_error の両方が定義されると、Mobile Link はすべての呼び出しから返されるアクション・コードのうち、最も大きい番号を持つものを使用します。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「handle_error 接続イベント」 469 ページ
- 「report_error 接続イベント」 502 ページ
- 「report_odbc_error 接続イベント」 505 ページ
- 「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』

SQL の例

次の例は SQL Anywhere 統合データベースで動作します。これによって、アプリケーションは ODBC 整合性制約違反を無視できます。

次の Mobile Link システム・プロシージャ・コールは、HandleODBCError ストアド・プロシージャを handle_odbc_error イベントに割り当てます。

```
CALL ml_add_connection_script(
 'ver1',
 'handle_odbc_error',
 'CALL HandleODBCError(
   {ml s.action_code},
   {ml s.ODBC_state},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )' )
```

次の SQL 文は HandleODBCError ストアド・プロシージャを作成します。

```
CREATE PROCEDURE HandleODBCError(
  INOUT action integer,
```



```
  IN odbc_state varchar(5),
  IN error_message varchar(1000),
  IN user_name varchar(128),
  IN table_name varchar(128) )
 BEGIN
  IF odbc_state = '23000' then
    -- Ignore the error and keep going.
    SET action = 1000;
  ELSE
    -- Abort the synchronization.
    SET action = 3000;
  END IF;
END
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handleODBCError という Java メソッドを handle_odbc_error イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'handle_odbc_error',
 'ExamplePackage.ExampleClass.handleODBCError'
)
```

次に示すのは、サンプルの Java メソッド handleODBCError です。このメソッドは、渡されたデータに基づいてエラーを処理します。また、エラーの結果生じるエラー・コードも判別します。

```
public String handleODBCError(
  ianywhere.ml.script.InOutInteger actionCode,
  String ODBCState,
  String errorMessage,
  String user,
  String table ) {
  int newAC;
  if( user == null ) {
    newAC = handleNonSyncError( ODBCState,
  errorMessage );
  }
  else if( table == null ) {
    newAC = handleConnectionError( ODBCState,
     errorMessage, user );
  } else {
    newAC = handleTableError( ODBCState,
     errorMessage, user, table );
  }
  // Keep the most serious action code.
  if( actionCode.getValue() < newAC ) {
    actionCode.setValue( newAC );
  }
  return( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、HandleODBCError という .NET メソッドを handle_odbc_error イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'handle_odbc_error',
 'TestScripts.Test.HandleODBCError' )
```

次に示すのは、サンプルの .NET メソッド HandleODBCError です。

```
public string HandleODBCError (
  ref int actionCode,
  string ODBCState,
  string errorMessage,
  string user,
  string table ) {
  int new_ac;
  if( user == null ) {
    new_ac = HandleNonSyncError( ODBCState,
      errorMessage );
  }
  else if( table == null ) {
    new_ac = HandleConnectionError( ODBCState,
     errorMessage, user );
  } else {
    new_ac = HandleTableError( ODBCState,
     errorMessage, user, table );
  }
  // Keep the most serious action code.
  if( actionCode < new_ac ) {
    actionCode = new_ac;
  }
  return( null );
}
```


handle_UploadData 接続イベント

ダイレクト・ロー・ハンドリングにおいて、アップロードされたローの処理に使用されます。

パラメータ

SQL スクリプトのパラメータ名	説明	順序
UploadData	.NET または Java クラスは、Mobile Link クライアントによってアップロードされたテーブル操作をカプセル化します。このクラスは、Java と .NET 用 Mobile Link サーバ API で定義されています。	1

デフォルトのアクション

なし

備考

handle_UploadData イベントを使用すると、Mobile Link のダイレクト・ロー・ハンドリングでアップロードを処理できます。このイベントは、同期のアップロード・トランザクションごとに発生します。ただし、トランザクション・レベル・アップロードを使用している場合は、トランザクションごとに発生します。

「ダイレクト・ロー・ハンドリング」 679 ページを参照してください。

このイベントは 1 つの UploadData パラメータを取ります。Java または .NET メソッドは UploadData の getUploadedTables または getUploadedTableByName メソッドを使用して、UploadedTableData インスタンスを取得できます。UploadedTableData を使用して、現在の同期で Mobile Link クライアントによってアップロードされた挿入、更新、削除操作にアクセスできます。

UploadData と UploadedTableData クラスの詳細については、「ダイレクト・アップロードの処理」 685 ページを参照してください。

カラム名メタデータを読み込む場合は、SendColumnNames Mobile Link クライアント拡張オプションまたはプロパティを指定してください。それ以外の方法でカラムを参照するには、リモート・データベースに定義されているインデックスを使用します。

「SendColumnNames (scn) 拡張オプション」『Mobile Link - クライアント管理』と「Send Column Names 同期パラメータ」『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。

注意
このイベントは SQL として実装できません。

参照

- 「ダイレクト・ロー・ハンドリング」 679 ページ
- 「ダイレクト・アップロードの処理」 685 ページ
- Java：「UploadData インタフェース」 607 ページ
- Java：「UploadedTableData インタフェース」 609 ページ
- .NET：「UploadData インタフェース」 672 ページ
- .NET：「UploadedTableData インタフェース」 674 ページ
- dbmlsync：「SendColumnNames (scn) 拡張オプション」 『Mobile Link - クライアント管理』
- Ultra Light：「Send Column Names 同期パラメータ」 『Ultra Light データベース管理とリファレンス』
- 「handle_DownloadData 接続イベント」 465 ページ
- 「必要なスクリプト」 344 ページ
- 「スクリプトの追加と削除」 345 ページ

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handle_UploadData 接続イベントに対して handleUpload という Java メソッドを登録します。Mobile Link 統合データベースに対してこのシステム・プロシージャを実行します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'handle_UploadData',
  'MyPackage.MyClass.handleUpload' )
```

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」 705 ページを参照してください。

次の Java メソッドは remoteOrders テーブルのアップロードを処理します。UploadData.getUploadedTableByName メソッドは remoteOrders テーブルの UploadedTableData インスタンスを返します。UploadedTableData の getInserts メソッドは、新しいローを表す java.sql.ResultSet インスタンスを返します。

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

public void handleUpload( UploadData ut )
  throws SQLException, IOException {
  // Get an UploadedTableData instance representing the
  // remoteOrders table.
  UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");
  // Get inserts uploaded by the MobiLink client.
  java.sql.ResultSet results = remoteOrdersTable.getInserts();
  while( results.next() ) {
    // You can reference column names here because SendColumnNames is on
    // Get the primary key.
    int pk = results.getInt("pk");

    // Get the uploaded num_ordered value.
    int numOrdered = results.getInt("num_ordered");

    // The current insert row is now ready to be uploaded to wherever
    // you want it to go (a file, a web service, and so on).
```



```
    }

    results.close();

  }
```

次の例は、Mobile Link リモート・データベースによってアップロードされた挿入、更新、削除操作を出力します。UploadData の getUploadedTables メソッドは、リモートによってアップロードされたすべてのテーブルを表す UploadedTableData インスタンスを返します。配列内でのテーブルの順序は、リモートによってアップロードされた順序と同じです。UploadedTableData の getInserts、getUpdates、getDeletes メソッドは標準 JDBC 結果セットを返します。println メソッドまたは出力データを使用して、テキスト・ファイルまたは別の場所に出力できます。

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

public void handleUpload( UploadData ud )
  throws SQLException, IOException {
  UploadedTableData tables[] = ud.getUploadedTables();
  for( int i = 0; i < tables.length; i++ ) {
    UploadedTableData currentTable = tables[i];
    println( "table " + java.lang.Integer.toString( i ) +
     " name: " + currentTable.getName() );
    // Print out delete result set.
    println( "Deletes" );
    printRSInfo( currentTable.getDeletes() );
    // Print out insert result set.
    println( "Inserts" );
    printRSInfo( currentTable.getInserts() );
    // print out update result set
    println( "Updates" );
    printUpdateRSInfo( currentTable.getUpdates() );
  }
}
```

printRSInfo メソッドは、挿入、更新、または削除の結果セットを出力し、1 つの java.sql.ResultSet オブジェクトを受け入れます。カラム・ラベルを含む、詳細なカラム情報は、ResultSet の getMetaData メソッドによって返される ResultSetMetaData オブジェクトによって提供されます。カラム・ラベルは、クライアントで SendColumnNames オプションが有効になっている場合にのみ使用できます。printRow メソッドは結果セットの各ローを出力します。

```
public void printRSInfo( ResultSet results )
  throws SQLException, IOException {

  // Obtain the result set metadata.
  ResultSetMetaData metaData = results.getMetaData();
  String columnHeading = "";

  // Print out column headings.
  for( int c = 1; c <= metaData.getColumnCount(); c++ ) {
    columnHeading += metaData.getColumnLabel(c);
    if( c < metaData.getColumnCount() ) {
      columnHeading += ", ";
    }
  }

  println( columnHeading );
  while( results.next() ) {
```

```
    // Print out each row.
    printRow( results, metaData.getColumnCount() );
  }

  // Close the java.sql.ResultSet.
  results.close();
}
```

次に示す printRow メソッドは ResultSet の getString メソッドを使用して、各カラム値を取得します。

```
public void printRow( ResultSet results, int colCount )
  throws SQLException, IOException {
  String row = "( ";

  for( int c = 1; c <= colCount; c++ ) {
    // Get a column value.
    String currentColumn = results.getString( c );

    // Check for null values.
    if( currentColumn == null ) {
      currentColumn = "<NULL>";
    }

    // Add the column value to the row string.
    row += cur_col;
    if( c < colCount ) {
      row += ", ";
    }
  }

  row += " )";

  // Print out the row.
  println( row );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handle_UploadData 接続イベントに対して HandleUpload という .NET メソッドを登録します。Mobile Link 統合データベースに対してこのシステム・プロシージャを実行します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'handle_UploadData',
  'TestScripts.Test.HandleUpload' )
```

次の .NET メソッドは remoteOrders テーブルのアップロードを処理します。

```
using System;
using System.Data;
using System.IO;
using iAnywhere.MobiLink.Script;
using iAnywhere.MobiLink;

namespace MyScripts
{
    public class MyUpload
    {
 public MyUpload( DBConnectionContext cc )
```



```
{
}

~MyUpload()
{
}

public void handleUpload( UploadData ut )
{
    int i;
    UploadedTableData[] tables = ut.GetUploadedTables();

    for( i=0; i<tables.Length; i+=1 ) {
 UploadedTableData cur_table = tables[i];
 Console.Write( "table " + i + " name: " + cur_table.GetName() );
 // Print out delete result set.
 Console.Write( "Deletes" );
 printRSInfo( cur_table.GetDeletes() );

 // Print out insert result set.
 Console.Write( "Inserts" );
 printRSInfo( cur_table.GetInserts() );

 // print out update result set
 Console.Write( "Updates" );
 printUpdateRSInfo( cur_table.GetUpdates() );
    }
}

public void printRSInfo( IDataReader dr )
{
    // Obtain the result set metadata.
    DataTable dt = dr.GetSchemaTable();
    DataColumnCollection cc = dt.Columns;
    DataColumn dc;
    String columnHeading = "";

    // Print out column headings.
    for( int c=0; c < cc.Count; c = c + 1 ) {
 dc = cc[ c ];
 columnHeading += dc.ColumnName;
 if( c < cc.Count - 1 ) {
     columnHeading += ", ";
 }
    }
    Console.Write( columnHeading );

    while( dr.Read() ) {
 // Print out each row.
 printRow( dr, cc.Count );
    }

    // Close the java.sql.ResultSet.
    dr.Close();
}

public void printUpdateRSInfo( UpdateDataReader utr )
{
    // Obtain the result set metadata.
    DataTable dt = utr.GetSchemaTable();
    DataColumnCollection cc = dt.Columns;
    DataColumn dc;
    String columnHeading = "TYPE, ";
```

```
    // Print out column headings.
    for( int c = 0; c < cc.Count; c = c + 1 ) {
  dc = cc[ c ];
  columnHeading += dc.ColumnName;
  if( c < cc.Count - 1 ) {
      columnHeading += ", ";
  }
    }
    Console.Write( columnHeading );

    while( utr.Read() ) {
  // Print out the new values for the row.
  utr.SetNewRowValues();
  Console.Write( "NEW:" );
  printRow( utr, cc.Count );

  // Print out the old values for the row.
  utr.SetOldRowValues();
  Console.Write( "OLD:" );
  printRow( utr, cc.Count );
    }

    // Close the java.sql.ResultSet.
    utr.Close();
}

public void printRow( IDataReader dr, int col_count )
{
    String row = "( ";
    int c;

    for( c = 0; c < col_count; c = c + 1 ) {
  // Get a column value.
  String cur_col = dr.GetString( c );

  // Check for null values.
  if( cur_col == null ) {
      cur_col = "<NULL>";
  }

  // Add the column value to the row string.
  row += cur_col;
  if( c < col_count ) {
      row += ", ";
  }
    }

    row += " )";

    // Print out the row.
    Console.Write( row );
}
  }
}
```


modify_error_message 接続イベント

このスクリプトを使用すると、リモート・データベースに送信される (エラー、警告、情報) メッセージのテキストをカスタマイズできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.error_message	VARBINARY(1024)。これは INOUT パラメータです。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.error_code	INT	3

デフォルトのアクション

なし

備考

modify_error_message イベントの SQL スクリプトは、ストアド・プロシージャとして実装してください。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「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』

SQL の例

次の例は、データベースが 1 日前から現在までの間に同期されたかどうかに関わらず、1 日前からのデータをすべてダウンロードします。

次の SQL 文は ModifyLastErrorMessage ストアド・プロシージャを作成します。

```
CREATE PROCEDURE ModifyLastErrorMessage(
  inout error_message VARBINARY(1024),
  in username VARCHAR(128),
  in error_code INT )
BEGIN
  SELECT dateadd(day, -1, last_download_time )
  INTO last_download_time
END
```

次の Mobile Link システム・プロシージャ・コールは ModifyLastErrorMessage を、スクリプト・バージョン modify_ts_test の modify_error_message 接続イベントに割り当てます。

```
CALL ml_add_connection_script(
  'modify_ts_test',
  'modify_error_message',
  'CALL ModifyLastErrorMessage (
    {ml s.error_message},
    {ml s.username},
    {ml s.error_code} )' );
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyLastErrorMessage という Java メソッドを modify_error_message 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'modify_error_message',
  'ExamplePackage.ExampleClass.modifyLastErrorMessage' )
```

次に示すのは、サンプルの Java メソッド modifyLastErrorMessage です。このメソッドは、現在のエラー・メッセージとエラー・コードを出力します。

```
public String modifyLastErrorMessage(
  String lastErrorMessage,
  String userName,
  int errorCode ) {
  java.lang.System.out.println( "error message: " +
   lastErrorMessage );
  java.lang.System.out.println( "error code: " +
   String.valueOf(errorCode) );
  return( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、ModifyLastErrorMessage という .NET メソッドを modify_error_message 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
```



```
  'modify_error_message',
  'TestScripts.Test.ModifyLastErrorMessage' )
```

次に示すのは、サンプルの .NET メソッド ModifyLastErrorMessage です。このメソッドは、現在のエラー・コードとエラー・メッセージを出力します。

```
public string ModifyLastErrorMessage (
  string errorMessage,
  string userName,
  string errorCode  ) {
  System.Console.WriteLine( "error message: " + errorMessage );
  System.Console.WriteLine( "error code: " +  errorCode );
  return ( null );
}
```

modify_last_download_timestamp 接続イベント

このスクリプトを使用して、現在の同期の最終ダウンロード時刻を修正できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_download	TIMESTAMP。同期テーブルの最後のダウンロード時刻。これは INOUT パラメータです。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2

デフォルトのアクション

なし

備考

このスクリプトは、現在の同期に対する last_download タイムスタンプを修正するために使用します。このスクリプトが定義されている場合、Mobile Link サーバは、ダウンロード・スクリプトに渡す last_download タイムスタンプとして、修正した last_download タイムスタンプを使用します。通常、このスクリプトは、リモートで失われたデータをリカバリするために使用します。つまり、last_download タイムスタンプを 1900-01-01 00:00 などの過去の時間にリセットして、次の同期ですべてのデータをダウンロードできます。

modify_last_download_timestamp イベントの SQL スクリプトは、ストアド・プロシージャとして実装してください。Mobile Link サーバは、ストアド・プロシージャへの最初のパラメータとして last_download タイムスタンプを渡し、タイムスタンプをストアド・プロシージャが渡した最初の値で置き換えます。

このスクリプトは、同じトランザクション内にある prepare_for_download スクリプトの直前に実行されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「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」 134 ページ
- 「ダウンロード・タイムスタンプの生成および使用方法」 135 ページ
- 「modify_next_last_download_timestamp 接続イベント」 489 ページ

SQL の例

次の SQL 文はストアド・プロシージャを作成します。次は Oracle 統合データベース用の構文です。

```
CREATE PROCEDURE ModifyDownloadTimestamp(
      download_timestamp  OUT DATE,
      user_name   IN VARCHAR )
      AS
      BEGIN
       -- N is the maximum replication latency in the consolidated cluster
      download_timestamp := download_timestamp - N;
      END;
```

次は、SQL Anywhere、Adaptive Server Enterprise、SQL Server 統合データベース用の構文です。

```
CREATE PROCEDURE ModifyDownloadTimestamp
      @download_timestamp DATETIME OUTPUT,
      @t_name    VARCHAR( 128 )
      AS
      BEGIN
       -- N is the maximum replication latency in consolidated cluster
      SELECT @download_timestamp = @download_timestamp - N
      END
```

次は DB2 メインフレーム統合データベース用の構文です。

```
CREATE PROCEDURE modify_ldts(
      OUT ts      TIMESTAMP,
      IN  t_name  VARCHAR(128) )
      LANGUAGE SQL
      BEGIN
         set ts = TIMESTAMP_FORMAT('2000-01-02 03:04:05','YYYY-MM-DD HH24:MI:SS');
      END
```

次の Mobile Link システム・プロシージャ・コールは、ModifyDownloadTimestamp ストアド・プロシージャを modify_last_download_timestamp イベントに割り当てます。次は SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_connection_script(
  'my_version',
  'modify_last_download_timestamp',
  '{CALL ModifyDownloadTimestamp(
   {ml s.last_download},
   {ml s.username} ) }' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyLastDownloadTimestamp という Java メソッドを modify_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'modify_last_download_timestamp',
 'ExamplePackage.ExampleClass.modifyLastDownloadTimestamp' )
```

次に示すのは、サンプルの Java メソッド modifyLastDownloadTimestamp です。このメソッドは、現在の新しいタイムスタンプを出力し、渡されたタイムスタンプを修正します。

```
public String modifyLastDownloadTimestamp(
     Timestamp lastDownloadTime,
     String userName ) {
   java.lang.System.out.println( "old date: " +
    lastDownloadTime.toString() );
   lastDownloadTime.setDate(
    lastDownloadTime.getDate() -1 );
   java.lang.System.out.println( "new date: " +
    lastDownloadTime.toString() );
   return( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、ModifyLastDownloadTimestamp という .NET メソッドを modify_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'modify_last_download_timestamp',
 'TestScripts.Test.ModifyLastDownloadTimestamp' )
```

次に示すのは、サンプルの .NET メソッド ModifyLastDownloadTimestamp です。

```
public string ModifyLastDownloadTimestamp(
  DateTime lastDownloadTime,
  string userName ) {
  System.Console.WriteLine( "old date: " +
   last_download_time.ToString() );
  last_download_time = DateTime::Now;
  System.Console.WriteLine( "new date: " +
   last_download_time.ToString() );
  return( null );
}
```


modify_next_last_download_timestamp 接続イベント

このスクリプトを使用して、次回の同期の最終ダウンロード時刻を修正できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.next_last_download	TIMESTAMP。これは INOUT パラメータです。Mobile Link サーバは、アップロードがコミットされた直後にこの値を生成します。	1
s.last_download	TIMESTAMP。これは IN パラメータです。これは現在の同期の最終ダウンロード時刻です。next_last_download タイムスタンプを last_download タイムスタンプよりも前の時間に修正していないことを確認すると、重複を避けるのに役立ちます。	2
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	3

デフォルトのアクション

なし

備考

このスクリプトを使用すると、next_last_download タイムスタンプを変更でき、次回の同期の last_download スタンプが効率的に変更されます。これにより、現在の同期に影響を与えずに次回の同期をリセットできます。

modify_next_last_download_timestamp イベントの SQL スクリプトは、ストアド・プロシージャとして実装してください。Mobile Link サーバは、ストアド・プロシージャへの最初のパラメータとして next_last_download タイムスタンプを渡し、タイムスタンプをストアド・プロシージャが渡した最初の値で置き換えます。

このスクリプトは、ユーザ・テーブルがダウンロードされた後にダウンロード・トランザクションで実行されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「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」 134 ページ
- 「ダウンロード・タイムスタンプの生成および使用方法」 135 ページ
- 「modify_last_download_timestamp 接続イベント」 486 ページ

SQL の例

次の例は、このスクリプトの適用例の 1 つを示します。ストアド・プロシージャを作成します。次は SQL Anywhere 統合データベース用の構文です。

```
CREATE PROCEDURE ModifyNextDownloadTimestamp(
  inout download_timestamp TIMESTAMP ,
  in last_download TIMESTAMP ,
  in user_name VARCHAR(128) )
  BEGIN
    SELECT dateadd(hour, -1, download_timestamp )
    INTO download_timestamp
END
```

スクリプトを SQL Anywhere 統合データベースにインストールします。

```
CALL ml_add_connection_script(
  'modify_ts_test',
  'modify_next_last_download_timestamp',
  'CALL  ModifyNextDownloadTimestamp (
   {ml s.next_last_download},
   {ml s.last_download},
   {ml s.username} )' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyNextDownloadTimestamp という Java メソッドを modify_next_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'modify_next_last_download_timestamp',
   'ExamplePackage.ExampleClass.modifyNextDownloadTimestamp' )
```

次に示すのは、サンプルの Java メソッド modifyNextDownloadTimestamp です。このメソッドは、ダウンロード・タイムスタンプを 1 時間前に設定します。

```
public String modifyNextDownloadTimestamp(
  Timestamp downloadTimestamp,
```



```
  Timestamp lastDownload,
  String userName ) {
  downloadTimestamp.setHours(
  downloadTimestamp.getHours() -1 );
  return( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、ModifyNextDownloadTimestamp という .NET メソッドを modify_next_last_download_timestamp 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'modify_next_last_download_timestamp',
  'TestScripts.Test.ModifyNextDownloadTimestamp' )
```

次に示すのは、サンプルの .NET メソッド ModifyNextDownloadTimestamp です。このメソッドは、ダウンロード・タイムスタンプを 1 時間前に設定します。

```
public string ModifyNextDownloadTimestamp (
  DateTime download_timestamp,
  DateTime last_download,
  string user_name ) {
  download_timestamp = download_timestamp.AddHours( -1 );
  return ( null );
}
```

modify_user 接続イベント

Mobile Link ユーザ名を指定します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。これは INOUT パラメータです。	1

デフォルトのアクション

なし

備考

Mobile Link サーバは、ユーザ名をパラメータとして指定します。このパラメータによってスクリプトが呼び出され、Mobile Link クライアントからユーザ名が送信されます。必要に応じて、代替ユーザ名を作成することもできます。このスクリプトを使用すると、Mobile Link スクリプトの呼び出しで使用するユーザ名を変更できます。

username パラメータは、ユーザ名を格納するのに十分な長さでなければなりません。

modify_user イベント用の SQL スクリプトは、ストアド・プロシージャとして実装してください。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「authenticate_user 接続イベント」 377 ページ
- 「authenticate_user_hashed 接続イベント」 382 ページ
- 「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』

SQL の例

次の例は、マッピング・テーブル user_device を使用して、リモート・データベースのユーザ名をデバイスを使用しているユーザの ID にマッピングします。この方法は、同じ人物が複数のリモート (PDA やラップトップなど) を所有し、(ユーザの名前または ID に基づいて) 同じ同期論理が必要なときに使用できます。

次の Mobile Link システム・プロシージャ・コールは、ModifyUser ストアド・プロシージャを modify_user イベントに割り当てます。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_connection_script(
 'ver1',
 'modify_user',
 'call ModifyUser( {ml s.username} )' )
```

次の SQL 文は ModifyUser ストアド・プロシージャを作成します。

```
CREATE PROCEDURE ModifyUser( INOUT u_name varchar(128) )
BEGIN
 SELECT user_name
   INTO u_name
   FROM user_device
   WHERE device_name = u_name;
END
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyUser という Java メソッドを modify_user 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'modify_user',
  'ExamplePackage.ExampleClass.modifyUser' )
```

次に示すのは、サンプルの Java メソッド modifyUser です。このメソッドは、データベースからユーザ ID を取得し、それを使用してユーザ名を設定します。

```
public String modifyUser(
  InOutString ioUserName )
  throws SQLException {
  Statement uidSelect = curConn.createStatement();
  ResultSet uidResult = uidSelect.executeQuery(
   "SELECT rep_id FROM SalesRep WHERE name = '" +
   ioUserName.getValue() + "' " );
  uidResult.next();
  ioUserName.setValue(
   java.lang.Integer.toString(uidResult.getInt( 1 ));
   uidResult.close();
   uidSelect.close();
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、ModUser という .NET メソッドを modify_user 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'modify_user',
  'TestScripts.Test.ModUser'
)
```

次に示すのは、サンプルの .NET メソッド ModUser です。

```
public string ModUser(
  string user ) {
  return ( "SELECT rep_id FROM SalesRep WHERE name = '" + user + "' " );
}
```

nonblocking_download_ack 接続イベント

非ブロッキング・ダウンロード確認を使用する場合、このスクリプトは、ダウンロードが正常に適用されたという情報を記録する場所を提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.last_download	TIMESTAMP。これは IN パラメータです。これは現在の同期の最終ダウンロード時刻です。next_last_download タイムスタンプを last_download タイムスタンプよりも前の時間に修正していないことを確認すると、重複を避けるのに役立ちます。	3

備考

このイベントを使って、リモート・データベースでダウンロードが正常に適用された時間を記録できます。

このイベントが呼び出されるのは、非ブロッキング・ダウンロード確認を使用している場合のみです。非ブロッキング・モードでは、ダウンロードが送信されたときに、ダウンロード・トランザクションがコミットされ、同期が終了します。このイベントは、同期クライアントが正常なダウンロードを確認したときに呼び出されます。このイベントは、新しい接続で、元の同期の end_synchronization スクリプトの後に呼び出されます。このイベントのアクションは、Mobile Link システム・テーブルのダウンロード時刻の更新とともにコミットされます。

このスクリプトが持つ特殊な性質によって、同期中に設定された接続レベルの変数は、このイベントを実行するときには使用できません。

参照

- 「publication_nonblocking_download_ack 接続イベント」 500 ページ
- 「-nba オプション」 76 ページ
- dbmlsync：「SendDownloadACK (sa) 拡張オプション」 『Mobile Link - クライアント管理』
- Ultra Light：「Send Download Acknowledgement 同期パラメータ」 『Ultra Light データベース管理とリファレンス』

SQL の例

次のスクリプトは、テーブル download_pubs_acked にレコードを追加します。レコードには、リモート ID、最初の認証パラメータ、ダウンロード・タイムスタンプが含まれます。

```
INSERT INTO download_pubs_acked( rem_id, auth_parm, last_download )
  VALUES( {ml s.remote_id},  {ml a.1}, {ml s.last_publication_download} )
```


prepare_for_download 接続イベント

アップロード・トランザクションとダウンロード・トランザクション間で必要な操作を処理します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.last_download	TIMESTAMP。同期テーブルの最後のダウンロード時刻。	1
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2

デフォルトのアクション

なし

備考

Mobile Link サーバは、アップロード・トランザクションからダウンロード・トランザクションの開始までの間に、このスクリプトを別個のトランザクションとして実行します。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「end_upload 接続イベント」 454 ページ
- 「begin_download 接続イベント」 388 ページ
- 「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」 134 ページ

SQL の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、prepareForDownload という SQL メソッドを prepare_for_download イベント用のスクリプトとして登録します。

```
CALL ml_add_connection_script(
  'ver1',
  'prepare_for_download',
  'CALL prepareForDownload(
     { ml s.current_time },
     { ml s.username } )' )
```

次に示すのは、サンプルの SQL メソッド prepareForDownload です。このメソッドはデータベース内のローを修正する SQL メソッドを呼び出します。

```
CREATE PROCEDURE prepareForDownload (
  IN ts TIMESTAMP,
  IN "user" VARCHAR(128))
BEGIN
  CALL adjustUploadedRows(user)
END;
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、prepareForDownload という Java メソッドを prepare_for_download イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'prepare_for_download',
  'ExamplePackage.ExampleClass.prepareForDownload' )
```

次に示すのは、サンプルの Java メソッド prepareForDownload です。このメソッドはデータベース内のローを修正する Java メソッドを呼び出します。

```
public String prepareForDownload(
  Timestamp ts,
  String user ) {
  adjustUploadedRows( _syncConn, user );
  return( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、PrepareForDownload という .NET メソッドを prepare_for_download 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
 'ver1',
 'prepare_for_download',
 'TestScripts.Test.PrepareForDownload'
)
```

次に示すのは、サンプルの .NET メソッド PrepareForDownload です。このメソッドはデータベース内のローを修正する .NET メソッドを呼び出します。


```
public string PrepareForDownload(
  DateTime timestamp,
  string user ) {
  AdjustUploadedRows ( _syncConn, user );
  return ( null );
}
```

publication_nonblocking_download_ack 接続イベント

非ブロッキング・ダウンロード確認を使用する場合、このスクリプトは、パブリケーションが正常にダウンロードされたという情報を記録する場所を提供します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	2
s.last_publication_download	TIMESTAMP。同期テーブルの最後のダウンロード時刻。	3
s.publication name	VARCHAR(128)。パブリケーションの名前。	4
s.subscription_id	VARCHAR(128)。パブリケーション ID。	5

備考

このイベントを使って、リモート・データベースでこのパブリケーションのダウンロードが正常に適用された時間を記録できます。

このイベントが呼び出されるのは、非ブロッキング・ダウンロード確認を使用している場合のみです。非ブロッキング・モードでは、ダウンロードが送信されたときに、ダウンロード・トランザクションがコミットされ、同期が終了します。このイベントは、同期クライアントが正常なダウンロードを確認したときに、ダウンロードのパブリケーションごとに 1 回呼び出されます。このイベントは、新しい接続で、元の同期の end_synchronization スクリプトの後に呼び出されま

す。このイベントのアクションは、Mobile Link システム・テーブルのダウンロード時刻の更新とともにコミットされます。

このスクリプトが持つ特殊な性質によって、同期中に設定された接続レベルの変数は、このイベントを実行するときには使用できません。

参照

- 「nonblocking_download_ack 接続イベント」 495 ページ
- 「-nba オプション」 76 ページ
- dbmlsync：「SendDownloadACK (sa) 拡張オプション」 『Mobile Link - クライアント管理』
- Ultra Light：「Send Download Acknowledgement 同期パラメータ」 『Ultra Light データベース管理とリファレンス』

SQL の例

次のスクリプトは、download_pubs_acked というテーブルにレコードを追加します。レコードには、パブリケーション名、最初の認証パラメータ、ダウンロード・タイムスタンプが含まれます。

```
INSERT INTO download_pubs_acked( pub_name, auth_parm, last_download )
  VALUES( {ml s.publication_name},  {ml a.1}, {ml s.last_publication_download} )
```

report_error 接続イベント

エラーのログを取ったり、handle_error スクリプトによって選択されたアクションを記録したりできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.action_code	INTEGER。これは INOUT パラメータです。このパラメータは必須です。	1
s.error_code	INTEGER。	2
s.error_message	TEXT。	3
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	4
s.table	VARCHAR(128)。	5

デフォルトのアクション

なし

備考

このスクリプトを使用すると、エラーのログを取ったり、handle_error スクリプトによって選択されたアクションを記録したりできます。このスクリプトは、handle_error スクリプトが定義されているかどうかに関わらず handle_error イベントの後に実行されます。また、同期接続とは異なるデータベース接続 (管理／情報接続) の専用トランザクションで常に実行されます。

エラーの内容は、エラー・コードとエラー・メッセージで識別できます。現在のエラーの原因となった SQL 操作について、エラー処理スクリプトの最後の呼び出しによってアクション・コード値が返されます。

同期の一部としてエラーが発生した場合は、ユーザ名が指定されます。それ以外の場合、この値は NULL です。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、リモート・データベースでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、同期システムの設計によって異なります。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「handle_error 接続イベント」 469 ページ
- 「handle_odbc_error 接続イベント」 473 ページ
- 「report_odbc_error 接続イベント」 505 ページ
- 「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』

SQL の例

次の例は SQL Anywhere 統合データベースで動作します。これは、同期エラーを記録するために使用されるテーブルにローを挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'report_error',
 'INSERT INTO sync_error(
   action_code,
   error_code,
   error_message,
   user_name,
   table_name )
  VALUES (
   {ml s.action_code},
   {ml s.error_code},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )'  )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、reportError という Java メソッドを report_error 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'report_error',
   'ExamplePackage.ExampleClass.reportError' )
```

次に示すのは、サンプルの Java メソッド reportError です。このメソッドは、Mobile Link が提供する JDBC 接続を使用してテーブルにエラーのログを取ります。また、アクション・コードも設定します。

```
public String reportError(
  ianywhere.ml.script.InOutInteger actionCode,
  int errorCode,
```

```
  String errorMessage,
  String user,
  String table )
  throws java.sql.SQLException {
   // Insert error information in a table,
   JDBCLogError( _syncConn, errorCode, errorMessage,
    user, table );
   actionCode.setValue( getActionCode( errorCode ) );
   return( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、ReportError という .NET メソッドを report_error 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'report_error',
  'TestScripts.Test.ReportError' )
```

次に示すのは、サンプルの .NET メソッド ReportError です。このメソッドは、.NET メソッドを使用してテーブルにエラーのログを取ります。

```
public string ReportError(
  ref int actionCode,
  int errorCode,
  string errorMessage,
  string user,
  string table ) {
  LogError(_syncConn, errorCode, errorMessage, user, table);
}
```


report_odbc_error 接続イベント

エラーのログを取ったり、handle_odbc_error スクリプトによって選択されたアクションを記録したりできます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.action_code	INTEGER。これは INOUT パラメータです。このパラメータは必須です。	1
s.ODBC_state	VARCHAR(5)。	2
s.error_message	TEXT。	3
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	4
s.table	VARCHAR(128)。	5

デフォルトのアクション

なし

備考

このスクリプトを使用すると、エラーのログを取ったり、handle_odbc_error スクリプトによって選択されたアクションを記録したりできます。このスクリプトは、handle_odbc_error スクリプトが定義されているかどうかに関わらず handle_odbc_error イベントの後に実行されます。また、同期接続とは異なるデータベース接続 (管理／情報接続) の専用トランザクションで常に実行されます。

エラーの内容は、エラー・コードとエラー・メッセージで識別できます。現在のエラーの原因となった SQL 操作について、エラー処理スクリプトの最後の呼び出しによってアクション・コード値が返されます。

同期の一部としてエラーが発生した場合は、ユーザ名が指定されます。それ以外の場合、この値は NULL です。

特定のテーブルの操作中にエラーが発生した場合は、テーブル名が指定されます。それ以外の場合、この値は NULL です。テーブル名は、リモート・データベースでのテーブル名です。この名前に直接対応するものが統合データベース内にあるかどうかは、同期システムの設計によって異なります。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「handle_error 接続イベント」 469 ページ
- 「handle_odbc_error 接続イベント」 473 ページ
- 「report_error 接続イベント」 502 ページ
- 「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』

SQL の例

次の例は SQL Anywhere 統合データベースで動作します。これは、同期エラーを記録するために使用されるテーブルにローを挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'report_odbc_error',
 'INSERT INTO sync_error(
    action_code,
    odbc_state,
    error_message,
    user_name,
    table_name )
  VALUES(
   {ml s.action_code},
   {ml s.ODBC_state},
   {ml s.error_message},
   {ml s.username},
   {ml s.table} )' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、reportODBCError という Java メソッドを report_odbc_error イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'report_odbc_error',
   'ExamplePackage.ExampleClass.reportODBCError' )
```

次に示すのは、サンプルの Java メソッド reportODBCError です。このメソッドは、Mobile Link が提供する JDBC 接続を使用してテーブルにエラーのログを取ります。また、アクション・コードも設定します。

```
public String reportODBCError(
  ianywhere.ml.script.InOutInteger actionCode,
  String ODBCState,
```



```
  String errorMessage,
  String user,
  String table )
  throws java.sql.SQLException {
  JDBCLogError( _syncConn, ODBCState, errorMessage,
   user, table );
  actionCode.setValue( getActionCode( ODBCState ) );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、ReportODBCError という .NET メソッドを report_odbc_error イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'report_odbc_error',
  'TestScripts.Test.ReportODBCError' )
```

次に示すのは、サンプルの .NET メソッド ReportODBCError です。このメソッドは、.NET メソッドを使用してテーブルにエラーのログを取ります。

```
public string ReportODBCError (
  ref int actionCode,
  string ODBCState,
  string errorMessage,
  string user,
  string table ) {
  LogError(_syncConn, ODBCState, errorMessage, user, table);
  return ( null );
}
```

resolve_conflict テーブル・イベント

特定のテーブルの競合を解決する処理を定義します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2

デフォルトのアクション

なし

備考

リモート・データベースでローが更新されると、Mobile Link クライアントは元の値のコピーを保存します。クライアントは、Mobile Link サーバに古い値と新しい値の両方を送信します。

Mobile Link サーバは更新されたローを受信すると、元の値と統合データベース内の現在の値を比較します。比較は、upload_fetch スクリプトを使用して行われます。

アップロードされた古い値が統合データベース内の現在の値と一致しない場合は、そのローに競合が発生します。ローを更新する代わりに、Mobile Link サーバは古い値と新しい値の両方を統合データベースに挿入します。古いローと新しいローは、それぞれスクリプト upload_old_row_insert と upload_new_row_insert を使用して処理されます。

値が挿入されると、Mobile Link サーバは resolve_conflict スクリプトを実行します。ここで、競合を解決できます。どのスキームでも選択して実装できます。

このスクリプトは競合ごとに 1 回実行されます。

別の方法として、resolve_conflict スクリプトを定義する代わりに、end_upload_rows スクリプトまたは end_upload テーブル・スクリプトで競合解決論理を使用して、集合指向型の方法で競合を解決することもできます。

リモート・データベースのテーブルごとに、resolve_conflict スクリプトを 1 つ指定できます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「upload_old_row_insert テーブル・イベント」 535 ページ
- 「upload_new_row_insert テーブル・イベント」 532 ページ
- 「upload_update テーブル・イベント」 548 ページ
- 「end_upload_rows テーブル・イベント」 462 ページ
- 「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』

SQL の例

次の文は、Oracle インストール環境用の CustDB サンプル・アプリケーションに適した resolve_conflict スクリプトを定義します。このスクリプトは、ストアド・プロシージャ ULResolveOrderConflict を呼び出します。

```
exec ml_add_table_script(
  'custdb', 'ULOrder', 'resolve_conflict',
  'begin ULResolveOrderConflict();
end; ')
CREATE OR REPLACE PROCEDURE ULResolveOrderConflict()
AS
  new_order_id integer;
  new_status   varchar(20);
  new_notes    varchar(50);
BEGIN
  -- approval overrides denial
  SELECT order_id, status, notes
    INTO new_order_id, new_status, new_notes
    FROM ULNewOrder
   WHERE syncuser_id = SyncUserID;
  IF new_status = 'Approved' THEN
    UPDATE ULOrder o
       SET o.status = new_status, o.notes =
         new_notes
     WHERE o.order_id = new_order_id;
  END IF;
  DELETE FROM ULOldOrder;
  DELETE FROM ULNewOrder;
END;
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、resolveConflict という Java メソッドを resolve_conflict テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
```

```
    'resolve_conflict',
    'ExamplePackage.ExampleClass.resolveConflict' )
```

次に示すのは、サンプルの Java メソッド resolveConflict です。このメソッドは、競合を解決するために Mobile Link が提供する JDBC 接続を使用する Java メソッドを呼び出します。

```
public String resolveConflict(
  String user,
  String table) {
  resolveRows(_syncConn, user );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、ResolveConflict という .NET メソッドを resolve_conflict テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
   'ver1',
   'table1',
   'resolve_conflict',
   'TestScripts.Test.ResolveConflict' )
```

次に示すのは、サンプルの .NET メソッド ResolveConflict です。このメソッドは、競合を解決する .NET メソッドを呼び出します。

```
public string ResolveConflict(
  String user,
  String table) {
  ResolveRows(_syncConn, user );
}
```


synchronization_statistics 接続イベント

同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.warnings	INTEGER。同期中に発行された警告の数。	2
s.errors	INTEGER。同期で発生したエラーの数。	3
s.deadlocks	INTEGER。同期で検出された統合データベース内のデッドロックの数。	4
s.synchronized_tables	INTEGER。同期に関係したクライアント・テーブルの数。	5
s.connection_retries	INTEGER。Mobile Link サーバが統合データベースとの接続をリトライした回数。	6

デフォルトのアクション

なし

備考

synchronization_statistics イベントを使用すると、任意のユーザと接続について、現在の同期に関する各種の統計を収集できます。最後の同期トランザクション終了時のコミット直前に、synchronization_statistics 接続スクリプトが呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「download_statistics 接続イベント」 425 ページ
- 「download_statistics テーブル・イベント」 428 ページ
- 「upload_statistics 接続イベント」 538 ページ
- 「upload_statistics テーブル・イベント」 543 ページ
- 「synchronization_statistics テーブル・イベント」 514 ページ
- 「time_statistics 接続イベント」 517 ページ
- 「time_statistics テーブル・イベント」 520 ページ
- 「Mobile Link モニタ」 183 ページ
- 「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』

SQL の例

次の例は、同期の統計を sync_con_audit テーブルに挿入します。

```
CALL ml_add_connection_script(
  'ver1',
  'synchronization_statistics',
 'INSERT INTO sync_con_audit(
   ml_user,
   warnings,
   errors,
   deadlocks,
   synchronized_tables,
   connection_retries)
  VALUES (
   {ml s.username},
   {ml s.warnings},
   {ml s.errors},
   {ml s.deadlocks},
   {ml s.synchronized_tables},
   {ml s.connection_retries})' )
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、synchronizationStatisticsConnection という Java メソッドを synchronization_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'synchronization_statistics',
  'ExamplePackage.ExampleClass.synchronizationStatisticsConnection'
)
```


次に示すのは、サンプルの Java メソッド synchronizationStatisticsConnection です。このメソッドは、Mobile Link メッセージ・ログに統計の一部を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String synchronizationStatisticsConnection(
  String user,
  int warnings,
  int errors,
  int deadlocks,
  int synchronizedTables,
  int connectionRetries ) {
  java.lang.System.out.println(
   "synch statistics number of deadlocks: "
   + deadlocks ;
  return( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、SyncStats という .NET メソッドを synchronization_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'synchronization_statistics',
  'TestScripts.Test.SyncStats'
)
```

次に示すのは、サンプルの .NET メソッド SyncStats です。このメソッドは、Mobile Link メッセージ・ログに統計の一部を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string SyncStats(
  string user,
  int warnings,
  int errors,
  int deadLocks,
  int syncedTables,
  int connRetries ) {
  System.Console.WriteLine( "synch statistics
   number of deadlocks: " + deadlocks ;
  return( null );
}
```

synchronization_statistics テーブル・イベント

同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.warnings	INTEGER。同期中にテーブルに対して発生した警告の数。	3
s.errors	INTEGER。同期中に発生したテーブルに関係するエラーの数。	4

デフォルトのアクション

なし

備考

synchronization_statistics イベントを使用すると、任意のユーザとテーブルについて、同期中に発生した警告とエラーの数を収集できます。最後の同期トランザクション終了時のコミット直前に、synchronization_statistics テーブル・スクリプトが呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「download_statistics 接続イベント」 425 ページ
- 「download_statistics テーブル・イベント」 428 ページ
- 「upload_statistics 接続イベント」 538 ページ
- 「upload_statistics テーブル・イベント」 543 ページ
- 「synchronization_statistics 接続イベント」 511 ページ
- 「time_statistics 接続イベント」 517 ページ
- 「time_statistics テーブル・イベント」 520 ページ
- 「Mobile Link モニタ」 183 ページ
- 「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』

SQL の例

次の例は、同期の統計を sync_tab_audit テーブルに挿入します。

```
CALL ml_add_table_script(
  'ver1',
  'table1',
 'upload_insert',
  'INSERT INTO sync_tab_audit (
    ml_user,
    table,
    warnings,
    errors)
   VALUES (
    {ml s.username},
    {ml s.table},
    {ml s.warnings},
    {ml s.errors} ) ' )
```

監査テーブルに同期統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、synchronizationStatisticsTable という Java メソッドを synchronization_statistics テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'synchronization_statistics',
  'ExamplePackage.ExampleClass.synchronizationStatisticsTable'
)
```

次に示すのは、サンプルの Java メソッド synchronizationStatisticsTable です。このメソッドは、Mobile Link メッセージ・ログに統計の一部を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String synchronizationStatisticsTable(
  String user,
  String table,
```

```
  int warnings,
  int errors ) {
  java.lang.System.out.println( "synch statistics for
   table: " + table + " errors: " + errors );
  return( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、SyncTableStats という .NET メソッドを synchronization_statistics テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'synchronization_statistics',
  'TestScripts.Test.SyncTableStats'
)
```

次に示すのは、サンプルの .NET メソッド SyncTableStats です。このメソッドは、Mobile Link メッセージ・ログに統計の一部を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string SyncTableStats(
  string user,
  string table,
  int warnings,
  int errors ) {
  System.Console.WriteLine( "synch statistics for
   table: " + table + " errors: " + errors );
  return( null );
}
```


time_statistics 接続イベント

ユーザ別、イベント別の時間統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.event_name	VARCHAR(128)	2
s.num_of_calls	INTEGER。スクリプトが呼び出された回数。	3
s.minimum_time	INTEGER。ミリ秒。この同期中にスクリプトを実行するのにかかった最短の時間。	4
s.maximum_time	INTEGER。ミリ秒。この同期中にスクリプトを実行するのにかかった最長の時間。	5
s.total_time	INTEGER。ミリ秒。同期ですべてのスクリプトを実行するのにかかった合計時間(これは同期の長さとは異なります)。	6

デフォルトのアクション

なし

備考

time_statistics イベントを使用すると、任意のユーザについて同期中の時間統計を収集できます。対応するスクリプトがあるイベントについてのみ、統計が収集されます。単一のイベントが数回発生する場合、スクリプトは集合データを収集します。このスクリプトは、ユーザ、イベント、テーブル間で時間を比較する場合に特に便利です。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「time_statistics テーブル・イベント」 520 ページ
- 「download_statistics 接続イベント」 425 ページ
- 「download_statistics テーブル・イベント」 428 ページ
- 「upload_statistics 接続イベント」 538 ページ
- 「upload_statistics テーブル・イベント」 543 ページ
- 「synchronization_statistics 接続イベント」 511 ページ
- 「synchronization_statistics テーブル・イベント」 514 ページ
- 「Mobile Link モニタ」 183 ページ
- 「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』

SQL の例

次の例は、統計情報を time_statistics テーブルに挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'time_statistics',
 'INSERT INTO time_statistics (
   id,
   ml_user,
   table,
   event_name,
   number_of_calls,
   minimum_time,
   maximum_time,
   total_time)
  VALUES (
   ts_id.nextval,
   {ml s.username},
   {ml s.event_name},
   {ml s.number_of_calls},
   {ml s.minimum_time},
   {ml s.maximum_time},
   {ml s.total_time} ) ' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、timeStatisticsConnection という Java メソッドを time_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
 'ver1',
 'time_statistics',
 'ExamplePackage.ExampleClass.timeStatisticsConnection' )
```


次に示すのは、サンプルの Java メソッド timeStatisticsConnection です。このメソッドは、prepare_for_download イベントの統計を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String timeStatisticsConnection(
  String username,
  String tableName,
  String eventName,
  int numberOfCalls,
  int minimumTime,
  int maximumTime,
  int totalTime ) {
  if( eventName.equals( "prepare_for_download") ) {
    java.lang.System.out.println(
      "prepare_for_download num_calls: " + numCalls +
      "total_time: " + totalTime );
  }
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、TimeStats という .NET メソッドを time_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'time_statistics',
  'TestScripts.Test.TimeStats'
)
```

次に示すのは、サンプルの .NET メソッド TimeStats です。このメソッドは、prepare_for_download イベントの統計を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string TimeStats(
  string user,
  string eventName,
  int numberOfCalls,
  int minimumTime,
  int maximumTime,
  int totTime ) {
  if( event_name=="prepare_for_download") {
    System.Console.WriteLine(
      "prepare_for_download num_calls: " + num_calls +
      "total_time: " + total_time );
  }
  return ( null );
}
```

time_statistics テーブル・イベント

時間統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.event_name	VARCHAR(128)	3
s.number_of_calls	INTEGER。スクリプトが呼び出された回数。	4
s.minimum_time	INTEGER。ミリ秒。このテーブルの同期中にスクリプトを実行するのにかかった最短の時間。	5
s.maximum_time	INTEGER。ミリ秒。このテーブルの同期中にスクリプトを実行するのにかかった最長の時間。	6
s.total_time	INTEGER。ミリ秒。テーブルの同期ですべてのスクリプトを実行するのにかかった合計時間(これは同期の長さとは異なります)。	7

デフォルトのアクション

なし

備考

time_statistics テーブル・イベントを使用すると、任意のユーザとテーブルについて同期中の時間統計を収集できます。対応するスクリプトがあるイベントについてのみ、統計が収集されます。単一のイベントが数回発生する場合、スクリプトは集合データを収集します。このスクリプトは、ユーザ、イベント、テーブル間で時間を比較する場合に特に便利です。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「time_statistics 接続イベント」 517 ページ
- 「download_statistics 接続イベント」 425 ページ
- 「download_statistics テーブル・イベント」 428 ページ
- 「upload_statistics 接続イベント」 538 ページ
- 「upload_statistics テーブル・イベント」 543 ページ
- 「synchronization_statistics 接続イベント」 511 ページ
- 「synchronization_statistics テーブル・イベント」 514 ページ
- 「Mobile Link モニタ」 183 ページ
- 「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』

SQL の例

次の例は、統計情報を time_statistics テーブルに挿入します。

```
CALL ml_add_table_script (
 'ver1',
 'table1',
 'time_statistics',
 'INSERT INTO time_statistics(
   ml_user,
   table,
   event_name,
   number_of_calls,
   minimum_time,
   maximum_time,
   total_time)
  VALUES (
   {ml s.username},
   {ml s.table},
   {ml s.event_name},
   {ml s.number_of_calls},
   {ml s.minimum_time},
   {ml s.maximum_time},
   {ml s.total_time} )' );
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、timeStatisticsTable という Java メソッドを time_statistics テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'time_statistics',
  'ExamplePackage.ExampleClass.timeStatisticsTable' )
```

次に示すのは、サンプルの Java メソッド timeStatisticsTable です。このメソッドは、upload_old_row_insert イベントの統計を出力します。

```
public String timeStatisticsTable(
  String username,
  String tableName,
  String eventName,
  int numberOfCalls,
  int minimumTime,
  int maximumTime,
  int totalTime ) {
  if( eventName.equals( "upload_old_row_insert") ) {
    java.lang.System.out.println(
     "upload_old_row_insert num_calls: " + numCalls +
     "total_time: " + totalTime );
  }
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、TimeTableStats という .NET メソッドを time_statistics テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'time_statistics',
  'TestScripts.Test.TimeTableStats'
)
```

次に示すのは、サンプルの .NET メソッド TimeTableStats です。このメソッドは、upload_old_row_insert イベントの統計を出力します。

```
public string TimeTableStats(
  string user,
  string table,
  string eventName,
  int numberOfCalls,
  int minimumTime,
  int maximumTime,
  int totTime ) {
  if( event_name == "upload_old_row_insert") {
    System.Console.WriteLine(
      "upload_old_row_insert num_calls: " + num_calls +
      "total_time: " + total_time );
  }
  return ( null );
}
```

upload_delete テーブル・イベント

リモート・データベースから削除されたローを処理するために、Mobile Link サーバがアップロード処理中に使用するイベントを提供します。

パラメータ

SQL スクリプトのパラメータ名	順序
r.*pk-column-1*	1
...	...
r.*pk-column-N*	*N*
r.*column-1*	*N* + 1
...	...
r.*column-M*	*N* + *M*

デフォルトのアクション

なし

備考

文ベースの upload_delete スクリプトは、リモート・データベースで削除されるローを処理します。統合データベース側で実行される動作として DELETE 文を指定できますが、他の動作も指定できます。

リモート・データベースのテーブルごとに、upload_delete スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「upload_insert テーブル・イベント」 530 ページ
- 「upload_update テーブル・イベント」 548 ページ

SQL の例

この例は Contact の例から抜粋したもので、*Samples¥MobiLink¥Contact¥build_consol.sql* にあります。リモート・データベースから削除される顧客に、非アクティブのマークを付けます。

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_delete',
  'UPDATE Customer
```



```
  SET active = 0
  WHERE cust_id={ml r.cust_id}' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadDeleteTable という Java メソッドを upload_delete テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'upload_delete',
  'ExamplePackage.ExampleClass.uploadDeleteTable' )
```

次に示すのは、サンプルの Java メソッド uploadDeleteTable です。このメソッドは、UPLOAD 文を動的に生成する genUD を呼び出します。

```
public String uploadDeleteTable() {
  return( genUD(_curTable) );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、UploadDelete という .NET メソッドを upload_delete テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'upload_delete',
  'TestScripts.Test.UploadDelete'
)
```

次に示すのは、サンプルの .NET メソッド UploadDelete です。このメソッドは、UPLOAD 文を動的に生成する genUD を呼び出します。

```
public string UploadDelete( object pk1 ) {
  return( genUD(_curTable) );
}
```


upload_fetch テーブル・イベント

ローレベルの競合検出の目的で、統合データベース内の同期テーブルからローをフェッチします。

パラメータ

SQL スクリプトのパラメータ名	順序
r.*primary-key-1*	1
r.*primary-key-2*	2
...	...
r.*primary-key-N*	N

デフォルトのアクション

なし

備考

文ベースの upload_fetch スクリプトは、競合検出の目的で、同期テーブルからローをフェッチします。このスクリプトは、upload_update イベントに対応します。

結果セットのカラム数は、このテーブルについてリモート・データベースからアップロードされるカラムの数と一致します。返される値がアップロードされるローの更新前のイメージと一致しないと、競合が識別されます。

upload_fetch スクリプトでは READPAST テーブル・ヒントを使用しないでください。スクリプトが READPAST を使用してロックされたローをスキップした場合、同期論理は、そのローが削除されたものとみなします。これにより、定義したスクリプトに応じて、アップロードされた更新が無視されるか、または競合解決がトリガされます。更新の無視は、許容されない動作であることが多く、問題になる場合があります。実装している解決論理によっては、競合解決がトリガされても問題にならない場合があります。

リモート・データベースのテーブルごとに、upload_fetch または upload_fetch_column_conflict スクリプトを 1 つのみ指定できます。

以下のスクリプトが 1 つも定義されていない場合、このスクリプトは無視されます。
upload_new_row_insert、upload_old_row_insert、resolve_conflict

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「競合の検出」 151 ページ
- 「resolve_conflict テーブル・イベント」 508 ページ
- 「upload_delete テーブル・イベント」 523 ページ
- 「upload_insert テーブル・イベント」 530 ページ
- 「upload_update テーブル・イベント」 548 ページ
- 「FROM 句」 『SQL Anywhere サーバ - SQL リファレンス』の「Mobile Link 同期で READPAST を使用」

SQL の例

次の SQL スクリプトは、Contact の例から抜粋したもので、*samples-dir¥MobiLink¥Contact ¥build_consol.sql* にあります。リモート・データベースの Product テーブル内で更新されるローのアップロード時に発生する競合を識別するために使用されます。このスクリプトは、テーブル Product からローを選択しますが、統合データベースとリモート・データベースのスキーマによっては、2 つのテーブルの名前が一致しない場合があります。

```
CALL ml_add_table_script(
 'ver1',
 'Product',
 'upload_fetch',
 'SELECT id, name, size, quantity, unit_price
  FROM Product
  WHERE id={ml r.id}' )
```

Java の例

このスクリプトは有効な SQL を返します。

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadFetchTable という Java メソッドを upload_fetch テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
 'ver1',
 'Product',
 'upload_fetch',
 'ExamplePackage.ExampleClass.uploadFetchTable' )
```

次に示すのは、サンプルの Java メソッド uploadFetchTable です。このメソッドは、UPLOAD 文を動的に生成する genUF を呼び出します。

```
public String uploadFetchTable() {
  return( genUF(_curTable) );
}
```

.NET の例

このスクリプトは有効な SQL を返します。

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、UploadFetchTable という .NET メソッドを upload_fetch テーブル・イベント用のスクリプトとして登録します。


```
CALL ml_add_dnet_table_script(
  'ver1',
  'Product',
  'upload_fetch',
  'TestScripts.Test.UploadFetchTable' )
```

次に示すのは、サンプルの .NET メソッド UploadFetchTable です。このメソッドは、UPLOAD 文を動的に生成する GenUF を呼び出します。

```
public string UploadFetchTable() {
  return( GenUF(_curTable) );
}
```

upload_fetch_column_conflict テーブル・イベント

カラムレベルの競合検出の目的で、統合データベース内の同期テーブルからローをフェッチします。

パラメータ

SQL スクリプトのパラメータ名	順序
r.*pk-column-1*	1
...	...
r.*pk-column-N*	*N*
r.*column-1*	*N* + 1
...	...
r.*column-M*	*N* + *M*

デフォルトのアクション

なし

備考

文ベースの upload_fetch_column_conflict スクリプトは、競合検出の目的で、同期テーブルからカラムをフェッチします。このスクリプトは、upload_update イベントに対応します。

このスクリプトは、2 人のユーザが同じカラムを更新する場合のみ競合を検出します。異なるユーザは、同じカラムを更新しないかぎり、同じローを更新することができ、競合は発生しません。

たとえば、upload_fetch_column_conflict スクリプトを使用すると、一方のリモート・ユーザが ULOrder テーブルの quant カラムを更新し、もう 1 人のリモート・ユーザが同じローの notes ローを更新した場合に競合が検出されないようにできます。両方のユーザが quant カラムを更新した場合のみ、競合が検出されます。

リモート・データベースのテーブルごとに、upload_fetch または upload_fetch_column_conflict スクリプトを 1 つのみ指定できます。

以下のスクリプトが 1 つも定義されていない場合、このスクリプトは無視されます。
upload_new_row_insert、upload_old_row_insert、resolve_conflict

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「競合の検出」 151 ページ
- 「upload_fetch テーブル・イベント」 525 ページ
- 「resolve_conflict テーブル・イベント」 508 ページ
- 「upload_delete テーブル・イベント」 523 ページ
- 「upload_insert テーブル・イベント」 530 ページ
- 「upload_update テーブル・イベント」 548 ページ

upload_insert テーブル・イベント

リモート・データベースに挿入されたローを処理するために、Mobile Link サーバがアップロード処理中に使用するイベントを提供します。

パラメータ

SQL スクリプトのパラメータ名	順序
r.*pk-column-1*	1
...	...
r.*pk-column-N*	*N*
r.*column-1*	*N*+1
...	...
r.*column-M*	*N* + *M*

デフォルトのアクション

なし

備考

文ベースの upload_insert スクリプトは、カラム値の直接挿入を実行します。

リモート・データベースのテーブルごとに、upload_insert スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「upload_delete テーブル・イベント」 523 ページ
- 「upload_update テーブル・イベント」 548 ページ
- 「upload_fetch テーブル・イベント」 525 ページ

SQL の例

この例では、リモート・データベース内の Customer テーブルに対して行われた挿入を処理します。このスクリプトは、統合データベース内のテーブル Customer に値を挿入します。このテーブルの最後のカラムでは、Customer がアクティブであると識別されます。最後のカラムは、リモート・データベースには含まれません。

```
CALL ml_add_table_script(
 'ver1',
 'Customer',
 'upload_insert',
```



```
'INSERT INTO Customer(
  cust_id,
  name,
  rep_id,
  active )
 VALUES (
  {ml r.cust_id},
  {ml r.name},
  {ml r.rep_id},
  1 )' );
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadInsertTable という Java メソッドを upload_insert テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'upload_insert',
  'ExamplePackage.ExampleClass.uploadInsertTable' )
```

次に示すのは、サンプルの Java メソッド uploadInsertTable です。このメソッドは、INSERT 文を動的に生成します。これは SQL Anywhere 統合データベース用の構文です。

```
public String uploadInsertTable() {
  return("INSERT INTO " + _curTable + getCols(_curTable)
    + " VALUES " + getQM(_curTable));
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、UploadInsert という .NET メソッドを upload_insert テーブル・イベント用のスクリプトとして登録します。これは SQL Anywhere 統合データベース用の構文です。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'upload_insert',
  'TestScripts.Test.UploadInsert'
)
```

次に示すのは、サンプルの .NET メソッド UploadInsert です。このメソッドは、ULCustomer テーブル用の INSERT 文を返します。

```
public static string UploadInsert() {
   return("INSERT INTO ULCustomer( cust_id, cust_name ) VALUES ( {ml r.cust_id}, {ml
r.cust_name} )");
 }
```

upload_new_row_insert テーブル・イベント

通常、文ベースのアップロード用の競合解決スクリプトは、リモート・データベースからアップロードされるローの古い値と新しい値にアクセスする必要があります。このイベントを使用すると、リモート・データベースからアップロードされるローの更新済みの新しい値を処理できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。このパラメータは省略可能です。	省略可
r.*pk-column-1*	古い (更新前イメージ) ローからのカラム値。名前付きパラメータは、**r.** を前に付けたカラム名として指定されます。	1 (username を参照する場合は 2)
...		...
r.*pk-column-N*	古い (更新前イメージ) ローからのカラム値。名前付きパラメータは、**r.** を前に付けたカラム名として指定されます。	*N* (username を参照する場合は *N*+1)
r.*column-1*	古い (更新前イメージ) ローからのカラム値。名前付きパラメータは、**r.** を前に付けたカラム名として指定されます。	*N* + 1 (username を参照する場合は *N*+2)
...	...	...
r.*column-M*	古い (更新前イメージ) ローからのカラム値。名前付きパラメータは、**r.** を前に付けたカラム名として指定されます。	*N* + *M* (username を参照する場合は *N*+*M* +1)

デフォルトのアクション

なし

備考

Mobile Link クライアントが更新済みのローを Mobile Link サーバに送信するときは、新しい値 (更新後イメージ) だけでなく、古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

このイベントを使用すると、更新後イメージの値をテーブルに保存できます。このイベントは、文ベースの更新用の競合解決プロシージャ開発を支援するために使用できます。このイベントのパラメータは、対応する統合データベース・テーブルで更新が実行される前のリモート・データベースからの新しい値を保持しています。また、文ベースの強制的な競合モードで、ローを挿入するために使用されます。

このイベントのスクリプトは、resolve_conflict スクリプトが使用するテンポラリ・テーブルに新しいローを挿入する INSERT 文である場合がほとんどです。

リモート・データベースのテーブルごとに、upload_new_row_insert スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「競合の解決」 150 ページ
- 「resolve_conflict テーブル・イベント」 508 ページ
- 「upload_old_row_insert テーブル・イベント」 535 ページ
- 「upload_update テーブル・イベント」 548 ページ
- 「強制的な競合解決」 159 ページ
- 「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』

SQL の例

この例は、リモート・データベース内の product テーブルに対する更新を処理します。このスクリプトは、ローの新しい値をグローバルなテンポラリ・テーブル product_conflict に挿入します。このテーブルの最後のカラムでは、ローが新しいローとして識別されます。

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'upload_new_row_insert',
 'INSERT INTO DBA.product_conflict(
   id,
   name,
   size,
   quantity,
   unit_price,
   row_type )
  VALUES(
   {ml r.id},
```

```
    {ml r.name},
    {ml r.size},
    {ml r.quantity},
    {ml r.unit_price},
    "New" )' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadNewRowInsertTable という Java メソッドを upload_new_row_insert テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'upload_new_row_insert',
  'ExamplePackage.ExampleClass.uploadNewRowInsertTable'
)
```

次に示すのは、サンプルの Java メソッド uploadNewRowInsertTable です。このメソッドは、INSERT 文を動的に生成します。これは SQL Anywhere 統合データベース用の構文です。

```
public String uploadNewRowInsertTable() {
  return("insert into" + _curTable + "_new" +
    getCols(_curTable) + "values" + getNamedParams(_curTable));
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、UploadNewRowInsertTable という .NET メソッドを upload_new_row_insert テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'upload_new_row_insert',
  'TestScripts.Test.UploadNewRowInsertTable'
)
```

次に示すのは、サンプルの .NET メソッド UploadNewRowInsertTable です。このメソッドは、INSERT 文を動的に生成します。これは SQL Anywhere 統合データベース用の構文です。

```
public string UploadNewRowInsertTable() {
  return("insert into" + _curTable + "_new" +
    GetCols(_curTable) + "values" + GetNamedParams(_curTable));
}
```


upload_old_row_insert テーブル・イベント

通常、文ベースのアップロード用の競合解決スクリプトは、リモート・データベースからアップロードされるローの古い値と新しい値にアクセスする必要があります。このイベントを使用すると、リモート・データベースからアップロードされるローの古い値を処理できます。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。このパラメータは省略可能です。	省略可
r.*pk-column-1*	古い (更新前イメージ) ローからのカラム値。名前付きパラメータは、**r.** を前に付けたカラム名として指定されます。	1 (username を参照する場合は 2)
...		...
r.*pk-column-N*	古い (更新前イメージ) ローからのカラム値。名前付きパラメータは、**r.** を前に付けたカラム名として指定されます。	*N* (username を参照する場合は *N*+1)
r.*column-1*	古い (更新前イメージ) ローからのカラム値。名前付きパラメータは、**r.** を前に付けたカラム名として指定されます。	*N* + 1 (username を参照する場合は *N*+2)
...	...	...
r.*column-M*	古い (更新前イメージ) ローからのカラム値。名前付きパラメータは、**r.** を前に付けたカラム名として指定されます。	*N* + *M* (username を参照する場合は *N*+*M* +1)

デフォルトのアクション

なし

備考

Mobile Link クライアントが更新済みのローを Mobile Link サーバに送信するときは、新しい値 (更新後イメージ) だけでなく、古いローの値 (更新前イメージ) のコピーも含まれています。更新前イメージが統合データベースの現在の値と一致しないと、競合が検出されます。

このイベントを使用すると、更新前イメージの値をテーブルに保存できます。このイベントは、文ベースの更新用の競合解決プロシージャ開発を支援するために使用できます。このイベントのパラメータは、対応する統合データベース・テーブルで更新が実行される前のリモート・データベースからの古い値を保持しています。また、文ベースの強制的な競合モードで、ローを挿入するために使用されます。

このイベントのスクリプトは、resolve_conflict スクリプトが使用するテンポラリ・テーブルに古いローを挿入する INSERT 文である場合がほとんどです。

リモート・データベースのテーブルごとに、upload_old_row_insert スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「競合の解決」 150 ページ
- 「resolve_conflict テーブル・イベント」 508 ページ
- 「upload_new_row_insert テーブル・イベント」 532 ページ
- 「upload_update テーブル・イベント」 548 ページ
- 「強制的な競合解決」 159 ページ
- 「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』

SQL の例

この例は、リモート・データベース内の product テーブルに対する更新を処理します。このスクリプトは、ローの古い値をグローバルなテンポラリ・テーブル product_conflict に挿入します。このテーブルの最後のカラムでは、ローが古いローとして識別されます。

```
CALL ml_add_table_script(
 'ver1',
 'table1',
 'upload_old_row_insert',
 'INSERT INTO DBA.product_conflict (
   id,
   name,
   size,
   quantity,
   unit_price,
   row_type )
  VALUES (
   {ml r.id},
```



```
  {ml r.name},
  {ml r.size},
  {ml r.quantity},
  {ml r.unit_price},
  "Old" )' )
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadOldRowInsertTable という Java メソッドを upload_old_row_insert テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'upload_old_row_insert',
  'ExamplePackage.ExampleClass.uploadOldRowInsertTable'
)
```

次に示すのは、サンプルの Java メソッド uploadOldRowInsertTable です。このメソッドは、INSERT 文を動的に生成します。

```
public String uploadOldRowInsertTable() {
  return( "old" + getCols(_curTable) +
    "values" + getNamedParams(_curTable));
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、UploadOldRowInsertTable という .NET メソッドを upload_old_row_insert テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'upload_old_row_insert',
  'TestScripts.Test.UploadOldRowInsertTable'
)
```

次に示すのは、サンプルの .NET メソッド UploadOldRowInsertTable です。このメソッドは、UPLOAD 文を動的に生成します。

```
public string UploadOldRowInsertTable() {
  return( "old" + GetCols(_curTable) +
    "values" + GetNamedParams(_curTable));
}
```

upload_statistics 接続イベント

アップロード操作の同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.warnings	INTEGER。発生した警告の数。	2
s.errors	INTEGER。発生したエラーの数。	3
s.inserted_rows	INTEGER。統合データベースに正常に挿入されたローの数。	4
s.deleted_rows	INTEGER。統合データベースから正常に削除されたローの数。	5
s.updated_rows	INTEGER。統合データベースで正常に更新されたローの数。	6
s.conflicted_inserts	INTEGER。常に 0 (ゼロ) です。	7
s.conflicted_deletes	INTEGER。常に 0 (ゼロ) です。	8
s.conflicted_updates	INTEGER。競合を引き起こした更新ローの数。対処する競合解決スクリプトが正常に呼び出された場合のみ、ローは含まれます。	9

SQL スクリプトのパラメータ名	説明	順序
s.ignored_inserts	INTEGER。無視されたアップロード挿入ローの合計数。無視される理由は次のとおりです。1) 通常モードで upload_insert スクリプトないか、または強制的な競合モードで upload_new_row_insert スクリプトがない。2) Mobile Link サーバが対応するスクリプトを呼び出し中にエラーが発生し、handle_error または handle_odbc_error イベントが 1000 を返した。	10
s.ignored_deletes	INTEGER。upload_delete スクリプトを呼び出したとき、handle_error または handle_odbc_error が定義され、1000 を返した場合、または指定のテーブルに対して upload_delete スクリプトが定義されていない場合にエラーを発生したアップロード削除ローの数。	11
s.ignored_updates	INTEGER。競合を引き起こしたが、競合解決スクリプトが正常に呼び出されなかったり、upload_update スクリプトが定義されていなかったアップロード更新ローの数。	12
s.bytes	INTEGER。アップロードを保存するために Mobile Link サーバ内で使用されるメモリの容量。	13
s.deadlocks	INTEGER。同期で検出された統合データベース内のデッドロックの数。	14

デフォルトのアクション

なし

備考

upload_statistics イベントを使用すると、任意のユーザについてアップロードに関する統計を収集できます。アップロード・トランザクション終了時のコミット直前に、upload_statistics 接続スクリプトが呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「download_statistics 接続イベント」 425 ページ
- 「download_statistics テーブル・イベント」 428 ページ
- 「upload_statistics テーブル・イベント」 543 ページ
- 「synchronization_statistics 接続イベント」 511 ページ
- 「synchronization_statistics テーブル・イベント」 514 ページ
- 「time_statistics 接続イベント」 517 ページ
- 「time_statistics テーブル・イベント」 520 ページ
- 「Mobile Link モニタ」 183 ページ
- 「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』

SQL の例

次の例は、アップロード操作での同期の統計を upload_summary_audit テーブルに挿入します。

```
CALL ml_add_connection_script (
 'ver1',
 'upload_statistics',
 'INSERT INTO upload_summary_audit (
   ml_user,
   warnings,
   errors,
   inserted_rows,
   deleted_rows,
   updated_rows,
   conflicted_inserts,
   conflicted_deletes,
   conflicted_updates,
   bytes,
   ignored_inserts,
   ignored deletes,
   ignored_updates,
   bytes, deadlocks )
  VALUES (
   {ml s.username},
   {ml s.warnings},
   {ml s.errors},
   {ml s.inserted_rows},
   {ml s.deleted_rows},
   {ml s.updated_rows},
   {ml s.conflicted_inserts},
   {ml s.conflicted_deletes},
   {ml s.conflicted_updates},
   {ml s.ignored_inserts},
   {ml s.ignored_deletes},
   {ml s.ignored_updates},
   {ml s.bytes},
   {ml s.deadlocks} ) ' )
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadStatisticsConnection という Java メソッドを upload_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
  'ver1',
  'upload_statistics',
  'ExamplePackage.ExampleClass.uploadStatisticsConnection' )
```

次に示すのは、サンプルの Java メソッド uploadStatisticsConnection です。このメソッドは、Mobile Link メッセージ・ログに統計の一部を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public String uploadStatisticsConnection(
  String user,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictedInserts,
  int conflictedDeletes,
  int conflictedUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks ) {
  java.lang.System.out.println( "updated rows: " +
   updatedRows );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、UploadStats という .NET メソッドを upload_statistics 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_connection_script(
  'ver1',
  'upload_statistics',
  'TestScripts.Test.UploadStats'
)
```

次に示すのは、サンプルの .NET メソッド UploadStats です。このメソッドは、Mobile Link メッセージ・ログに統計の一部を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string UploadStats (
  string user,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictInserts,
  int conflictDeletes,
```

```
    int conflictUpdates,
    int ignoredInserts,
    int ignoredDeletes,
    int ignoredUpdates,
    int bytes,
    int deadlocks ) {
    System.Console.WriteLine( "updated rows: " +
      updatedRows );
    return ( null );
  }
```


upload_statistics テーブル・イベント

特定のテーブルに対するアップロード操作について、同期統計を追跡します。

パラメータ

次の表の説明では、SQL データ型を示します。Java または .NET でスクリプトを作成する場合、適切なデータ型を使用してください。「SQL データ型と Java データ型」 558 ページと「SQL データ型と .NET データ型」 623 ページを参照してください。

SQL スクリプトでは、名前または疑問符を使用してイベント・パラメータを指定できますが、スクリプト内に名前と疑問符を混在させることはできません。疑問符を使用する場合、パラメータは以下に示す順に指定する必要があり、後続のパラメータが指定されていない場合のみ省略可能です (たとえば、パラメータ 2 を使用する場合は、パラメータ 1 を使用してください)。名前付きパラメータを使用する場合は、パラメータの任意のサブセットを任意の順に指定できます。

SQL スクリプトのパラメータ名	説明	順序
s.remote_id	VARCHAR(128)。Mobile Link リモート ID。名前付きパラメータを使用している場合のみ、リモート ID を参照できます。	適用不可
s.username	VARCHAR(128)。Mobile Link ユーザ名。	1
s.table	VARCHAR(128)。テーブル名。	2
s.warnings	INTEGER。テーブルのアップロードで発行された警告の数。	3
s.errors	INTEGER。処理済みのエラーを含め、テーブルのアップロードで発生したエラーの数。	4
s.inserted_rows	INTEGER。統合データベースに正常に挿入されたローの数。	5
s.deleted_rows	INTEGER。統合データベースから正常に削除されたローの数。	6
s.updated_rows	INTEGER。	7
s.conflicted_inserts	INTEGER。常に 0 (ゼロ) です。	8
s.conflicted_deletes	INTEGER。常に 0 (ゼロ) です。	9

SQL スクリプトのパラメータ名	説明	順序
s.conflicted_updates	INTEGER。競合を引き起こした更新ローの数。対処する競合解決スクリプトが正常に呼び出された場合のみ、ローは含まれます。	10
s.ignored_inserts	INTEGER。無視されたアップロード挿入ローの合計数。無視される理由は次のとおりです。1) 通常モードで upload_insert スクリプトないか、または強制的な競合モードで upload_new_row_insert スクリプトがない。2) Mobile Link サーバが対応するスクリプトを呼び出し中にエラーが発生し、handle_error または handle_odbc_error イベントが 1000 を返した。	11
s.ignored_deletes	INTEGER。upload_delete スクリプトを呼び出したとき、handle_error または handle_odbc_error が定義され、1000 を返した場合、または指定のテーブルに対して upload_delete スクリプトが定義されていない場合にエラーを発生したアップロード削除ローの数。	12
s.ignored_updates	INTEGER。競合を引き起こしたが、競合解決スクリプトが正常に呼び出されなかったり、upload_update スクリプトが定義されていなかったアップロード更新ローの数。	13
s.bytes	INTEGER。アップロードを保存するために Mobile Link サーバ内で使用されるメモリの容量。	14
s.deadlocks	INTEGER。同期で検出された統合データベース内のデッドロックの数。	15

デフォルトのアクション

なし

備考

upload_statistics イベントを使用すると、任意のユーザについて、任意のテーブルに適用される同期発生の重要な統計を収集できます。アップロード・トランザクション終了時のコミット直前に、upload_statistics テーブル・スクリプトが呼び出されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「download_statistics 接続イベント」 425 ページ
- 「upload_statistics 接続イベント」 538 ページ
- 「upload_statistics テーブル・イベント」 543 ページ
- 「synchronization_statistics 接続イベント」 511 ページ
- 「synchronization_statistics テーブル・イベント」 514 ページ
- 「time_statistics 接続イベント」 517 ページ
- 「time_statistics テーブル・イベント」 520 ページ
- 「Mobile Link モニタ」 183 ページ
- 「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』

SQL の例

次の例は、アップロードの統計を追跡するために使用されるテーブルにローを挿入します。

```
CALL ml_add_connection_script(
 'ver1',
 'upload_statistics',
 'INSERT INTO my_upload_statistics (
   user_name,
   table_name,
   num_warnings,
   num_errors,
   inserted_rows,
   deleted_rows,
   updated_rows,
   conflicted_inserts,
   conflicted_deletes,
   conflicted_updates,
   ignored_inserts,
   ignored_deletes,
   ignored_updates, bytes,
   deadlocks )
  VALUES(
   {ml s.username},
   {ml s.table},
   {ml s.warnings},
   {ml s.errors},
   {ml s.inserted_rows},
   {ml s.deleted_rows},
   {ml s.updated_rows},
   {ml s.conflicted_inserts},
   {ml s.conflicted_deletes},
   {ml s.conflicted_updates},
   {ml s.ignored_inserts},
   {ml s.ignored_deletes},
   {ml s.ignored_updates},
```

```
    {ml s.bytes},
    {ml s.deadlocks} )'  )
```

次の例は、Oracle 統合データベースで動作します。

```
CALL ml_add_connection_script(
 'ver1',
 'upload_statistics',
 'INSERT INTO upload_tables_audit (
  id,
  user_name,
  table,
  warnings,
  errors,
  inserted_rows,
  deleted_rows,
  updated_rows,
  conflicted_inserts,
  conflicted_deletes,
  conflicted_updates,
  ignored_inserts,
  ignored_deletes,
  ignored_updates,
  bytes,
  deadlocks )
 VALUES (
   ut_audit.nextval,
   {ml s.username},
   {ml s.table},
   {ml s.warnings},
   {ml s.errors},
   {ml s.inserted_rows},
   {ml s.deleted_rows},
   {ml s.updated_rows},
   {ml s.conflicted_inserts},
   {ml s.conflicted_deletes},
   {ml s.conflicted_updates},
   {ml s.ignored_inserts},
   {ml s.ignored_deletes},
   {ml s.ignored_updates},
   {ml s.bytes},
   {ml s.deadlocks} )' )
```

監査テーブルに統計が挿入されたら、これらの統計を使用して同期をモニタし、必要に応じて最適化を実行できます。

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadStatisticsTable という Java メソッドを upload_statistics テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'upload_statistics',
  'ExamplePackage.ExampleClass.uploadStatisticsTable' )
```

次に示すのは、サンプルの Java メソッド uploadStatisticsTable です。このメソッドは、Mobile Link メッセージ・ログに統計の一部を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。


```
public String uploadStatisticsTable(
  String user,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictedInserts,
  int conflictedDeletes,
  int conflictedUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks ) {
  java.lang.System.out.println( "updated rows: " +
    updatedRows );
  return ( null );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、UploadTableStats という .NET メソッドを upload_statistics テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'upload_statistics',
  'TestScripts.Test.UploadTableStats'
)
```

次に示すのは、サンプルの .NET メソッド uploadStatisticsTable です。このメソッドは、Mobile Link メッセージ・ログに統計の一部を出力します (統計を Mobile Link メッセージ・ログに出力すると、開発時には便利ですが、運用サーバのパフォーマンスが遅くなります)。

```
public string UploadTableStats(
  string user,
  string table,
  int warnings,
  int errors,
  int insertedRows,
  int deletedRows,
  int updatedRows,
  int conflictInserts,
  int conflictDeletes,
  int conflictUpdates,
  int ignoredInserts,
  int ignoredDeletes,
  int ignoredUpdates,
  int bytes,
  int deadlocks ) {
  System.Console.WriteLine( "updated rows: " +
    updatedRows );
  return ( null );
}
```

upload_update テーブル・イベント

リモート・データベースで更新されるローを処理するために、Mobile Link サーバがアップロード処理中に使用するイベントを提供します。

パラメータ

パラメータ	順序
r.*column-1*	1
...	...
r.*column-M*	M
r.*pk-column-1*	M + 1
...	...
r.*pk-column-N*	M + N
o.*column-N*	M + N + 1
...	...
o.*column-M*	M + N + M

デフォルトのアクション

なし

備考

文ベースの upload_update スクリプトは、UPLOAD 文で指定されたカラム値の直接更新を実行できます。

WHERE 句には、同期するプライマリ・キー・カラムをすべて含めます。SET 句には、同期する非プライマリ・キー・カラムをすべて含めます。

SET 句にはテーブルにある非プライマリ・キー・カラムをすべて指定します。Mobile Link は、適切な数のカラム値を送信します。また、WHERE 句には必要な数だけプライマリ・キーを指定できますが、ここですべてを指定してください。Mobile Link は適切な値を送信し、これらのカラム値とプライマリ・キー値を、スキーマの Mobile Link レポートに表示される順に送信します。-vh オプションを使用すると、このテーブル・スキーマのカラムの順序を指定できます。

たとえば、次の upload_update スクリプトでは、疑問符が適切な順序になっています。

```
UPDATE MyTable
  SET column_1 = ?, ..., column_M = ?
  WHERE pk_column_1 = ? AND ... AND pk_column_N = ?
```

リモート・データベースのテーブルごとに、upload_update スクリプトを 1 つ指定できます。

Java アプリケーションと .NET アプリケーションの場合、このスクリプトは有効な SQL を返します。

upload_update スクリプトを使用して競合を検出するには、以下のようにすべての 非プライマリ・キー・カラムを WHERE 句に含めます。

```
UPDATE table-name
SET col1 = {ml r.col1}, col2 = {ml r. col2 } ...
WHERE pk1 = {ml r.pk1} AND pk2 = {ml r.pk2} ...
  AND col1 = {ml o.col1} AND col2 ={ml o.col2} ...
```

この文では、col1 と col2 はプライマリ・キー・カラムではありませんが、pk1 と pk2 はプライマリ・キー・カラムです。非プライマリ・キー・カラムの 2 番目のセットに渡される値は、更新ローの更新前イメージです。WHERE 句は、リモート・データベースから更新された古い値と、統合データベースの現在の値を比較します。これらの値が一致しないと更新は無視されるので、すでに統合データベースにあった値は保持されます。

参照

- 「スクリプトのパラメータ」 336 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「upload_update スクリプトによる競合の検出」 152 ページ
- 「upload_update スクリプトによる競合の解決」 155 ページ
- 「upload_delete テーブル・イベント」 523 ページ
- 「upload_fetch テーブル・イベント」 525 ページ
- 「upload_insert テーブル・イベント」 530 ページ

SQL の例

この例は、リモート・データベース内の Customer テーブルに対する更新を処理します。このスクリプトは、統合データベース内のテーブル Customer 内の値を更新します。

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_update',
'UPDATE Customer
   SET name = {ml r.name}, rep_id = {ml r.rep_id}
     WHERE cust_id = {ml o.cust_id}')
```

Java の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、uploadUpdateTable という Java メソッドを upload_update テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'upload_update',
  'ExamplePackage.ExampleClass.uploadUpdateTable' )
```

次に示すのは、サンプルの Java メソッド uploadUpdateTable です。このメソッドは、UPLOAD 文を動的に生成する genUU というメソッドを呼び出します。

```
public String uploadUpdateTable() {
  return( genUU(_curTable) );
}
```

.NET の例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 とテーブル table1 を同期するときに、UploadUpdate という .NET メソッドを upload_update テーブル・イベント用のスクリプトとして登録します。

```
CALL ml_add_dnet_table_script(
  'ver1',
  'table1',
  'upload_update',
  'TestScripts.Test.UploadUpdate'
)
```

次に示すのは、サンプルの .NET メソッド UploadUpdate です。このメソッドは、UPLOAD 文を動的に生成する GenUU というメソッドを呼び出します。

```
public string UploadUpdate() {
  return ( genUU(_curTable) );
}
```

Mobile Link サーバ API

この項では、Java と .NET 用の Mobile Link サーバ API について説明します。

Java による同期スクリプトの作成

目次

Java 同期論理の概要

Mobile Link サーバの動作を制御するには、同期スクリプトを作成します。これらのスクリプトの実装には、SQL、.NET または Java を使用できます。Java 同期論理には、SQL 論理と同じ機能を持たせることができます。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できるのと同様に、Java メソッドを呼び出すことができます。Java メソッドは、SQL 文字列を Mobile Link に返すことができます。

この項では、Java 同期論理を設定、開発、実行する方法について説明します。また、サンプル・アプリケーションと Java 用 Mobile Link サーバ API リファレンスも含まれています。

参照

- 「チュートリアル：Java 同期論理の使用」 『Mobile Link - クイック・スタート』
- 「サーバ側の同期論理の作成オプション」 『Mobile Link - クイック・スタート』
- 「同期スクリプトの作成」 327 ページ

Java 同期論理の設定

SQL Anywhere をインストールすると、インストーラによって Java 用 Mobile Link サーバ API クラスのロケーションが自動的に設定されます。これらのクラスは、Mobile Link サーバの起動時にクラスパスに自動的に組み込まれます。Java 用 Mobile Link サーバ API クラスは、*install-dir¥Java¥mlscript.jar* にあります。

♦ Java を使用して同期スクリプトを実装するには、次の手順に従います。

1. 1 つまたは複数の独自クラスを作成します。必要な同期スクリプトごとに、メソッドを作成します。これらのメソッドは、パブリックにしてください。クラスは、パッケージ内ではパブリックにしてください。

   「メソッド」 559 ページを参照してください。

   非静的メソッドを持つ各クラスには、パブリック・コンストラクタが必要です。Mobile Link サーバは、各クラスのメソッドが初めて呼び出されるときに、そのクラスを自動的にインスタンス化します。

   「コンストラクタ」 558 ページを参照してください。

2. クラスをコンパイルするときは、JAR ファイル *java¥mlscript.jar* をインクルードしてください。

   次に例を示します。

   ```
   javac MyClass.java -classpath "c:¥Program Files¥SQL Anywhere 11¥Java¥mlscript.jar"
   ```

3. 統合データベースの Mobile Link システム・テーブルで、各同期スクリプトについて、呼び出すパッケージ、クラス、メソッドの名前を指定します。スクリプトのバージョンごとに、クラスを 1 つずつ使用します。

   たとえば、ml_add_java_connection_script ストアド・プロシージャまたは ml_add_java_table_script ストアド・プロシージャを使用して、この情報を Mobile Link システム・テーブルに追加できます。

   たとえば、次の SQL 文は、SQL Anywhere データベース内で実行すると、スクリプト・バージョン ver1 に対して、authenticate_user 接続レベル・イベントが発生するたびに myPackage.myClass.myMethod を実行するように指定します。指定されるメソッドはパブリック Java メソッドの完全修飾名で、大文字と小文字が区別されます。

   ```
   call ml_add_java_connection_script('ver1',
   'authenicate_user', 'myPackage.myClass.myMethod')
   ```

   スクリプト追加の詳細については、以下の項目を参照してください。

   - 「スクリプトを追加または削除するためのシステム・プロシージャ」 698 ページ
   - 「ml_add_java_connection_script システム・プロシージャ」 705 ページ
   - 「ml_add_java_table_script システム・プロシージャ」 706 ページ

4. Mobile Link サーバがクラスをロードするよう指定します。Java 同期論理の設定の中で最も重要な部分は、Java クラスの検索場所を仮想マシンに対して指定することです。このようにするには、次の 2 つの方法があります。

- クラスを検索するディレクトリまたは jar ファイルを指定するには、mlsrv11 -sl java -cp オプションを使用します。たとえば、次のコマンドを実行します。

  ```
  mlsrv11 -c "dsn=consolidated1" -sl java (-cp %classpath%;c:¥local¥Java¥myclasses.jar)
  ```

  Mobile Link サーバは、一連のディレクトリまたは jar ファイルに Java 用 Mobile Link サーバ API クラスのロケーション (java¥mlscript.jar) を自動的に追加します。また、-sl java オプションは、Java VM がサーバの起動時にロードされるよう指定します。

  使用可能な Java オプションの詳細については、「-sl java オプション」 94 ページを参照してください。

- 明示的にクラスパスを設定します。ユーザ定義クラスのクラスパスを設定するには、次のような文を使用します。

  ```
  SET classpath=%classpath%;c:¥local¥Java¥myclasses.jar
  ```

  システムのクラスパスに Java 同期論理のクラスが含まれている場合、Mobile Link サーバのコマンド・ラインを変更する必要はありません。

  -sl java オプションを使用すると、サーバ起動時に Java 仮想マシンを強制的にロードできます。このオプションを使用しない場合は、Java メソッドが最初に実行されるときに Java 仮想マシンが起動します。

  使用可能な Java オプションの詳細については、「-sl java オプション」 94 ページを参照してください。

5. UNIX で特定の JRE をロードする場合は、LD_LIBRARY_PATH (AIX の場合は LIBPATH、HP-UX の場合は SHLIB_PATH) を設定して、JRE を含むディレクトリを指定します。ディレクトリは、すべての SQL Anywhere インストール・ディレクトリの前に指定します。

参照

- 「Java 同期論理の作成」 557 ページ
- 「Java 同期の例」 565 ページ
- 「チュートリアル：Java 同期論理の使用」 『Mobile Link - クイック・スタート』
- 「Java 用 Mobile Link サーバ API リファレンス」 570 ページ
- 「サーバ側の同期論理の作成オプション」 『Mobile Link - クイック・スタート』
- 「同期スクリプトの作成」 327 ページ

Java 同期論理の作成

Java 同期論理を作成するには、Mobile Link イベントの知識、Java に関する若干の知識、Java 用 Mobile Link サーバ API の知識が必要です。

API の完全な説明については、「Java 用 Mobile Link サーバ API リファレンス」 570 ページを参照してください。

Java 同期論理を使用すると、ステータス情報を管理し、アップロード・イベントとダウンロード・イベント関連の論理を実装できます。たとえば、Java で作成された begin_synchronization スクリプトを使用すると、Mobile Link ユーザ名を変数に格納できます。同期処理中に後で呼び出されるスクリプトは、この変数にアクセスできます。また、Java は、コミットの実行前または実行後に統合データベースのローにアクセスするために使用できます。

Java を使用すると、統合データベースへの依存度が減少します。また、統合データベースを新バージョンにアップグレードしたり、別のデータベース管理システムに切り替えたりする場合も、動作に与える影響が少なくて済みます。

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、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・ハンドリングでは、Java または .NET 用 Mobile Link サーバ API の特別なクラスを使用して、同期対象のデータに直接アクセスします。

「ダイレクト・ロー・ハンドリング」 679 ページを参照してください。

クラス・インスタンス

Mobile Link サーバは、クラスを接続レベルでインスタンス化します。非静的 Java メソッドをあるイベントに対して作成した場合、そのイベントに達すると、現在の接続でクラスが作成されていなければ、Mobile Link サーバが自動的にクラスのインスタンスを作成します。

「コンストラクタ」 623 ページを参照してください。

1 つのスクリプト・バージョンの接続レベル・イベントまたはテーブルレベル・イベントに直接関連するすべてのメソッドは、「同じクラスに属している必要があります」。

データベース接続ごとに、インスタンス化されたクラスはその接続が終了するまで持続します。したがって、連続する複数の同期セッションで、同じインスタンスを使用することができます。明示的にクリアされないかぎり、パブリック変数またはプライベート変数内の情報は、同じ接続で発生するすべての同期を通して持続されます。

また、静的なクラスや変数も使用できます。この場合、値はすべての接続を通して使用できます。

統合データベースへの接続が終了した場合にのみ、Mobile Link サーバはクラス・インスタンスを自動的に削除します。

トランザクション

Java メソッドには、トランザクションに関する通常のルールが適用されます。データベース・トランザクションの開始と継続期間は、同期処理に重要になります。トランザクションの開始と終了は、Mobile Link サーバのみが行います。Java メソッド内の同期接続でトランザクションを明示的にコミットまたはロールバックすると、同期処理の整合性違反になり、エラーが発生することがあります。

これらのルールは、Mobile Link サーバによって作成されるデータベース接続、特に、メソッドから返される SQL 文にのみ適用されます。クラスによって他のデータベース接続が作成される場合は、既存の管理ルールを使用して、他のデータベース接続によって作成されたクラスを管理してください。

SQL データ型と Java データ型

次の表は、SQL データ型とそれに対応する Java データ型を示します。

SQL データ型	対応する Java データ型
VARCHAR	java.lang.String
CHAR	java.lang.String
INTEGER	int または Integer
BINARY	byte[ ]
TIMESTAMP	java.sql.Timestamp
INOUT INTEGER	ianywhere.ml.script.InOutInteger
INOUT VARCHAR	ianywhere.ml.script.InOutString
INOUT CHAR	ianywhere.ml.script.InOutString
INOUT BYTEARRAY	ianywhere.ml.script.InOutByteArray

ianywhere.ml.script パッケージが存在しない場合は、Mobile Link サーバがクラスパスに自動的に追加します。ただし、クラスをコンパイルするときは、*install-dir¥Java¥mlscript.jar* のパスを追加する必要があります。

コンストラクタ

クラスのコンストラクタは、次の 2 つのシグニチャのどちらかを持ちます。

```
public MyScriptClass(ianywhere.ml.script.DBConnectionContext sc)
```


または

```
public MyScriptClass()
```

渡される同期コンテキストは、Mobile Link サーバが現在のユーザの同期に使用している接続です。

DBConnectionContext.getConnection メソッドは、Mobile Link が現在のユーザの同期に使用しているのと同じデータベース接続を返します。この接続で文を実行することはできますが、トランザクションのコミットやロールバックは行わないでください。トランザクションは Mobile Link サーバによって管理されます。

Mobile Link サーバは、最初のシグニチャを持つコンストラクタを使用しようとします。引数のないコンストラクタが使用されるのは、最初のシグニチャを持つコンストラクタが存在しない場合のみです。

「DBConnectionContext インタフェース」 570 ページを参照してください。

メソッド

通常は、同期イベントごとにメソッドを 1 つずつ実装します。これらのメソッドは、パブリックにしてください。プライベート・メソッドの場合、Mobile Link サーバでは使用できず、その存在を認識できません。

統合データベース内の ml_script テーブルで指定されている名前と一致していれば、メソッド名は重要ではありません。ただし、このマニュアルの例では、メソッド名は Mobile Link イベント名と同じです。これは、Java コードを読みやすくするためです。

メソッドのシグニチャは、そのイベント用スクリプトのシグニチャと一致していなければなりません。ただし、パラメータ・リストの最後にパラメータ値が必要でない場合は、リストをトランケートできます。パラメータを渡すとオーバヘッドが生じる可能性があるため、必要なパラメータのみを受け入れてください。

ただし、メソッドはオーバロードできません。ml_script システム・テーブルには、クラスごとに 1 つのメソッド・プロトタイプのみが格納されます。

メソッドの登録

メソッドを作成したら、それを登録します。メソッドを登録すると、統合データベースの Mobile Link システム・テーブル内にメソッドへの参照が作成されて、イベントが発生すると、そのメソッドが呼び出されます。メソッドの登録方法は、同期スクリプトの追加方法と同じです。ただし、SQL スクリプト全体を Mobile Link システム・テーブルに追加する代わりに、メソッド名のみを追加します。

「スクリプトの追加と削除」 345 ページを参照してください。

戻り値

Mobile Link アップロードまたはダウンロードに対して呼び出されるメソッドは、有効な SQL 言語の文を返さなければなりません。これらのメソッドの戻り値の型は、java.lang.String にしてください。他の戻り値の型は使用できません。

他のすべてのスクリプトの戻り値の型は、java.lang.String または void にします。他の型は使用できません。戻り値の型が NULL ではなく文字列の場合、Mobile Link サーバはその文字列に有効な SQL 文が含まれているとみなして、この文を通常の SQL 言語による同期スクリプトと同様に統合データベース内で実行します。通常、メソッドは文字列を返しますが、その戻り値によってデータベースに対して SQL 文を実行しない場合は NULL を返すことができます。

Java クラスのデバッグ

Mobile Link は、Java コードのデバッグ時に役立つ各種の情報と機能を提供します。この項では、これらの情報の格納場所と機能の活用方法について説明します。

Mobile Link サーバのログ・ファイル内の情報

Mobile Link サーバは、メッセージをメッセージ・ログ・ファイルに書き込みます。サーバのログ・ファイルには、次の情報が書き込まれます。

- Java Runtime Environment。-jrepath オプションを使用すると、Mobile Link サーバの起動時に、特定の JRE を要求できます。デフォルト・パスは、SQL Anywhere 11 でインストールされた JRE のパスです。
- ロードされている標準 Java クラスのパス。これらのパスを明示的に指定しない場合、Mobile Link サーバはクラスパスに自動的に追加してから、Java 仮想マシンを起動します。
- 呼び出された特定のメソッドの完全指定名。この情報を使用すると、Mobile Link サーバが適切なメソッドを呼び出していることを確認できます。
- Java メソッドで java.lang.System.out または java.lang.System.err に書き込まれた出力。すべて Mobile Link サーバのログ・ファイルにリダイレクトされます。
- mlsrv11 コマンド・ラインのオプション。-verbose を使用できます。

  「-v オプション」 104 ページを参照してください。

Java デバッガの使用

標準 Java デバッガを使用して、Java クラスをデバッグできます。mlsrv11 コマンド・ラインで -sl java オプションを使用して、必要なパラメータを指定します。

「-sl java オプション」 94 ページを参照してください。

デバッガを指定すると、Java 仮想マシンは一時停止し、Java デバッガからの接続を待機します。

Java からの情報の出力

もう 1 つの方法として、java.lang.System.err か java.lang.System.out を使用して、Mobile Link メッセージ・ログに情報を出力する文を Java メソッドに追加することもできます。これにより、クラスの進行状況と動作を追跡できます。

パフォーマンスに関するヒント
この方法で情報を出力すると、モニタ・ツールとして活用できますが、運用環境では推奨いたしません。

これと同じ方法を利用して、任意の同期情報のログを取ったり、スクリプトの使用方法に関する統計情報を収集したりできます。

独自のテスト・ドライバの作成

独自のドライバを作成して、Java クラスをテストできます。このアプローチでは、Java メソッドのアクションが Mobile Link システムの残りの部分から分離されるため便利な場合があります。

Java での Mobile Link サーバ・エラーの処理

ログをスキャンするだけでは不十分な場合は、プログラムによってアプリケーションをモニタリングできます。たとえば、特定のタイプのメッセージを電子メールで送信できます。

ログに出力される各エラー・メッセージまたは各警告メッセージを表すクラスに渡されるメソッドを作成することも可能です。この方法は、Mobile Link サーバをモニタおよび監査するのに役立ちます。

次のコードは、すべての警告メッセージ用に LogListener をインストールし、その情報をファイルに書き込みます。

```
class TestLogListener implements LogListener {
   FileOutputStream _out_file;

   public TestLogListener(FileOutputStream out_file) {
      _out_file = out_file;
   }

   public void messageLogged(ServerContext sc, LogMessage msg) {
      String  type;
      String  user;

      try {
         if (msg.getType() == LogMessage.ERROR) {
            type = "ERROR";
         } else if (msg.getType() == LogMessage.WARNING) {
            type = "WARNING";
         }
         else {
            type = "UNKNOWN!!!";
         }

         user = msg.getUser();
         if (user == null) {
            user = "NULL";
         }
         _out_file.write(("Caught msg type="
            + type
            + " user=" + user
            + " text=" + msg.getText()
            + "¥n").getBytes()
         );
         _out_file.flush();
      }
      catch(Exception e) {
         // Print some error output to the MobiLink log.
         e.printStackTrace();
      }
```

```
    }
}
```

次のコードは TestLogListener を登録して、警告メッセージを受信します。クラス・コンストラクタや同期スクリプトなど、ServerContext にアクセスできる任意の場所からこのコードを呼び出してください。

```
// ServerContext serv_context;
serv_context.addWarningListener(
    new MyLogListener(ll_out_file)
);
```

参照

- 「addErrorListener メソッド」 595 ページ
- 「removeErrorListener メソッド」 599 ページ
- 「addWarningListener メソッド」 595 ページ
- 「removeWarningListener メソッド」 600 ページ
- 「LogListener インタフェース」 587 ページ
- 「LogMessage クラス」 588 ページ

ユーザ定義起動クラス

サーバの起動時に自動的にロードされる起動クラスを定義できます。この機能の目的は、最初の同期の前に Mobile Link サーバが JVM を起動する時点で実行される Java コードを記述できるようにすることです。つまり、ユーザ同期要求の前に、接続の作成またはデータのキャッシュを実行できます。

この操作を行うには、mlsrv11 -sl java オプションの DMLStartClasses オプションを使用します。たとえば、次に示すのは mlsrv11 コマンド・ラインの一部です。mycl1 と mycl2 が起動クラスとしてロードされます。

```
-sl java(-DMLStartClasses=com.test.mycl1,com.test.mycl2)
```

クラスはリスト内の順序でロードされます。同じクラスがリストに 2 回以上指定されている場合は、複数のインスタンスが作成されます。

起動クラスはすべてパブリックにしてください。また、引数を受け付けないか、ianywhere.ml.script.ServerContext 型の引数を 1 つ受け入れるパブリック・コンストラクタが必要です。

ロードされた起動クラスの名前は、「Java 起動クラス *classname* がロードされました。」というメッセージとともに Mobile Link ログに出力されます。

Java 仮想マシンのオプションの詳細については、「-sl java オプション」 94 ページを参照してください。

サーバ起動時に構築される起動クラスを表示する方法については、「getStartClassInstances メソッド」 598 ページを参照してください。

例

次に示すのは、起動クラスのテンプレートです。これは、イベントを処理してデータベース接続を確立するデーモン・スレッドを起動します(すべての起動クラスがスレッドの作成を必要とするわけではありませんが、スレッドを作成する場合はデーモン・スレッドでなければなりません)。

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

public class StartTemplate extends
    Thread implements ShutdownListener {
    ServerContext   _sc;
    Connection      _conn;
    boolean         _exit_loop;

    public StartTemplate(ServerContext sc) throws SQLException {

        // Perform setup first so that an exception
        // causes MobiLink startup to fail.
        _sc = sc;

        // Create a connection for use later.
        _conn = _sc.makeConnection();

        _exit_loop = false;
        setDaemon(true);
        start();
    }

    public void run() {
        _sc.addShutdownListener(this);

        // run() cannot throw exceptions.
        try {
            handlerLoop();
            _conn.close();
            _conn = null;
        }
        catch(Exception e) {

            // Print some error output to the MobiLink log.
            e.printStackTrace();

            // This thread shuts down and so does not
            // need to be notified of shutdown.
            _sc.removeShutdownListener(this);

            // Ask server to shutdown so that this fatal
            // error is fixed.
            _sc.shutdown();
        }
        // Shortly after return this thread no longer exists.
        return;
    }

    // stop our event handler loop
    public void shutdownPerformed(ServerContext sc) {
        try {
            // Wait max 10 seconds for thread to die.
            join(10*1000);
        }
        catch(Exception e) {
```

```
            // Print some error output to the MobiLink log.
            e.printStackTrace();
        }
    }

    private void handlerLoop() throws InterruptedException {
        while (!_exit_loop) {
            // Handle events in this loop. Sleep not
            // needed, block on event queue.
            sleep(1 * 1000);
        }
    }
}
```


Java 同期の例

Java 同期論理は Mobile Link や共通 Java クラスと連動し、Mobile Link サーバを使用したアプリケーションの配備に柔軟性を提供します。次の項では、単純な例を使用して、この広範囲な機能について説明します。

この項では、Java 同期論理の実例を挙げて説明します。このクラスを使用したり独自のクラスを作成したりする前に、次のチェックリストを使用してすべてを満たしているかどうかを確認してから、クラスをコンパイルしてください。

- 擬似コードなどを使用する機能の計画。
- データベース・テーブルとカラムのマッピングの作成。
- Mobile Link システム・テーブルで、Java 同期メソッドの言語タイプとロケーションを指定していることを確認して、Java 同期用の統合データベースを設定。

  「Java 同期論理の設定」 555 ページを参照してください。
- Java クラスの実行中に呼び出される関連 Java クラスのリストの作成。
- Mobile Link サーバのクラスパスに含まれるロケーションに Java クラスを格納。

プラン

この例の Java 同期論理は、例の処理に必要な機能を指定する関連 Java ファイルとクラスを指しています。この例は、クラス CustEmpScripts の作成方法を示します。また、接続用同期コンテキストの設定方法も示します。最後に、次の用途を持つ Java メソッドを提供します。

- Mobile Link ユーザを認証する。
- 各データベース・テーブル用のカーソルを使用して、ダウンロード操作とアップロード操作を実行する。

スキーマ

同期対象となるテーブルは emp と cust です。emp テーブルには、3 つのカラム emp_id、emp_name、manager があります。cust テーブルには、cust_id、cust_name、emp_id という 3 つのカラムがあります。各テーブルのすべてのカラムが同期されます。統合データベースからリモート・データベースへのマッピングにより、テーブル名とカラム名はどちらのデータベースでも同じです。さらに、監査テーブルが統合データベースに追加されます。

Java クラス・ファイル

この例で使用するファイルは *Samples¥MobiLink¥JavaAuthentication* ディレクトリにあります。

設定

次のコードは Java 同期論理を設定します。import 文は、Java 仮想マシンに対して、必要なファイルのロケーションを指定します。public class 文はクラスを宣言します。

```
// Use a package when you create your own script.
import ianywhere.ml.script.InOutInteger;
import ianywhere.ml.script.DBConnectionContext;
import ianywhere.ml.script.ServerContext;
```

```
import java.sql.*;

public class CustEmpScripts {
    // Context for this synchronization connection.
    DBConnectionContext _conn_context;

    // Same connection MobiLink uses for sync.
    // Do not commit or close this.
    Connection _sync_connection;
    Connection _audit_connection;

    //Get a user id given the user name. On audit connection.
    PreparedStatement _get_user_id_pstmt;

    // Add record of user logins added. On audit connection.
    PreparedStatement _insert_login_pstmt;

    // Prepared statement to add a record to the audit table.
    // On audit connection.
    PreparedStatement _insert_audit_pstmt;

    // ...
}
```

CustEmpScripts コンストラクタは、authenticateUser メソッドの準備文をすべて設定します。また、メンバ・データも設定します。

```
public CustEmpScripts(DBConnectionContext cc) throws SQLException {
    try {
        _conn_context = cc;
        _sync_connection = _conn_context.getConnection();

        ServerContext serv_context =
        _conn_context.getServerContext();
        _audit_connection = serv_context.makeConnection();

        // Get the prepared statements ready.
        _get_user_id_pstmt =
            _audit_connection.prepareStatement(
                "select user_id from ml_user where name = ?"
            );

        _insert_login_pstmt =
            _audit_connection.prepareStatement(
                "INSERT INTO login_added(ml_user, add_time) "
                    + "VALUES (?, { fn CONVERT({ fn NOW() }, SQL_VARCHAR) })"
            );

        _insert_audit_pstmt =
            _audit_connection.prepareStatement(
                "INSERT INTO login_audit(ml_user_id, audit_time, audit_action) "
                    + "VALUES (?, { fn CONVERT({ fn NOW() }, SQL_VARCHAR) }, ?)"
            );
    }
    catch(SQLException e) {
        freeJDBCResources();
        throw e;
    }
    catch(Error e) {
        freeJDBCResources();
        throw e;
    }
}
```


finalize メソッドは、end_connection が呼び出されなければ JDBC リソースをクリーンアップします。freeJDBCResources メソッドを呼び出し、割り付けられたメモリを解放して、監査接続を終了します。

```
protected void finalize() throws SQLException, Throwable {
    super.finalize();
    freeJDBCResources();
}

private void freeJDBCResources() throws SQLException {
    if (_get_user_id_pstmt != null) {
        _get_user_id_pstmt.close();
    }
    if (_insert_login_pstmt != null) {
        _insert_login_pstmt.close();
    }
    if (_insert_audit_pstmt != null) {
        _insert_audit_pstmt.close();
    }
    if (_audit_connection != null) {
        _audit_connection.close();
    }
    _conn_context       = null;
    _sync_connection    = null;
    _audit_connection   = null;
    _get_user_id_pstmt  = null;
    _insert_login_pstmt = null;
    _insert_audit_pstmt = null;
}
```

endConnection メソッドは、不要になった時点でリソースをクリーンアップします。

```
public void endConnection() throws SQLException {
    freeJDBCResources();
}
```

次の authenticateUser メソッドは、すべてのユーザ・ログインを承認し、データベース・テーブルにユーザ情報のログを取ります。ユーザが ml_user テーブルにない場合は、そのログが login_added に書き込まれます。ユーザ ID が ml_user 内で見つかると、そのログが login_audit に書き込まれます。実際のシステムでは user_password を無視することはありませんが、この例では単純にするためにすべてのユーザを承認しています。データベース操作のいずれかが失敗して例外が発生すると、endConnection メソッドは SQLException を発行します。

```
public void authenticateUser(
    InOutInteger authentication_status,
    String user_name) throws SQLException
{

    boolean new_user;
    int user_id;

    // Get ml_user id.
    _get_user_id_pstmt.setString(1, user_name);

    ResultSet user_id_rs =
    _get_user_id_pstmt.executeQuery();
    new_user = !user_id_rs.next();
    if (!new_user) {
        user_id = user_id_rs.getInt(1);
    }
```

```
        else {
            user_id = 0;
        }

        user_id_rs.close();
        user_id_rs = null;

        // In this tutorial always allow the login.
        authentication_status.setValue(1000);

        if (new_user) {
            _insert_login_pstmt.setString(1, user_name);
            _insert_login_pstmt.executeUpdate();
            java.lang.System.out.println("user: " + user_name + " added. ");
        }
        else {
            _insert_audit_pstmt.setInt(1, user_id);
            _insert_audit_pstmt.setString(2, "LOGIN ALLOWED");
            _insert_audit_pstmt.executeUpdate();
        }
        _audit_connection.commit();
        return;
    }
```

次のメソッドは、SQL コードを使用して、データベース・テーブル上でカーソルとして動作します。これらはカーソル・スクリプトであるため、SQL 文字列を返します。

```
    public static String empUploadInsertStmt() {
        return("INSERT INTO emp(emp_id, emp_name) VALUES(?, ?)");
    }

    public static String empUploadDeleteStmt() {
        return("DELETE FROM emp WHERE emp_id = ?");
    }

    public static String empUploadUpdateStmt() {
        return("UPDATE emp SET emp_name = ? WHERE emp_id = ?");
    }

    public static String empDownloadCursor() {
        return("SELECT emp_id, emp_name FROM emp");
    }

    public static String custUploadInsertStmt() {
        return("INSERT INTO cust(cust_id, emp_id, cust_name) VALUES (?, ?, ?)");
    }

    public static String custUploadDeleteStmt() {
        return("DELETE FROM cust WHERE cust_id = ?");
    }

    public static String custUploadUpdateStmt() {
        return("UPDATE cust SET emp_id = ?, cust_name = ? WHERE cust_id = ?");
    }

    public static String custDownloadCursor() {
        return("SELECT cust_id, emp_id, cust_name FROM cust");
    }
```

次のコマンドを使用して、コードをコンパイルします。

```
    javac -cp %sqlany11%¥java¥mlscript.jar CustEmpScripts.java
```


クラスパスの CustEmpScripts.class のロケーションを使用して Mobile Link サーバを実行します。次に、コマンド・ラインの一部を示します。

```
mlsrv11 ... -sl java (-cp <class_location>)
```

Java 用 Mobile Link サーバ API リファレンス

この項では、Mobile Link Java のインタフェースとクラス、これらに関連するメソッド、コンストラクタについて説明します。これらのクラスを使用するには、*install-dir¥java¥* にある *mlscript.jar* アセンブリを参照してください。

DBConnectionContext インタフェース

構文

public **ianywhere.ml.script.DBConnectionContext**

備考

現在のデータベース接続に関する情報を取得し、アクセスするためのインタフェース。DBConnectionContext インスタンスは、スクリプトを含むクラスのコンストラクタに渡されます。コンテキストがバックグラウンド・スレッドに必要な場合や、接続期間を超えて必要な場合は、ServerContext クラスを使用してください。

参照

- 「コンストラクタ」 558 ページ
- 「ServerContext インタフェース」 593 ページ

メンバ

ianywhere.ml.script.DBConnectionContext のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getConnection メソッド」 571 ページ
- 「getDownloadData メソッド」 571 ページ
- 「getProperties メソッド」 572 ページ
- 「getRemoteID メソッド」 573 ページ
- 「getServerContext メソッド」 573 ページ
- 「getVersion メソッド」 574 ページ

例

次の例は、同期スクリプトで使用するクラス・レベルの DBConnectionContext インスタンスを作成する方法を示します。DBConnectionContext の getConnection メソッドは、Mobile Link 統合データベースとの現在の接続を表す java.sql.Connection インスタンスを取得します。

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

public class OrderProcessor {
   DBConnectionContext _cc;

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



```
        // The method used for the handle_DownloadData event.
        public void HandleEvent() throws SQLException {
            java.sql.Connection my_connection = _cc.getConnection();
            // ...
        }

        // ...
    }
```

警告
DBConnectionContext インスタンスは、Java コードに呼び出すスレッド以外で使用しないでください。

getConnection メソッド

構文

public java.sql.Connection **getConnection( )**
throws java.sql.SQLException

備考

Mobile Link 統合データベースとの既存の接続を JDBC 接続として返します。このメソッドによって返された java.sql.Connection オブジェクトは、SQL スクリプトを実行するために Mobile Link サーバが使用するとの同じ接続を表します。

この接続は、Mobile Link サーバのこの接続での使用に影響する方法でコミット、クローズ、または変更しないでください。返される接続は、基本となる Mobile Link 接続の期間中にのみ有効です。接続に対して end_connection イベントが呼び出された後は、その接続を使用しないでください。

既存の接続を JDBC 接続としてバインドするときにエラーが発生すると、java.sql.SQLException が発行されます。

フル・アクセス権を持つサーバ接続が必要な場合、ServerContext.makeConnection() を使用してください。

戻り値

Mobile Link 統合データベースとの既存の接続 (JDBC 接続として)

例

「DBConnectionContext インタフェース」 570 ページを参照してください。

getDownloadData メソッド

構文

public DownloadData **getDownloadData()**

備考

現在の同期に対する DownloadData インスタンスを返します。ダイレクト・ロー・ハンドリング用のダウンロードを作成する場合は、DownloadData クラスを使用してください。

戻り値

現在の同期に対する DownloadData インスタンス

参照

- 「DownloadData インタフェース」 575 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次の例は、DBConnectionContext の getDownloadData メソッドを使用して、現在の同期に対する DownloadData インスタンスを取得する方法を示します。

> **注意**
> この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void HandleDownload() throws SQLException {
    // get the DownloadData for the current synchronization
    DownloadData my_dd = _cc.getDownloadData();

    // ...
}

// ...
```

getProperties メソッド

構文

public java.util.Properties **getProperties( )**

備考

この接続のスクリプト・バージョンに基づいて、この接続のプロパティを返します。プロパティは、ml_property テーブルに格納されます。

java.util.Properties の詳細については、Java SDK マニュアルを参照してください。

戻り値

この接続のプロパティ

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

例

次の例は、DBConnectionContext のプロパティを出力する方法を示します。

注意
この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used to output the connection properties.
public void outputProperties() {
    // output the Properties for the current synchronization
    java.util.Properties properties = _cc.getProperties();
    System.out.println(properties.toString());
}
```

getRemoteID メソッド

構文

public java.lang.String **getRemoteID( )**

備考

この接続で現在同期中のデータベースのリモート ID を返します。バージョン 10 以前のリモート・データベースの場合は、Mobile Link ユーザ名を返します。

戻り値

リモート ID

参照

- 「リモート ID」 『Mobile Link - クライアント管理』

例

次の例は、DBConnectionContext のリモート ID を出力する方法を示します。

注意
この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used to output the remote ID.
public void outputRemoteID() {
    // output the Remote ID for the current synchronization
    String remoteID = _cc.getRemoteID();
    System.out.println(remoteID);
}
```

getServerContext メソッド

構文

public ServerContext **getServerContext( )**

備考

この Mobile Link サーバの ServerContext を返します。

戻り値

この Mobile Link サーバの ServerContext

参照

- 「ServerContext インタフェース」 593 ページ

例

次の例は、DBConnectionContext の ServerContext インスタンスを取得して、サーバを停止する方法を示します。

> **注意**
> この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// A method that uses an instance of the ServerContext to shut down the server
public void shutDownServer() {
    ServerContext context = _cc.getServerContext();
    context.shutdown();
}
```

getVersion メソッド

構文

public java.lang.String **getVersion( )**

備考

スクリプト・バージョン文字列を返します。

戻り値

スクリプト・バージョン文字列

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

例

次の例は、スクリプト・バージョンを取得して、決定を行うために使用する方法を示します。

> **注意**
> この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// A method that uses the script version
public void handleEvent() {
```



```
        // ...

        String version = _cc.getVersion();
        if (version.equals("My Version 1")) {
            // ...
        } else if (version.equals("My Version 2")) {
            // ...
        }
    }

    // ...
```

DownloadData インタフェース

構文

public **ianywhere.ml.script.DownloadData**

備考

ダイレクト・ロー・ハンドリングで使用するダウンロード・データ操作をカプセル化します。DownloadData インスタンスを取得するには、DBConnectionContext の getDownloadData メソッドを使用します。

DownloadTableData インスタンスを返すには、DownloadData.getDownloadTables メソッドと getDownloadTableByName メソッドを使用します。

ダウンロード・データは DBConnectionContext を使用して取得できます。begin_synchronization イベントの前または end_download イベントの後にダウンロード・データにアクセスすることはできません。また、DownloadData にアップロード専用同期でアクセスすることもできません。

参照

- 「DownloadTableData インタフェース」 577 ページ
- 「handle_DownloadData 接続イベント」 465 ページ
- DBConnectionContext 「getDownloadData メソッド」 571 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

メンバ

ianywhere.ml.script.DownloadData のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getDownloadTableByName メソッド」 576 ページ
- 「getDownloadTables メソッド」 577 ページ

例

次の例は、DBConnectionContext の getDownloadData メソッドを使用して、現在の同期に対する DownloadData インスタンスを取得する方法を示します。

```
DBConnectionContext _cc;

// Your class constructor.
```

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

// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // ...
}
```

getDownloadTableByName メソッド

構文

public DownloadTableData getDownloadTableByName(
string *table-name*);

備考

現在の同期に使用する名前付きダウンロード・テーブルを取得します。指定された名前のテーブルが現在の同期にない場合は NULL を返します。

パラメータ

- **table_name**　ダウンロード・データの取得先テーブルの名前

戻り値

指定されたテーブルを表す DownloadTableData インスタンス。指定された名前のテーブルが現在の同期で存在しない場合は NULL。

参照

- 「DownloadData インタフェース」 575 ページ
- 「DownloadTableData インタフェース」 577 ページ
- 「DBConnectionContext インタフェース」 570 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次の例では getDownloadTableByName メソッドを使用して、remoteOrders テーブルの DownloadTableData インスタンスを返します。

> **注意**
> この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData my_download_table = my_dd.getDownloadTableByName("remoteOrders");
```



```
    // ...
}
```

getDownloadTables メソッド

構文

public DownloadTableData[ ] **getDownloadTables( )**

備考

現在の同期のダウンロード・データのすべてのテーブルを含む配列を取得します。このテーブルに対して実行された操作はリモート・データベースに送信されます。

戻り値

現在の同期の DownloadTableData オブジェクトの配列。配列内でのテーブルの順序は、リモートのアップロード順と同じです。

参照

- 「DownloadData インタフェース」 575 ページ
- 「DownloadTableData インタフェース」 577 ページ
- 「DBConnectionContext インタフェース」 570 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次の例では DownloadData.getDownloadTables メソッドを使用して、現在の同期の DownloadTableData オブジェクトの配列を取得します。この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get an array of tables to set download operations.
    DownloadTableData[] download_tables = my_dd.getDownloadTables();

    // Get the first table in the DownloadTableData array.
    DownloadTableData my_download_table = download_tables[0];

    // ...
}
```

DownloadTableData インタフェース

構文

public **ianywhere.ml.script.DownloadTableData**

備考

Mobile Link ダイレクト・ダウンロードのテーブル操作をカプセル化します。このインタフェースを使用して、クライアントにダウンロードされるデータ操作を設定します。現在の同期の DownloadTableData インスタンスを取得するには、DownloadData インタフェースを使用します。DownloadTableData.getUpsertPreparedStatement と getDeletePreparedStatement メソッドを使用して、それぞれ更新／挿入操作と削除操作を行う Java 準備文を取得できます。java.sql.PreparedStatement.executeUpdate メソッドはダウンロードの操作を登録します。

注意
挿入と更新用の準備文のすべてのカラム値を設定してください。削除操作の場合は、プライマリ・キー値を設定します。

削除とアップサート準備文を両方同時に開いておくことはできません。

java.sql.PreparedStatement の詳細については、Java SDK マニュアルを参照してください。

参照

- 「DownloadData インタフェース」 575 ページ
- 「handle_DownloadData 接続イベント」 465 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

メンバ

ianywhere.ml.script.DownloadTableData のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getDeletePreparedStatement メソッド」 579 ページ
- 「getUpsertPreparedStatement メソッド」 580 ページ
- 「getName メソッド」 582 ページ
- 「getMetaData メソッド」 582 ページ
- 「getLastDownloadTime メソッド」 583 ページ

例

Mobile Link クライアント・データベースで remoteOrders というテーブルを使用しているものとします。

```
CREATE TABLE remoteOrders (
   pk INT NOT NULL,
   col1 VARCHAR(200),
   PRIMARY KEY (pk)
);
```

次の例では DownloadData.getDownloadTableByName メソッドを使用して、remoteOrders テーブルを表す DownloadTableData インスタンスを返します。

注意
この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event
public void handleDownload() throws SQLException {
```



```
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

    // User defined-methods to set download operations.
    setDownloadInserts(td);
    setDownloadDeletes(td);

    // ...
}
```

この例では、setDownloadInserts メソッドは DownloadTableData.getUpsertPreparedStatement を使用して、挿入または更新するローの準備文を取得します。PreparedStatement.setInt メソッドと PreparedStatement.setString メソッドは、リモート・データベースに挿入するカラム値を設定します。

```
void setDownloadInserts(DownloadTableData td) {
    java.sql.PreparedStatement insert_ps = td.getUpsertPreparedStatement();

    // The following method calls are the same as the following SQL statement:
    // INSERT INTO remoteOrders(pk, col1) values(2300, "truck");
    insert_ps.setInt(1, 2300);
    insert_ps.setString(2, "truck");

    int update_result = insert_ps.executeUpdate();
    if (update_result == 0) {
        // Insert was filtered because it was uploaded
        // in the same synchronization.
    }
    else {
        // Insert was not filtered.
    }
}
```

setDownloadDeletes メソッドは DownloadTableData.getDeletePreparedStatement を使用して、削除するローの準備文を取得します。java.sql.PreparedStatement.setInt メソッドは、リモート・データベースで削除するローのプライマリ・キー値を設定し、java.sql.PreparedStatement.executeUpdate メソッドはダウンロードするロー値を登録します。

```
void setDownloadDeletes(DownloadTableData td) {
    java.sql.PreparedStatement delete_ps = td.getDeletePreparedStatement();

    // The following method calls are the same as the following SQL statement:
    // DELETE FROM remoteOrders  where pk=2300;
    delete_ps.setInt(1, 2300);
    delete_ps.executeUpdate();
}
```

getDeletePreparedStatement メソッド

構文

public java.sql.PreparedStatement **getDeletePreparedStatement( )** throws **SQLException**

備考

ユーザが削除操作をダウンロードに追加できるようにする java.sql.PreparedStatement インスタンスを返します。この準備文はダウンロード・テーブルに適用され、テーブルの各プライマリ・キー・カラムに対するパラメータを含んでいます。

この準備文は DownloadTableData インスタンスに適用され、テーブルの各プライマリ・キー・カラムに対するパラメータを含んでいます。

ダウンロードに削除操作を含めるには、java.sql.PreparedStatement ですべてのカラムを設定してから、java.sql.PreparedStatement.executeUpdate メソッドを呼び出します。

> **注意**
> ダウンロード削除操作対象のすべてのプライマリ・キー値を設定してください。

戻り値

削除操作をダウンロードに追加するための java.sql.PreparedStatement インスタンス

例外

- **SQLException**　削除用の java.sql.PreparedStatement インスタンスの取り出し時に問題が発生した場合に発行されます。

参照

- 「DownloadTableData インタフェース」 577 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次の例では、setDownloadDeletes メソッドは DownloadTableData.getDeletePreparedStatement を使用して、削除するローの準備文を取得します。java.sql.PreparedStatement.setInt メソッドは、リモート・データベースで削除するローのプライマリ・キー値を設定し、java.sql.PreparedStatement.executeUpdate メソッドはダウンロード内のロー値を設定します。

```
void setDownloadDeletes(DownloadTableData td) {
   java.sql.PreparedStatement delete_ps = td.getDeletePreparedStatement();
   // This is the same as executing the following SQL statement:
   // DELETE FROM remoteOrders  where pk=2300;
   delete_ps.setInt(1, 2300);
   delete_ps.executeUpdate();
   delete_ps.close();
}
```

getUpsertPreparedStatement メソッド

構文

public java.sql.PreparedStatement **getUpsertPreparedStatement( )** throws **SQLException**

備考

ユーザがアップサート (更新または挿入) 操作を同期のダウンロードに追加できるようにする java.sql.PreparedStatement インスタンスを返します。この準備文は DownloadTableData インスタンスに適用され、テーブルの各カラムに対するパラメータを含んでいます。

ダウンロードに挿入または更新操作を含めるには、java.sql.PreparedStatement ですべてのカラム値を設定してから、java.sql.PreparedStatement.executeUpdate メソッドを呼び出します。準備文で java.sql.PreparedStatement.executeUpdate を呼び出すと、挿入または更新操作がフィルタされた場合には 0 が返され、フィルタされなかった場合には 1 が返されます。同じ同期でアップロードされた場合、操作はフィルタされます。

注意
ダウンロード挿入と更新操作のすべてのカラム値を設定してください。

戻り値

アップサート操作をダウンロードに追加するための java.sql.PreparedStatement インスタンス

例外

- **SQLException**　アップサート用の java.sql.PreparedStatement インスタンスの取り出し時に問題が発生した場合に発行されます。

参照

- 「DownloadTableData インタフェース」 577 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次の例では、setDownloadInserts メソッドは DownloadTableData.getUpsertPreparedStatement を使用して、挿入または更新するローの準備文を取得します。java.sql.PreparedStatement.setInt メソッドと PreparedStatement.setString メソッドはカラム値を設定し、PreparedStatement.executeUpdate メソッドはダウンロード内のロー値を設定します。

```
void setDownloadInserts(DownloadTableData td) {
   java.sql.PreparedStatement insert_ps = td.getUpsertPreparedStatement();

   // This is the same as executing the following SQL statement:
   // INSERT INTO remoteOrders(pk, col1) VALUES (2300, "truck");
   insert_ps.setInt(1, 2300);
   insert_ps.setString(2, "truck");

   int update_result = insert_ps.executeUpdate();
   if (update_result == 0) {
      // Insert was filtered because it was uploaded
      // in the same synchronization.
   }
   else {
      // Insert was not filtered.
   }
   insert_ps.close();
}
```

getName メソッド

構文

public java.lang.String **getName( )**

備考

DownloadTableData インスタンスのテーブル名を返します。DownloadTableData.getMetaData メソッドによって返される java.sql.ResultSetMetaData インスタンスを使用して、テーブル名にアクセスすることもできます。

戻り値

DownloadTableData インスタンスのテーブル名

参照

- 「DownloadTableData インタフェース」 577 ページ
- DownloadTableData 「getMetaData メソッド」 582 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次の例は、DownloadTableData インスタンスのテーブル名を出力する方法を示します。

注意
この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event
public void handleDownload() throws SQLException {

    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

    // Print the table name to standard output (remoteOrders)
    System.out.println(td.getName());

    // User defined-methods to set download operations.
    setDownloadInserts(td);
    setDownloadDeletes(td);

    // ...
}
```

getMetaData メソッド

構文

public java.sql.ResultSetMetaData **getMetaData( )**

備考

DownloadTableData インスタンスのメタデータを取得します。メタデータは標準 java.sql.ResultSetMetaData オブジェクトです。

メタデータにカラム名情報を含める場合は、クライアントで、アップロードとともにカラム名が送信されるように指定します。

java.sql.ResultSetMetaData の詳細については、Java SDK マニュアルを参照してください。

戻り値

DownloadTableData インスタンスのメタデータ

参照

- 「DownloadTableData インタフェース」 577 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ
- SQL Anywhere クライアント：「SendColumnNames (scn) 拡張オプション」 『Mobile Link - クライアント管理』
- Ultra Light：「Send Column Names 同期パラメータ」 『Ultra Light データベース管理とリファレンス』

例

次の例は、DownloadTableData インスタンスのクエリで使用されるカラムの数を取得する方法を示します。

```
import java.sql.ResultSetMetaData;

// The method used to return the number of columns in a DownloadTableData instance query
public int getNumColumns(DownloadTableData td) {
    ResultSetMetaData rsmd = td.getMetaData();
    return rsmd.getColumnCount();
}
```

getLastDownloadTime メソッド

構文

public java.sql.Timestamp **getLastDownloadTime( )**

備考

このテーブルの最終ダウンロード時刻を返します。これは、テーブルごとのダウンロード・イベントの多くで渡される最終ダウンロード時刻と同じです。

最終ダウンロード時刻は、特定の同期に対してテーブル・ダウンロード・データを生成する場合に便利です。

戻り値

このダウンロード・テーブルの最終ダウンロード時刻

参照

- 「DownloadTableData インタフェース」 577 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次の例は、最終ダウンロード時刻を使用した挿入を含むダウンロードでテーブルを設定するコードを抜粋したものです。この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event
public void handleDownload() throws SQLException {

    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.getDownloadData();

    // Get the DownloadTableData for the remoteOrders table.
    DownloadTableData td = my_dd.getDownloadTableByName("remoteOrders");

    // get the inserts given a last download time
    ResultSet inserts_rs  = makeInsertsFromTimestamp(td.getLastDownloadTime());

    // fill the DownloadTableData using the inserts resultset.
    setDownloadInsertsFromRS(td, inserts_rs);
    inserts_rs.close();

    // ...
}
```

InOutInteger インタフェース

構文

public **ianywhere.ml.script.InOutInteger**

備考

SQL スクリプトに渡される in/out パラメータの機能を有効にするために、メソッドに渡されます。

メンバ

ianywhere.ml.script.InOutInteger のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getValue メソッド」 585 ページ
- 「setValue メソッド」 585 ページ

例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handleError という Java メソッドを handle_error 接続イベント用のスクリプトとして登録します。


```
CALL ml_add_java_connection_script(
   'ver1',
   'handle_error',
   'ExamplePackage.ExampleClass.handleError'
)
```

次に示すのは、サンプルの Java メソッド handleError です。このメソッドは、渡されたデータに基づいてエラーを処理します。また、エラーの結果生じるエラー・コードも判別します。

```
public String handleError(
   ianywhere.ml.script.InOutInteger actionCode,
   int errorCode,
   String errorMessage,
   String user,
   String table)
{

   int new_ac;
   if (user == null) {
      new_ac = handleNonSyncError(errorCode, errorMessage);
   } else if (table == null) {
      new_ac = handleConnectionError(errorCode, errorMessage, user);
   }
   else {
      new_ac = handleTableError(errorCode, errorMessage, user, table);
   }

   // Keep the most serious action code.
   if (actionCode.getValue() < new_ac) {
      actionCode.setValue(new_ac);
   }
}
```

getValue メソッド

構文

public int **getValue( )**

備考

この整数パラメータの値を返します。

戻り値

この整数パラメータの値

例

「InOutInteger インタフェース」 584 ページを参照してください。

setValue メソッド

構文

public void **setValue(** int *new_value* **)**

備考

この整数パラメータの値を設定します。

パラメータ

- **new_value**　この整数が取る値

例

「InOutInteger インタフェース」 584 ページを参照してください。

InOutString インタフェース

構文

public **ianywhere.ml.script.InOutString**

備考

SQL スクリプトに渡される in/out パラメータの機能を有効にするために、メソッドに渡されます。

メンバ

ianywhere.ml.script.InOutString のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getValue メソッド」 587 ページ
- 「setValue メソッド」 587 ページ

例

次の Mobile Link システム・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、modifyUser という Java メソッドを modify_user 接続イベント用のスクリプトとして登録します。

```
CALL ml_add_java_connection_script(
   'ver1',
   'modify_user',
   'ExamplePackage.ExampleClass.modifyUser'
)
```

次に示すのは、サンプルの Java メソッド modifyUser です。このメソッドは、データベースからユーザ ID を取得し、それを使用してユーザ名を設定します。

```
public String modifyUser(InOutString io_user_name) throws SQLException {
   Statement uid_select = curConn.createStatement();
   ResultSet uid_result = uid_select.executeQuery(
      "SELECT rep_id FROM SalesRep WHERE name = '"
         + io_user_name.getValue() + "' "
   );
   uid_result.next();
   io_user_name.setValue(java.lang.Integer.toString(uid_result.getInt(1));
   uid_result.close();
   uid_select.close();
```

```
        return (null);
    }
```

getValue メソッド

構文

public java.lang.String **getValue( )**

備考

この文字列パラメータの値を返します。

戻り値

この文字列パラメータの値

例

「InOutString インタフェース」 586 ページを参照してください。

setValue メソッド

構文

public void **setValue(** java.lang.String *new_value* **)**

備考

この文字列パラメータの値を設定します。

パラメータ

- **new_value**　この文字列が取る値

例

「InOutString インタフェース」 586 ページを参照してください。

LogListener インタフェース

構文

public **ianywhere.ml.script.LogListener**

備考

ログに出力されるメッセージを取得するためのリスナ・インタフェースです。

参照

- 「Java での Mobile Link サーバ・エラーの処理」 561 ページ

メンバ

ianywhere.ml.script.LogListener のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「messageLogged メソッド」 588 ページ

例

「Java での Mobile Link サーバ・エラーの処理」 561 ページを参照してください。

messageLogged メソッド

構文

```
public void messageLogged(
  ServerContext sc,
  LogMessage message )
```

備考

メッセージがログに出力されたときに呼び出されます。

パラメータ

- **sc**　メッセージを出力しているサーバのコンテキスト
- **message**　Mobile Link ログに送信された LogMessage

例

「Java での Mobile Link サーバ・エラーの処理」 561 ページを参照してください。

LogMessage クラス

構文

public **ianywhere.ml.script.LogMessage**

備考

ログ・メッセージに関連付けられたデータを保持します。

java.lang.Object を拡張します。

参照

- 「Java での Mobile Link サーバ・エラーの処理」 561 ページ

メンバ

ianywhere.ml.script.LogMessage のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「ERROR 変数」 591 ページ
- 「INFO 変数」 591 ページ
- 「WARNING 変数」 591 ページ
- 「getType メソッド」 592 ページ
- 「getUser メソッド」 592 ページ
- 「getText メソッド」 592 ページ

例

次の例は、すべての警告、エラー、情報のメッセージ用に LogListener をインストールし、その情報をファイルに書き込みます。次のコードは、すべての警告メッセージ用に LogListener をインストールします。

```
class WarningLogListener implements LogListener {
   FileOutputStream _outFile;

   public WarningLogListener(FileOutputStream outFile) {
      _outFile = outFile;
   }

   public void messageLogged(ServerContext sc, LogMessage msg) {
      String user;

      try {
         if (msg.getType() != LogMessage.WARNING) {
            //this class deals exclusively with warnings
            return;
         }
         user = msg.getUser();

         if (user == null) {
            user = "NULL";
         }

         _outFile.write(("Caught warning"
            + " user=" + user
            + " text=" + msg.getText()
            + "¥n").getBytes()
         );
         _outFile.flush();
      }
      catch(Exception e) {
         // Print some error output to the MobiLink log.
         e.printStackTrace();
      }
   }
}
```

次のコードは、すべてのエラー・メッセージ用に LogListener をインストールします。

```
class ErrorLogListener implements LogListener {
   FileOutputStream _outFile;

   public ErrorLogListener(FileOutputStream outFile) {
      _outFile = outFile;
```

```
    }

    public void messageLogged(ServerContext sc, LogMessage msg) {
        String user;

        try {
            if (msg.getType() != LogMessage.ERROR) {
                //this class deals exclusively with errors
                return;
            }

            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }

            _outFile.write(("Caught error"
                    + " user=" + user
                    + " text=" + msg.getText()
                    + "¥n").getBytes()
            );
            _outFile.flush();
        }
        catch(Exception e) {
            // Print some error output to the MobiLink log.
            e.printStackTrace();
        }
    }
}
```

次のコードは、すべての情報メッセージ用に LogListener をインストールします。

```
class InfoLogListener implements LogListener {
    FileOutputStream _outFile;

    public InfoLogListener(FileOutputStream outFile) {
        _outFile = outFile;
    }

    public void messageLogged(ServerContext sc, LogMessage msg) {
        String user;

        try {
            if (msg.getType() != LogMessage.ERROR) {
                // this class deals exclusively with info
                return;
            }

            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }

            _outFile.write(("Caught info"
                    + " user=" + user
                    + " text=" + msg.getText()
                    + "¥n").getBytes()
            );
            _outFile.flush();
        }
        catch(Exception e) {
            // Print some error output to the MobiLink log.
            e.printStackTrace();
        }
```



```
    }
  }
```

次のコードは、警告、エラー、情報のメッセージを受信するために、それぞれ WarningLogListener、ErrorLogListener、InfoLogListener を登録します。クラス・コンストラクタや同期スクリプトなど、ServerContext にアクセスできる任意の場所からこのコードを呼び出してください。

```
// ServerContext serv_context;
// FileOutputStream outFile
serv_context.addWarningListener(new WarningLogListener(outFile));
serv_context.addErrorListener(new ErrorLogListener(outFile));
serv_context.addInfoListener(new InfoLogListener(outFile));
```

ERROR 変数

構文

int **ERROR**

備考

ログ・メッセージはエラーです。

例

「LogMessage クラス」 588 ページを参照してください。

INFO 変数

構文

int **INFO**

備考

メッセージ・ログに情報が表示されます。

例

「addInfoListener メソッド」 593 ページを参照してください。

WARNING 変数

構文

int **WARNING**

備考

ログ・メッセージは警告です。

例

「LogMessage クラス」 588 ページを参照してください。

getType メソッド

構文

public int **getType( )**

備考

このメッセージ・タイプのアクセサ。

戻り値

このメッセージのタイプ (LogMessage.ERROR、LogMessage.INFO、または LogMessage.WARNING)。

例

「LogMessage クラス」 588 ページを参照してください。

getUser メソッド

構文

public java.lang.String **getUser( )**

備考

このメッセージ・ユーザのアクセサ。メッセージのユーザが存在しない場合、ユーザは NULL です。

戻り値

このメッセージに対応するユーザ

例

「LogMessage クラス」 588 ページを参照してください。

getText メソッド

構文

public java.lang.String **getText( )**

備考

このメッセージ・テキストのアクセサ。

戻り値

このメッセージの本文

例

「LogMessage クラス」 588 ページを参照してください。

ServerContext インタフェース

構文

public **ianywhere.ml.script.ServerContext**

備考

Mobile Link サーバの継続期間中に存在する、すべてのコンテキストのインスタンス化。このコンテキストを静的なデータとして保持し、バックグラウンド・スレッドで使用できます。Mobile Link によって起動される Java 仮想マシンの継続期間中は有効です。

ServerContext インスタンスにアクセスするには、DBConnectionContext.getServerContext メソッドを使用します。

メンバ

ianywhere.ml.script.ServerContext のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「addInfoListener メソッド」 593 ページ
- 「addErrorListener メソッド」 595 ページ
- 「addShutdownListener メソッド」 595 ページ
- 「addWarningListener メソッド」 595 ページ
- 「getProperties メソッド」 596 ページ
- 「getPropertiesByVersion メソッド」 597 ページ
- 「getPropertySetNames メソッド」 597 ページ
- 「getStartClassInstances メソッド」 598 ページ
- 「makeConnection メソッド」 598 ページ
- 「removeErrorListener メソッド」 599 ページ
- 「removeInfoListener メソッド」 599 ページ
- 「removeShutdownListener メソッド」 600 ページ
- 「removeWarningListener メソッド」 600 ページ
- 「shutdown メソッド」 601 ページ

addInfoListener メソッド

構文

public void **addInfoListener(** LogListener *l* **)**

備考

情報が出力されたときに通知を受信するリスナのリストから、指定した LogListener を追加します。メソッド LogListener.messageLogged (ianywhere.ml.script.ServerContext) が呼び出されます。

パラメータ

- **ll** 通知を受信する LogListener。

例

次のコードは、MyLogListener 型のリスナを登録して情報メッセージの通知を受信します。

```
// ServerContext serv_context;
serv_context.addInfoListener(new MyLogListener(ll_out_file));

// The following code shows an example of processing those messages:
class MyLogListener implements LogListener {
    FileOutputStream _out_file;
    public TestLogListener(FileOutputStream out_file) {
        _out_file = out_file;
    }

    public void messageLogged(ServerContext sc, LogMessage msg) {
        String type;
        String user;

        try {
            if (msg.getType() == LogMessage.ERROR) {
                type = "ERROR";
            } else if (msg.getType() == LogMessage.WARNING) {
                type = "WARNING";
            } else if (msg.getType() == LogMessage.INFO) {
                type = "INFO";
            } else {
                type = "UNKNOWN!!!";
            }

            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }
            _out_file.write(("Caught msg type="
                + type
                + " user=" + user
                + " text=" +msg.getText()
                + "¥n").getBytes()
            );
            _out_file.flush();
        }
        catch(Exception e) {

            // if we print the exception from processing an info message,
            // we may recurse indefinitely
            if (msg.getType() != LogMessage.ERROR) {
                // Print some error output to the MobiLink log.
                e.printStackTrace();
            }
        }
    }
}
```

addErrorListener メソッド

構文

public void **addErrorListener(** LogListener *ll* **)**

備考

エラーが出力されたときに通知を受信するために、指定された LogListener を追加します。

エラーが出力されると、次のメソッドが呼び出されます。
LogListener.messageLogged(ianywhere.ml.script.ServerContext, ianywhere.ml.script.LogMessage)

パラメータ

- **ll**　通知を受信する LogListener。

参照

- 「messageLogged メソッド」 588 ページ

例

「LogMessage クラス」 588 ページを参照してください。

addShutdownListener メソッド

構文

public void **addShutdownListener(** ShutdownListener *sl* **)**

備考

サーバ・コンテキストが破壊される前に通知を受信する指定の ShutdownListener を追加します。
シャットダウン時に、メソッド ShutdownListener.shutdownPerformed
(ianywhere.ml.script.ServerContext) が呼び出されます。

パラメータ

- **sl**　シャットダウン時に通知を受信する ShutdownListener

例

「ShutdownListener インタフェース」 602 ページを参照してください。

addWarningListener メソッド

構文

public void **addWarningListener(** LogListener *ll* **)**

備考

警告が出力されたときに通知を受信するために、指定された LogListener を追加します。

次のメソッドが呼び出されます。LogListener.messageLogged(ianywhere.ml.script.ServerContext, ianywhere.ml.script.LogMessage)

パラメータ

- **ll**　通知を受信する LogListener。

例

「LogMessage クラス」 588 ページを参照してください。

getProperties メソッド

構文

```
public java.util.Properties getProperties(
  java.lang.String component,
  java.lang.String set )
```

備考

指定されたコンポーネントとプロパティ・セットに関連する一連のプロパティを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。

パラメータ

- **component**　コンポーネント
- **set**　プロパティ・セット

戻り値

一連のプロパティ。空の場合があります。

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

例

次のコードは、ServerContext のすべてのプロパティをリストします。

```
import java.util.*;
// ServerContext serverContext;
// PrintStream out
Properties prop = serverContext.getProperties();
prop.list(out);
```

getPropertiesByVersion メソッド

構文

public java.util.Properties **getPropertiesByVersion(** java.lang.String *script_version* **)**

備考

スクリプト・バージョンに関連する一連のプロパティを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。component_name が ScriptVersion の場合、スクリプト・バージョンは property_set_name カラムに格納されます。

パラメータ

- **script_version**　関連するプロパティを返すスクリプト・バージョン

戻り値

指定したスクリプト・バージョンに関連する一連のプロパティ

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

例

次のコードは、指定したスクリプト・バージョンに関連する、ServerContext のすべてのプロパティをリストします。

```
import java.util.*;
// ServerContext serverContext;
// PrintStream out
Properties prop = serverContext.getPropertiesByVersion("MyScriptVersion");
prop.list(out);
```

getPropertySetNames メソッド

構文

public Iterator **getPropertySetNames(**
  java.lang.String *component_name* **)**

備考

指定したコンポーネントのプロパティ・セット名のリストを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。

パラメータ

- **component_name**　プロパティ名をリストするコンポーネントの名前。

戻り値

指定したコンポーネントのプロパティ・セット名のリスト

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

例

次のコードは、指定したコンポーネントに関連する、ServerContext のすべてのプロパティをリストします。

```
import java.util.*;
// ServerContext serverContext;
// PrintStream out
Properties prop = serverContext.getPropertySetNames("Component Name");
prop.list(out);
```

getStartClassInstances メソッド

構文

public java.lang.Object[ ] **getStartClassInstances( )**
サーバ起動時に構築された起動クラスの配列を取得します。起動クラスがない場合、配列の長さは 0 です。

戻り値

サーバ起動時に構築された起動クラスの配列。起動クラスがない場合は、長さが 0 の配列。

例

次に、getStartClassInstances() を使用する例を示します。

```
Object objs[] = sc.getStartClassInstances();
int i;
for (i=0; i < objs.length; i += 1) {
    if (objs[i] instanceof MyClass) {
        // Use class.
    }
}
```

参照

- 「ユーザ定義起動クラス」 562 ページ

makeConnection メソッド

構文

public java.sql.Connection **makeConnection( )**
throws **java.sql.SQLException**

備考

新しいサーバ接続を作成します。新しい接続を開くときにエラーが発生すると、このメソッドは java.sql.SQLException を発行します。

戻り値

新しいサーバ接続

例外

- **SQLException**　新しい接続を開くときにエラーが発生した場合に発行されます。

removeErrorListener メソッド

構文

public void **removeErrorListener(** LogListener *l* **)**

備考

エラーが出力されたときに通知を受信するリスナのリストから、指定した LogListener を削除します。

パラメータ

- **l**　通知を受信しないようにする LogListener

例

次のコードは、エラー・リスナのリストから LogListener を削除します。

```
// ServerContext serverContext;
// LogListener myErrorListener
serverContext.removeErrorListener(myErrorListener);
```

removeInfoListener メソッド

構文

public void **removeInfoListener(** LogListener *l* **)**

備考

情報が出力されたときに通知を受信するリスナのリストから、指定した LogListener を削除します。

パラメータ

- **l**　通知を受信しないようにする Listener。

例

次のコードは、情報 Listener のリストから LogListener を削除します。

```
// ServerContext serverContext;
// LogListener myInfoListener
serverContext.removeInfoListener(myInfoListener);
```

removeShutdownListener メソッド

構文

public void **removeShutdownListener(** ShutdownListener *sl* **)**

備考

サーバ・コンテキストが破壊される前に通知を受信するリスナのリストから、指定の ShutdownListener を削除します。

パラメータ

- **sl**　シャットダウン時に通知を受信しないようにする ShutdownListener

例

次のコードは、サーバ・コンテキストが破壊される前に通知を受信するリスナのリストから、ShutdownListener を削除します。

```
// ServerContext serverContext;
// ShutdownListener myShutdownListener
serverContext.removeShutdownListener(myShutdownListener);
```

removeWarningListener メソッド

構文

public void **removeWarningListener(** LogListener *ll* **)**

備考

警告が出力されたときに通知を受信するリスナのリストから、指定した LogListener を削除します。

パラメータ

- **ll**　通知を受信しないようにする LogListener

例

次のコードは、警告リスナのリストから LogListener を削除します。

```
// ServerContext serverContext;
// LogListener myWarningListener
serverContext.removeWarningListener(myWarningListener);
```

shutdown メソッド

構文

public void **shutdown( )**

備考

サーバを強制的に停止します。登録された ShutdownListener インスタンスで、shutdownPerformed メソッドが呼び出されます。

例

次のコードは、サーバを強制的に停止します。

```
// ServerContext serverContext;
serverContext.shutdown();
```

ServerException クラス

構文

public **ianywhere.ml.script.ServerException**

備考

サーバで同期の進行を妨げるエラー状態が存在することを示すために発行されます。この例外が発行されると、Mobile Link サーバはシャットダウンされます。

メンバ

ianywhere.ml.script.ServerException のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「ServerException コンストラクタ」 602 ページ

例

次のコードは、致命的な問題が発生した場合に ServerException をスローできる関数です。ServerException がスローされると、Mobile Link サーバはシャットダウンされます。

```
public void handleUpload(UploadData ud)
    throws SQLException, IOException, ServerException
{

    UploadedTableData tables[] = ud.getUploadedTables();
    if (tables == null) {
        throw new ServerException("Failed to read uploaded tables");
    }

    for (int i = 0; i < tables.length; i++) {
        UploadedTableData currentTable = tables[i];
        println("table " + java.lang.Integer.toString(i)
            + " name: " + currentTable.getName());

        // Print out delete result set.
```

```
            println("Deletes");
            printRSInfo(currentTable.getDeletes());

            // Print out insert result set.
            println("Inserts");
            printRSInfo(currentTable.getInserts());

            // print out update result set
            println("Updates");
            printUpdateRSInfo(currentTable.getUpdates());
        }
    }
```

ServerException コンストラクタ

構文

public **ServerException( )**

備考

詳細メッセージのない ServerException を構成します。

構文

public **ServerException(** java.lang.String *s* **)**

備考

指定した詳細メッセージを含む ServerException を構成します。

パラメータ

- **s**　詳細メッセージ

例

「ServerException クラス」 601 ページを参照してください。

ShutdownListener インタフェース

構文

public **ianywhere.ml.script.ShutdownListener**

備考

サーバのシャットダウンを取得するリスナ・インタフェースです。このインタフェースを使用して、サーバが終了する前に、スレッド、接続、およびその他のリソースがすべてクリーンアップされるようにします。

メンバ

ianywhere.ml.script.ShutdownListener のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「shutdownPerformed メソッド」 603 ページ

例

次のコードは、ServerContext 用に ShutdownListener をインストールします。

```
class MyShutdownListener implements ShutdownListener {
   FileOutputStream _outFile;
   public MySutdownListener(FileOutputStream outFile) {
      _outFile = outFile;
   }

   public void shutdownPerformed(ServerContext sc) {
      // Add shutdown code
      try {
         _outFile.write(("Shutting Down" + "¥n").getBytes());
         _outFile.flush();
      }
      catch(Exception e) {
         // Print some error output to the MobiLink log.
         e.printStackTrace();
      }
      // ...
   }
}
```

次のコードは、MyShutdownListener を登録します。クラス・コンストラクタや同期スクリプトなど、ServerContext にアクセスできる任意の場所からこのコードを呼び出してください。

```
// ServerContext serv_context;
// FileOutputStream outFile
serv_context.addShutdownListener(new MyShutdownListener(outFile));
```

shutdownPerformed メソッド

構文

public void **shutdownPerformed(** ServerContext *sc***)**

備考

サーバのシャットダウンによって ServerContext が破壊される前に起動されます。

パラメータ

- **sc**　シャットダウンされるサーバのコンテキスト

例

「ShutdownListener インタフェース」 602 ページを参照してください。

SynchronizationException クラス

構文

public **ianywhere.ml.script.SynchronizationException**

備考

現在の同期の完了を妨げるエラー状態が存在することを示すために発行されます。この例外が発行されると、Mobile Link サーバは強制的にロールバックされます。

メンバ

ianywhere.ml.script.SynchronizationException のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「SynchronizationException コンストラクタ」 604 ページ

例

次のコードは、致命的な問題が発生した場合に SynchronizationException をスローできる関数です。SynchronizationException がスローされると、Mobile Link サーバがロールバックされます。

```
public void handleUpload(UploadData ud)
    throws SQLException, IOException, SynchronizationException
{

    UploadedTableData tables[] = ud.getUploadedTables();

    for (int i = 0; i < tables.length; i++) {
        UploadedTableData currentTable = tables[i];
        println("table " + java.lang.Integer.toString(i)
            + " name: " + currentTable.getName());

        // Print out delete result set.
        println("Deletes");
        printRSInfo(currentTable.getDeletes());

        // Print out insert result set.
        println("Inserts");
        printRSInfo(currentTable.getInserts());

        // print out update result set
        println("Updates");
        printUpdateRSInfo(currentTable.getUpdates());

        if (/* Reason for Sync failure */) {
            throw new SynchronizationException("Sync Failed");
        }
    }
}
```

SynchronizationException コンストラクタ

構文

public **SynchronizationException( )**

備考

詳細メッセージのない SynchronizationException を構成します。

構文

public **SynchronizationException(** java.lang.String *s* **)**

備考

指定した詳細メッセージを含む SynchronizationException を構成します。

パラメータ

- **s**　詳細メッセージ

UpdateResultSet

構文

public **ianywhere.ml.script.UpdateResultSet**

備考

指定した行の更新前イメージ値 (古い値) と更新後イメージ値 (新しい値) にアクセスするための特別なメソッドを含む結果セット・オブジェクトです。UpdateResultSet インスタンスを取得するには、DownloadTableData.getUpdates メソッドを使用します。

UpdateResultSet は java.sql.ResultSet を拡張して、setNewRowValues メソッドと setOldRowValues メソッドを追加しています。それ以外の場合は、通常の結果セットと同様に使用できます。java.sql.ResultSet の詳細については、Java SDK マニュアルを参照してください。

参照

- DownloadTableData 「getUpdates メソッド」 612 ページ
- 「ダイレクト・アップロードでの競合の処理」 686 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

メンバ

ianywhere.ml.script.UpdateResultSet のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「setNewRowValues メソッド」 606 ページ
- 「setOldRowValues メソッド」 606 ページ

例

次のコードは、DownloadTableData インスタンスから UpdateResultSet インスタンスを取得する方法を示します。

```
// DownloadTableData tableData
UpdateResultSet results = tableData.getUpdates();
```

setNewRowValues メソッド

構文

public void **setNewRowValues( )**

備考

新しいカラム値 (更新後のロー) を返すように、この結果セットのモードを設定します。結果セットは、リモート・クライアント・データベース内の最新の更新値を表します。これがデフォルト・モードです。

参照

- 「UpdateResultSet」 605 ページ
- 「ダイレクト・アップロードでの競合の処理」 686 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次のコードは、新しいカラム値を返すように、UpdateResultSet のモードを設定する方法を示します。

```
// UpdateResultSet results
results.setNewRowValues();
```

setOldRowValues メソッド

構文

public void **setOldRowValues( )**

備考

古いカラム値 (更新前のロー) を返すように、この結果セットのモードを設定します。このモードでは、UpdateResultSet は、最後の同期中にクライアントが取得した古いカラム値を表します。

参照

- 「UpdateResultSet」 605 ページ
- 「ダイレクト・アップロードでの競合の処理」 686 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次のコードは、古いカラム値を返すように、UpdateResultSet のモードを設定する方法を示します。

```
// UpdateResultSet results
results.setOldRowValues();
```


UploadData インタフェース

構文

public **ianywhere.ml.script.UploadData**

備考

ダイレクト・ロー・ハンドリングで使用するアップロード操作をカプセル化します。単一のアップロード・トランザクションを表す UploadData インスタンスが、handle_UploadData 同期イベントに渡されます。

> **警告**
> ダイレクト・ロー・ハンドリングのアップロード操作は、handle_UploadData イベントに対して登録したメソッドで処理してください。登録されたメソッドを呼び出した後、UploadData は破棄されます。後続のイベントで使用するために新しい UploadData インスタンスを作成しないでください。

UploadedTableData インスタンスを取得するには、UploadData.getUploadedTables メソッドまたは UploadData.getUploadedTableByName メソッドを使用します。

リモート・データベースがトランザクション・アップロードを使用している場合を除き、同期の UploadData は 1 つです。

参照

- 「UploadedTableData インタフェース」 609 ページ
- 「handle_UploadData 接続イベント」 477 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ
- 「ダイレクト・アップロードの処理」 685 ページ

メンバ

ianywhere.ml.script.UploadData のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getUploadedTableByName メソッド」 607 ページ
- 「getUploadedTables メソッド」 608 ページ

例

「handle_UploadData 接続イベント」 477 ページを参照してください。

getUploadedTableByName メソッド

構文

```
public UploadedTableData getUploadedTableByName(
  java.lang.String table_name
)
```

備考

指定されたテーブルを表す UploadedTableData インスタンスを返します。

パラメータ

- **table_name**　アップロード・データの取得先アップロード・テーブルの名前

戻り値

指定されたテーブルを表す UploadedTableData インスタンス。指定された名前のテーブルが現在の同期で存在しない場合は NULL。

参照

- 「UploadData インタフェース」 607 ページ
- 「UploadedTableData インタフェース」 609 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

handle_UploadData 同期イベントに対して HandleUpload というメソッドを使用するものとします。次の例では getUploadedTableByName メソッドを使用して、remoteOrders テーブルの UploadedTableData インスタンスを返します。

```
// The method used for the handle_UploadData event.

public void handleUpload(UploadData ut)
    throws SQLException, IOException
{

    UploadedTableData uploaded_t1 = ut.getUploadedTableByName("remoteOrders");
    // ...
}
```

getUploadedTables メソッド

構文

public UploadedTableData[] **getUploadedTables( )**

備考

現在の同期の UploadedTableData オブジェクトの配列を返します。テーブルの配列内での順序は、Mobile Link による SQL のロー・ハンドリングでの順序と同じで、参照整合性違反を防ぐ最適な順序になります。データ・ソースがリレーショナル・データベースの場合は、このテーブル順序を使用してください。

戻り値

現在の同期の UploadedTableData オブジェクトの配列配列内でのテーブルの順序は、クライアントのアップロード順と同じです。

参照

- 「UploadData インタフェース」 607 ページ
- 「UploadedTableData インタフェース」 609 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

handle_UploadData 同期イベントに対して HandleUpload というメソッドを使用するものとします。次の例では getUploadedTables メソッドを使用して、現在のアップロード・トランザクションの UploadedTableData インスタンスを返します。

```
// The method used for the handle_UploadData event.

public void handleUpload(UploadData ud)
    throws SQLException, IOException
{
    UploadedTableData tables[] = ud.getUploadedTables();
    //...
}
```

UploadedTableData インタフェース

構文

public **ianywhere.ml.script.UploadedTableData**

備考

ダイレクト・ロー・ハンドリングアップロードで使用するテーブル操作をカプセル化します。UploadedTableData インスタンスを使用して、単一アップロード・トランザクションに対するテーブルの挿入、更新、削除操作を取得できます。UploadedTableData.getInserts、UploadedTableData.getUpdates、UploadedTableData.getDeletes メソッドを使用して、標準 JDBC java.sql.ResultSet オブジェクトを返します。

java.sql.ResultSet と java.sql.ResultSetMetaData の詳細については、Java SDK マニュアルを参照してください。

UploadedTableData.getMetaData メソッドを使用するか、getInserts、getUpdates、getDeletes によって返された結果セットを使用してテーブル・メタデータにアクセスできます。削除の結果セットには、テーブルのプライマリ・キー・カラムのみが含まれています。

参照

- 「UploadData インタフェース」 607 ページ
- 「handle_UploadData 接続イベント」 477 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

メンバ

ianywhere.ml.script.UploadedTableData のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getDeletes メソッド」 610 ページ
- 「getInserts メソッド」 611 ページ
- 「getUpdates メソッド」 612 ページ
- 「getName メソッド」 613 ページ
- 「getMetaData メソッド」 613 ページ

例

「UploadData インタフェース」 607 ページを参照してください。

getDeletes メソッド

構文

public java.sql.ResultSet **getDeletes( )**

備考

Mobile Link クライアントによってアップロードされた削除操作を表す java.sql.ResultSet オブジェクトを返します。結果セットには、削除されたローのプライマリ・キー値が含まれています。

戻り値

Mobile Link クライアントによってアップロードされた削除操作を表す java.sql.ResultSet オブジェクト

参照

- 「UploadedTableData インタフェース」 609 ページ
- 「handle_UploadData 接続イベント」 477 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

リモート・クライアントには remoteOrders というテーブルがあるものとします。次の例は DownloadTableData.getDeletes メソッドを使用して、削除されたローの結果セットを取得します。この場合、削除結果セットには 1 つのプライマリ・キー・カラムが含まれています。

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

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData for the remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");

    // Get deletes uploaded by the MobiLink client.
    java.sql.ResultSet delete_rs = remoteOrdersTable.getDeletes();
```



```
    while (delete_rs.next()) {
        // Get primary key values for deleted rows.
        int deleted_id = delete_rs.getInt(1);

        // ...
    }
    delete_rs.close();
}
```

getInserts メソッド

構文

public java.sql.ResultSet **getInserts( )**

備考

Mobile Link クライアントによってアップロードされた挿入操作を表す java.sql.ResultSet オブジェクトを返します。各挿入は結果セットの 1 つのローで表されています。

戻り値

Mobile Link クライアントによってアップロードされた挿入操作を表す java.sql.ResultSet オブジェクト

参照

- 「UploadedTableData インタフェース」 609 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

リモート・クライアントには remoteOrders というテーブルがあるものとします。次の例は DownloadTableData.getInserts メソッドを使用して、挿入されたローの結果セットを取得します。このコードは、現在のアップロード・トランザクションの各ローに対する発注額を取得します。

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

// The method used for the handle_UploadData event
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");

    // Get inserts uploaded by the MobiLink client.
    java.sql.ResultSet rs = remoteOrdersTable.getInserts();
    while (rs.next()) {
        // get the uploaded order_amount
        double order_amount = rs.getDouble("order_amount");

        // ...
    }
    rs.close();
}
```

getUpdates メソッド

構文

public UpdateResultSet **getUpdates( )**

備考

Mobile Link クライアントによってアップロードされた更新操作を表す UpdateResultSet オブジェクトを返します。各更新は、すべてのカラム値を含む 1 つのローで表されています。UpdateResultSet は java.sql.ResultSet を拡張して、Mobile Link での競合検出用の特別なメソッドを追加しています。

戻り値

Mobile Link クライアントによってアップロードされた更新操作を表す UpdateResultSet オブジェクト

参照

- 「UploadedTableData インタフェース」 609 ページ
- 「UpdateResultSet」 605 ページ
- 「handle_UploadData 接続イベント」 477 ページ
- 「ダイレクト・アップロードでの競合の処理」 686 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

リモート・クライアントには remoteOrders というテーブルがあるものとします。次の例は UploadedTableData.getUpdates メソッドを使用して、更新されたローの結果セットを取得します。このコードは各ローの発注額を取得します。

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

// the method used for the handle_UploadData event
public void HandleUpload(UploadData ut)
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");

    // Get inserts uploaded by the MobiLink client.
    java.sql.ResultSet rs = remoteOrdersTable.getUpdates();
    while (rs.next()) {
        // Get the uploaded order_amount.
        double order_amount = rs.getDouble("order_amount");

        // ...
    }
    rs.close();
}
```


getName メソッド

構文

public java.lang.String **getName( )**

備考

UploadedTableData インスタンスのテーブル名を返します。getMetaData メソッドによって返される java.sql.ResultSetMetaData インスタンスを使用して、テーブル名にアクセスすることもできます。

戻り値

UploadedTableData インスタンスのテーブル名

参照

- 「UploadedTableData インタフェース」 609 ページ
- UploadedTableData 「getMetaData メソッド」 613 ページ
- 「handle_UploadData 接続イベント」 477 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

例

次の例は、単一アップロード・トランザクションのアップロードされた各テーブルの名前を取得します。

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

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ud) {
    throws SQLException, IOException
{
    int i;

    // Get UploadedTableData instances.
    UploadedTableData tables[] = ud.getUploadedTables();

    for (i=0; i<tables.length; i+=1) {
        // Get the table name.
        String table_name = tables[i].getName();

        // ...
    }
}
```

getMetaData メソッド

構文

public java.sql.ResultSetMetaData **getMetaData( )**

備考

UploadedTableData インスタンスのメタデータを取得します。メタデータは標準 java.sql.ResultSetMetaData インスタンスです。

ResultSetMetaData にカラム名情報を含める場合は、カラム名を送信するためのクライアント拡張オプションを指定してください。

java.sql.ResultSetMetaData の詳細については、Java SDK マニュアルを参照してください。

戻り値

UploadedTableData インスタンスのメタデータ

参照

- dbmlsync：「SendColumnNames (scn) 拡張オプション」『Mobile Link - クライアント管理』
- Ultra Light：「Send Column Names 同期パラメータ」『Ultra Light データベース管理とリファレンス』

例

次の例は、remoteOrders というアップロードされたテーブルの java.sql.ResultSetMetaData インスタンスを取得します。このコードは ResultSetMetaData.getColumnCount と getColumnLabel メソッドを使用して、カラム名のリストをコンパイルします。

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

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {
    throws SQLException, IOException
{
    // Get an UploadedTableData instance representing the remoteOrders table.
    UploadedTableData remoteOrdersTable = ut.getUploadedTableByName("remoteOrders");

    // get inserts uploaded by the MobiLink client
    java.sql.ResultSet rs = remoteOrdersTable.getInserts();

    // Obtain the result set metadata.
    java.sql.ResultSetMetaData md = rs.getMetaData();
    String columnHeading = "";

    // Compile a list of column names.
    for (int c=1; c <= md.getColumnCount(); c += 1) {
        columnHeading += md.getColumnLabel();

        if (c < md.getColumnCount()) {
            columnHeading += ", ";
        }
    }
    //...
}
```

この場合、HandleUpload というメソッドが handle_UploadData 同期イベントを処理します。

参照

- 「UploadedTableData インタフェース」 609 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ
- 「SendColumnNames (scn) 拡張オプション」 『Mobile Link - クライアント管理』
- 「handle_UploadData 接続イベント」 477 ページ

.NET での同期スクリプトの作成

目次

.NET 同期論理の概要

Mobile Link は、同期スクリプトを作成するためのプログラミング言語として Visual Studio をサポートしています。Mobile Link スクリプトを .NET で作成する場合は、有効な .NET アセンブリを作成できる言語であれば、どれでも使用できます。特に、以下の言語がテスト済みで、文書化されています。

- C#
- Visual Basic .NET
- C++

.NET 同期論理には、SQL 論理と同じ機能を持たせることができます。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できるのと同様に、.NET メソッドを呼び出すことができます。.NET メソッドは、SQL 文字列を Mobile Link に返すことができます。

この項では、C#、Visual Basic .NET、C++ 用の .NET 同期論理を設定、開発、実行する方法について説明します。また、サンプル・アプリケーションと .NET 用 Mobile Link サーバ API リファレンスも含まれています。

参照

- 「チュートリアル：.NET 同期論理の使用」 『Mobile Link - クイック・スタート』
- 「サーバ側の同期論理の作成オプション」 『Mobile Link - クイック・スタート』
- 「同期スクリプトの作成」 327 ページ

.NET 同期論理の設定

同期スクリプトを .NET で実装するときには、アセンブリに含まれるパッケージ、クラス、メソッドの場所を Mobile Link に指示する必要があります。

♦ **.NET を使用して同期スクリプトを実装するには、次の手順に従います。**

1. 1 つまたは複数の独自クラスを作成します。必要な同期イベントごとに、メソッドを作成します。これらのメソッドは、パブリックにしてください。

   メソッドの詳細については、「メソッド」 624 ページを参照してください。

   非静的メソッドを持つ各クラスには、パブリック・コンストラクタが必要です。Mobile Link サーバは、各クラスのメソッドが接続のために初めて呼び出されるときに、そのクラスを自動的にインスタンス化します。

   「コンストラクタ」 623 ページを参照してください。

2. 1 つまたは複数のアセンブリを作成します。コンパイル中、独自の .NET メソッドで利用するために、Mobile Link サーバ API クラスのレポジトリを含む *iAnywhere.MobiLink.Script.dll* を参照します。*iAnywhere.MobiLink.Script.dll* は、*install-dir¥Assembly¥v2* にあります。

   クラスをコンパイルするには、コマンド・ラインを使用するか、Visual Studio や他の .NET 開発環境を使用することができます。

   「.NET 用 Mobile Link サーバ API リファレンス」 636 ページを参照してください。

3. プロジェクトをコンパイルします。

   たとえば、次のようにして Visual Studio からコンパイルします。

   a. VS.NET の **[プロジェクト]** - **[既存項目の追加]** を選択します。

   b. *iAnywhere.MobiLink.Script.dll* を探します。

      [開く] リストで **[リンク ファイル]** を選択します。

      > **注意**
      > Visual Studio の場合は、常に上記の [リンク ファイル] を使用します。*iAnywhere.MobiLink.Script.dll* を参照するために、[参照の追加] オプションを使用しないでください。このオプションは、クラス・アセンブリとして同じ物理ディレクトリに *iAnywhere.MobiLink.Script.dll* を複製するため、Mobile Link サーバにとって問題となります。

   c. アセンブリを構築するには、**[ビルド]** メニューを使用します。

   次のように入力して、コマンド・ラインからコンパイルすることもできます。

   dll-path を *iAnywhere.MobiLink.Script.dll* へのパスに置き換えます。たとえば、C# の場合は次のようになります。

   ```
   csc /out:dll-pathout.dll /target:library /reference:dll-pathiAnywhere.MobiLink.Script.dll sync_v1.cs
   ```

4. 統合データベース内の Mobile Link システム・テーブルで、各同期スクリプトについて、呼び出すパッケージ、クラス、メソッドの名前を指定します。スクリプトの各バージョンにつき使用できるクラスは 1 つだけです。

   たとえば、ml_add_dnet_connection_script ストアド・プロシージャまたは ml_add_dnet_table_script ストアド・プロシージャを使用して、この情報を Mobile Link システム・テーブルに追加できます。次の SQL 文は、SQL Anywhere データベース内で実行すると、authenticate_user 接続レベル・イベントが発生するたびに myNamespace.myClass.myMethod を実行するように指定します。

   ```
   CALL ml_add_dnet_connection_script(
      'version1',
      'authenicate_user',
      'myNamespace.myClass.myMethod'
   )
   ```

   注意
   完全に修飾されたメソッド名では、大文字と小文字が区別されます。

   このプロシージャ・コールの結果として、ml_script システム・テーブルの script_language カラムに、**dnet** という語が含まれます。script カラムにはパブリックな .NET メソッドの修飾名が含まれます。

   「ml_add_dnet_connection_script システム・プロシージャ」 703 ページと「ml_add_dnet_table_script システム・プロシージャ」 704 ページを参照してください。

   この情報を追加するには Sybase Central を使用する方法もあります。

   「スクリプトの追加と削除」 345 ページを参照してください。

5. アセンブリをロードするよう Mobile Link サーバに指示し、CLR を起動します。mlsrv11 コマンド・ラインでオプションを使用して、これらのアセンブリの場所を Mobile Link に指示します。2 つのオプションから選択できます。

   - **-sl dnet (-MLAutoLoadPath) を使用**　このオプションは、アプリケーションのベース・ディレクトリへの特定のパスを設定し、そのディレクトリ内のすべてのプライベート・アセンブリをロードします。ほとんどの場合、このオプションを使用してください。たとえば、*dll-path* に保存されたすべてのアセンブリをロードするには、次を入力します。

     ```
     mlsrv11 -c "dsn=consolidated1" -sl dnet(-MLAutoLoadPath=dll-path)
     ```

     -MLAutoLoadPath オプションを使用すると、イベント・スクリプトの完全に修飾されたメソッド名を入力するときに、ドメインを指定できません。

     「アセンブリのロード」 631 ページと「-sl dnet オプション」 92 ページを参照してください。

   - **-sl dnet (-MLDomConfigFile) を使用**　このオプションを使用するには、ドメインとアセンブリの設定を含む設定ファイルが必要です。このオプションを使用するのは、共有アセンブリがある場合、ディレクトリのすべてのアセンブリをロードする必要がない場合、またはその他の理由で設定ファイルを使用する必要がある場合です。

共有アセンブリのロードの詳細については、「アセンブリのロード」 631 ページを参照してください。mlsrv11 オプションの -sl dnet の詳細については、「-sl dnet オプション」 92 ページを参照してください。

注意
-MLAutoLoadPath オプションまたは -MLDomConfigFile オプションを使用できますが、両方は使用できません。

.NET 同期論理の作成

.NET 同期論理を作成するには、Mobile Link イベントの知識、.NET に関する若干の知識、.NET 用 Mobile Link サーバ API に関するある程度の知識が必要です。

API の完全な説明については、「.NET 用 Mobile Link サーバ API リファレンス」 636 ページを参照してください。

.NET 同期論理は、ステータス情報の管理と、アップロード・イベントとダウンロード・イベント関連の論理の実装に使用できます。たとえば、.NET で作成された begin_synchronization スクリプトを使用すると、Mobile Link ユーザ名を変数に格納できます。同期処理中に後で呼び出されるスクリプトは、この変数にアクセスできます。また、.NET は、コミットの実行前または実行後に統合データベースのローにアクセスするために使用できます。

.NET を使用すると、統合データベースへの依存度も減少します。また、統合データベースを新バージョンにアップグレードしたり、別のデータベース管理システムに切り替えたりする場合も、動作に与える影響が少なくて済みます。

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、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・ハンドリングでは、Java または .NET 用 Mobile Link サーバ API の特別なクラスを使用して、同期対象のデータに直接アクセスします。

「ダイレクト・ロー・ハンドリング」 679 ページを参照してください。

クラス・インスタンス

Mobile Link サーバは、クラスをデータベース接続レベルでインスタンス化します。非静的 .NET メソッドをあるイベントに対して作成した場合、そのイベントに達すると、現在のデータベース接続でクラスがインスタンス化されていなければ、Mobile Link サーバが自動的にクラスをインスタンス化します。

「コンストラクタ」 623 ページを参照してください。

注意
1 つのスクリプト・バージョンの接続レベル・イベントまたはテーブルレベル・イベントに直接関連するすべてのメソッドは、同じクラスに属している必要があります。

データベース接続ごとに、インスタンス化されたクラスはその接続が終了するまで持続します。したがって、連続する複数の同期セッションで、同じインスタンスを使用することができます。明示的にクリアされないかぎり、パブリック変数またはプライベート変数内の情報は、同じ接続で発生するすべての同期を通して持続されます。

また、静的なクラスや変数も使用できます。この場合、値は同じドメインのすべての接続を通して使用できます。

統合データベースへの接続が終了した場合にのみ、Mobile Link サーバはクラス・インスタンスを自動的に削除します。

トランザクション

.NET メソッドには、トランザクションに関する通常のルールが適用されます。データベース・トランザクションの開始と継続期間は、同期処理に重要になります。トランザクションの開始と終了は、Mobile Link サーバのみが行います。.NET メソッド内の同期接続でトランザクションを明示的にコミットまたはロールバックすると、同期処理の整合性違反になり、エラーが発生することがあります。

これらのルールは、Mobile Link サーバによって作成されるデータベース接続、特に、メソッドから返される SQL 文にのみ適用されます。

SQL データ型と .NET データ型

次の表は、Mobile Link スクリプト・パラメータの SQL データ型とそれに対応する .NET データ型を示します。

SQL データ型	.NET データ型
VARCHAR	string
CHAR	string
INTEGER	int
BINARY	byte [ ]
TIMESTAMP	DateTime
INOUT INTEGER	ref int
INOUT VARCHAR	ref string
INOUT CHAR	ref string
INOUT BYTEARRAY	ref byte [ ]

コンストラクタ

クラスのコンストラクタはパラメータなしにするか、1 つの iAnywhere.MobiLink.Script.DBConnectionContext をパラメータにできます。次に例を示します。

```
public ExampleClass(iAnywhere.MobiLink.Script.DBConnectionContext cc)
```

または

```
public ExampleClass()
```

渡される同期コンテキストは、Mobile Link サーバが現在のユーザの同期に使用している接続です。

DBConnectionContext.GetConnection メソッドは、Mobile Link が現在のユーザの同期に使用しているのと同じデータベース接続を返します。この接続で文を実行することはできますが、トランザクションのコミットやロールバックは行わないでください。トランザクションは Mobile Link サーバによって管理されます。

Mobile Link サーバでは、コンストラクタがある場合、iAnywhere.MobiLink.Script.DBConnectionContext パラメータを持つコンストラクタが使用されます。コンストラクタがない場合は、void コンストラクタが使用されます。

「DBConnectionContext インタフェース」 639 ページを参照してください。

メソッド

通常は、同期イベントごとにメソッドを 1 つずつ実装します。これらのメソッドは、パブリックにしてください。プライベート・メソッドの場合、Mobile Link サーバでは使用できず、その存在を認識できません。

統合データベース内の ml_script テーブルで指定されている名前と一致していれば、メソッド名は重要ではありません。ただし、このマニュアルの例では、メソッド名は Mobile Link イベント名と同じです。これは、.NET コードを読みやすくするためです。

メソッドのシグニチャは、そのイベント用スクリプトのシグニチャと一致していなければなりません。ただし、パラメータ・リストの最後にパラメータ値が必要でない場合は、リストをトランケートできます。パラメータを渡すとオーバヘッドが生じる可能性があるため、必要なパラメータのみを受け入れてください。

ただし、メソッドはオーバロードできません。ml_script システム・テーブルには、クラスごとにメソッド・プロトタイプが 1 つだけ格納されます。

メソッドの登録

メソッドを作成したら、それを登録します。メソッドを登録すると、統合データベースの Mobile Link システム・テーブル内にメソッドへの参照が作成されて、イベントが発生すると、そのメソッドが呼び出されます。メソッドの登録方法は、同期スクリプトの追加方法と同じです。ただし、SQL スクリプト全体を Mobile Link システム・テーブルに追加する代わりに、修飾されたメソッド名のみを追加します。

「スクリプトの追加と削除」 345 ページを参照してください。

戻り値

SQL ベースのアップロードまたはダウンロードに対して呼び出されるメソッドは、有効な SQL 言語の文を返さなければなりません。これらのメソッドの戻り値は String 型にします。他の戻り値の型は使用できません。

他のすべてのスクリプトの戻り値の型は、string または void にします。他の型は使用できません。戻り値の型が NULL ではなく文字列の場合、Mobile Link サーバはその文字列に有効な SQL 文が含まれているとみなして、この文を通常の SQL 言語による同期スクリプトと同様に統合データベース内で実行します。通常、メソッドは文字列を返しますが、その戻り値によってデータベースに対して SQL 文を実行しない場合は NULL を返すことができます。

ユーザ定義起動クラス

サーバの起動時に自動的にロードされる起動クラスを定義できます。この機能の目的は、最初の同期の前に Mobile Link サーバが CLR を起動する時点で実行される .NET コードを記述できるようにすることです。つまり、サーバ・インスタンスで、最初のユーザ同期要求の前に、接続の作成またはデータのキャッシュを実行できます。

この操作を行うには、mlsrv11 -sl dnet オプションの MLStartClasses オプションを使用します。たとえば、次に示すのは mlsrv11 コマンド・ラインの一部です。mycl1 と mycl2 が起動クラスとしてロードされます。

```
-sl dnet(-MLStartClasses=MyNameSpace.MyClass.mycl1,MyNameSpace.MyClass.mycl2)
```

クラスはリスト内の順序でロードされます。同じクラスがリストに 2 回以上指定されている場合は、複数のインスタンスが作成されます。

すべての起動クラスはパブリックでなければなりません。また、引数を 1 つも受け付けないか、または MobiLink.Script.ServerContext データ型の引数を 1 つ受け付けるパブリック・コンストラクタが必要です。

ロードされた起動クラスの名前は、「.NET 起動クラス *classname* がロードされました。」というメッセージとともに Mobile Link ログに出力されます。

.NET CLR の詳細については、「-sl dnet オプション」 92 ページを参照してください。

サーバ起動時に構築される起動クラスを表示する方法については、「GetStartClassInstances メソッド」 659 ページを参照してください。

例

次に示すのは、起動クラスのテンプレートです。これは、イベントを処理してデータベース接続を確立するデーモン・スレッドを起動します(すべての起動クラスがスレッドの作成を必要とするわけではありませんが、スレッドを作成する場合はデーモン・スレッドでなければなりません)。

```
using System;
using System.IO;
using System.Threading;
using iAnywhere.MobiLink.Script;

namespace TestScripts {
   public class MyStartClass {
      ServerContext    _sc;
      bool             _exit_loop;
      Thread           _thread;
      OdbcConnection   _conn;

      public MyStartClass(ServerContext sc) {
```

```
            // Perform setup first so that an exception
            // causes MobiLink startup to fail.
            _sc = sc;

            // Create connection for use later.
            _conn = _sc.makeConnection();
            _exit_loop = false;
            _thread = new Thread(new ThreadStart(run)) ;
            _thread.IsBackground = true;
            _thread.Start();
        }

        public void run() {
            ShutdownCallback callback = new ShutdownCallback(shutdownPerformed);
            _sc.ShutdownListener += callback;

            // run() can't throw exceptions.
            try {
                handlerLoop();
                _conn.close();
                _conn = null;
            }
            catch(Exception e) {
                // Print some error output to the MobiLink log.
                Console.Error.Write(e.ToString());

                // There is no need to be notified of shutdown.
                _sc.ShutdownListener -= callback;

                // Ask server to shut down so this fatal error can be fixed.
                _sc.Shutdown();
            }

            // Shortly after return, this thread no longer exists.
            return;
        }

        public void shutdownPerformed(ServerContext sc) {
            // Stop the event handler loop.
            try {
                _exit_loop = true;

                // Wait a maximum of 10 seconds for thread to die.
                _thread.Join(10*1000);
            }
            catch(Exception e) {
                // Print some error output to the MobiLink log.
                Console.Error.Write(e.ToString());
            }
        }

        private void handlerLoop() {
            while (!_exit_loop) {
                // Handle events in this loop.
                Thread.Sleep(1*1000);
            }
        }
    }
}
```


.NET からの情報の出力

System.Console を使用して、Mobile Link ログに情報を出力する文を .NET メソッドに追加することもできます。これにより、クラスの進行状況と動作を追跡できます。

パフォーマンスに関するヒント
この方法で Mobile Link のログに情報を出力すると、モニタ・ツールとして活用できますが、運用環境ではおすすめしません。

これと同じ方法を利用して、任意の同期情報のログを取ったり、スクリプトの使用方法に関する統計情報を収集したりできます。

.NET での Mobile Link サーバ・エラーの処理

ログをスキャンするだけでは不十分な場合は、プログラムによってアプリケーションをモニタリングできます。たとえば、特定のタイプのメッセージを電子メールで送信できます。

ログに出力される各エラー・メッセージまたは各警告メッセージを表すクラスに渡されるメソッドを作成することも可能です。この方法は、Mobile Link サーバをモニタおよび監査するのに役立ちます。

次のコードは、すべてのエラー・メッセージ用にリスナをインストールし、その情報を StreamWriter に出力します。

```
class TestLogListener {
    public TestLogListener(StreamWriter output_file) {
        _output_file    = output_file;
    }

    public void errCallback(ServerContext sc, LogMessage lm) {
        string type;
        string user;

        if (lm.Type == LogMessage.MessageType.ERROR) {
            type = "ERROR";
        } else if (lm.Type==LogMessage.MessageType.WARNING) {
            type = "WARNING";
        }
        else {
            type = "INVALID TYPE!!";
        }
        if (lm.User == null) {
            user = "null";
        }
        else {
            user = lm.User;
        }

        _output_file.WriteLine("Caught msg type=" + type
            + " user=" + user
            + " text=" + lm.Text);
        _output_file.Flush();
    }
    StreamWriter _output_file;
}
```

次のコードは、TestLogListener を登録します。クラス・コンストラクタや同期スクリプトなど、ServerContext にアクセスできる場所からこのコードを呼び出してください。

```
// ServerContext serv_context;
TestLogListener etll = new TestLogListener(log_listener_file);
serv_context.ErrorListener += new LogCallback(etll.errCallback);
```

参照

- 「LogCallback デリゲート」 657 ページ
- 「ServerContext インタフェース」 659 ページの ErrorListener と WarningListener
- 「LogMessage クラス」 657 ページ
- 「MessageType 列挙体」 657 ページ

.NET 同期論理のデバッグ

次の手順では、Visual Studio を使用して .NET スクリプトをデバッグする 1 つの方法を説明します。

♦ **.NET スクリプトをデバッグするには、次の手順に従います。**

1. 次のいずれかの方法を使用して、デバッグ情報をオンにした状態でコードをコンパイルします。
   - csc コマンド・ラインで、/debug+ オプションを設定します。
   - Microsoft Visual Studio の設定を使用してデバッグ出力を設定します。
     - **[ファイル] - [ビルド] - [構成マネージャ]** を選択します。

       [アクティブ ソリューション構成] リストで **[デバッグ]** を選択します。
     - アセンブリを構築します。
2. ソース・ファイルを含む Visual Studio の実行中のインスタンスを閉じます。
3. 次の手順では、新しい Visual Studio インスタンスを起動して、Mobile Link サーバと使用している .NET 同期スクリプトをデバッグします。コマンド・ライン・オプションを使用して Visual Studio を起動し、Mobile Link サーバをデバッグします。
   - コマンド・プロンプトで、Visual Studio インストール環境の *Common7¥IDE* サブディレクトリに移動します。
   - /debugexe オプションを使用して、devenv (Visual Studio IDE) を起動します。

     たとえば、次のコマンドを実行して、Mobile Link サーバをデバッグします。接続文字列と .NET アセンブリをロードするオプションを含めて、mlsrv11 オプションを指定してください。

     32 ビット Windows 環境の場合：

     ```
     devenv /debugexe %sqlany11%¥bin32¥mlsrv11.exe -c ...
     ```

     64 ビット Windows 環境の場合：

     ```
     devenv /debugexe %sqlany11%¥bin64¥mlsrv11.exe -c ...
     ```


Visual Studio が起動し、[ソリューション エクスプローラ] ウィンドウに *mlsrv11.exe* が表示されます。

4. .NET コードをデバッグするために Microsoft Visual Studio を次のように設定します。
   - Visual Studio の [ソリューション エクスプローラ] ウィンドウで *mlsrv11.exe* を右クリックし、[プロパティ] を選択します。
   - [デバッガのタイプ] を [自動] から [混合] または [マネージのみ] に変更して、Visual Studio が .NET 同期スクリプトのみをデバッグするようにします。
5. 関連する .NET ソース・ファイルを開き、ブレーク・ポイントを設定します。

   注意 : mlsrv11 ソリューションでソース・ファイルを個別に開きます。元のソリューションやプロジェクト・ファイルは開かないでください。
6. [デバッグ] メニューまたは [F5] キーを押して Mobile Link を起動します。

   プロンプトが表示されたら、*mlsrv11.sln* を保存します。

   [シンボル情報なし] ウィンドウが表示された場合は、**[OK]** をクリックしてデバッグを続けます。デバッグしているのは、Mobile Link が呼び出す管理対象の .NET 同期スクリプトであり、Mobile Link サーバ本体ではありません。
7. 同期を実行します。この結果、ブレークポイントのあるコードが Mobile Link によって実行されます。

.NET の同期の方法

この項では、一般的な .NET の同期タスクに取り組む場合に使用できる方法について説明します。

ローのアップロードまたはダウンロード

.NET を介したローのアップロードまたはダウンロードの詳細については、「ダイレクト・ロー・ハンドリング」 679 ページを参照してください。

共有アセンブリのロード

この項では、.NET アセンブリをロードするためのオプションと共有アセンブリをロードする処理について説明します。

アセンブリのロード

.NET アセンブリは、タイプ、メタデータ、プログラム・コードのパッケージです。.NET アプリケーションでは、すべてのコードがアセンブリになければなりません。アセンブリ・ファイルの拡張子は *.dll* または *.exe* です。

アセンブリには、次の種類があります。

- **プライベート・アセンブリ**　ファイル・システム内のファイル。
- **共有アセンブリ**　グローバル・アセンブリ・キャッシュにインストールされるアセンブリ。

Mobile Link では、クラスを含むアセンブリが指定されないと、クラスをロードしてそのクラスのメソッドを呼び出すことができません。指定する必要があるのは、Mobile Link が直接呼び出すアセンブリのみです。このアセンブリによって、その他の必要なアセンブリが呼び出されます。

たとえば、Mobile Link が MyAssembly を呼び出すと、MyAssembly が UtilityAssembly と NetworkingUtilsAssembly を呼び出します。この場合は、Mobile Link が MyAssembly だけを探すように設定します。

Mobile Link では、次の方法でアセンブリをロードできます。

- **-sl dnet (-MLAutoLoadPath) を使用**　このオプションは、プライベート・アセンブリに対してのみ有効です。このオプションは、アプリケーションのベース・ディレクトリへのパスを設定し、そのディレクトリ内のすべてのアセンブリをロードします。

  -MLAutoLoadPath オプションを使用すると、イベント・スクリプトの完全に修飾されたメソッド名を入力するときに、ドメインを指定できません。

  -MLAutoLoadPath でパスとディレクトリを指定すると、次のアクションが実行されます。

  ○ このパスがアプリケーション・ベース・パスとして設定される

  ○ 指定したディレクトリで *.dll* または *.exe* の付くすべてのファイルのすべてのクラスがロードされる

  ○ 1 つのアプリケーション・ドメインが作成され、ドメインが指定されていないすべてのユーザ・クラスがそのドメインにロードされる

  このオプションでは、グローバル・アセンブリ・キャッシュ内のアセンブリを直接呼び出すことはできません。これらの共有アセンブリを呼び出すには、-MLDomConfigFile を使用します。

- **-sl dnet (-MLDomConfigFile) を使用**　このオプションは、プライベート・アセンブリと共有アセンブリの両方に使用できます。このオプションを使用するには、ドメインとアセンブリの設定を含む設定ファイルが必要です。このオプションは、共有アセンブリがある場合、ア

プリケーション・ベース・パス内のすべてのアセンブリをロードする必要がない場合、またはその他の理由で設定ファイルを使用する必要がある場合に使用してください。

このオプションを使用すると、Mobile Link は指定されたドメイン設定ファイル内の設定を読み込みます。ドメイン設定ファイルには、1 つまたは複数の .NET ドメインの設定が入っています。このファイルに複数のドメインが記述されている場合は、1 番目に指定されているドメインがデフォルト・ドメインとして使用されます(デフォルト・ドメインは、指定されたドメインがスクリプトにない場合に使用されます)。

Mobile Link はアセンブリをロードするときに、まずプライベート・アセンブリとしてロードし、次にグローバル・アセンブリ・キャッシュからアセンブリをロードしようとします。プライベート・アセンブリは、アプリケーション・ベース・ディレクトリにあります。共有アセンブリはグローバル・アセンブリ・キャッシュからロードされます。

-MLDomConfigFile オプションでは、ドメイン設定ファイルで指定されているアセンブリのみが、イベント・スクリプトから直接呼び出せます。

サンプルのドメイン設定ファイル

mlDomConfig.xml というサンプルのドメイン設定ファイルが、Mobile Link とともにインストールされます。独自のファイルを最初から作成するか、要件に合わせてサンプルを編集できます。サンプル・ファイルは、SQL Anywhere の次のパスにあります。

MobiLink¥setup¥dnet¥mlDomConfig.xml

サンプルのドメイン設定ファイル *mlDomConfig.xml* の内容は次のとおりです。

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="iAnywhere.MobiLink.mlDomConfig"
xsi:schemaLocation='iAnywhere.MobiLink.mlDomConfig mlDomConfig.xsd' xmlns:xsi='http://
www.w3.org/2001/XMLSchema-instance' >
   <domain>
      <name>SampleDomain1</name>
      <appBase>C:¥scriptsDir</appBase>
      <configFile></configFile>
      <assembly name="Assembly1" />
      <assembly name="Assembly2" />
   </domain>

   <domain>
      <name>SampleDomain2</name>
      <appBase>¥Dom2assembly</appBase>
      <configFile>¥Dom2assembly¥AssemblyRedirects.config</configFile>
      <assembly name="Assembly3" />
      <assembly name="Assembly4" />
   </domain>
</config>
```

次に、*mlDomConfig.xml* の内容について説明します。

- **name**　イベント・スクリプトでドメインを指定するときに使用するドメイン名です。"DomainName:Namespace.Class.Method" というフォーマットのイベント・スクリプトを使用するには、ドメイン設定ファイル内に DomainName ドメインが必要です。

  ドメイン名は 1 つ以上指定してください。

- **appBase**　ドメインがアプリケーション・ベース・ディレクトリとして使用するディレクトリです。すべてのプライベート・アセンブリは、このディレクトリに基づいて .NET CLR によってロードされます。appBase は必ず指定してください。
- **configFile**　ドメインに使用する .NET アプリケーション設定ファイルです。ここは空白でもかまいません。通常は、この部分を使用してアセンブリのバインドとロードのデフォルトの動作を変更します。アプリケーション設定ファイルの詳細については、.NET のマニュアルを参照してください。
- **assembly**　イベント・スクリプト内の型の参照を解決するときに Mobile Link がロードして検索するアセンブリの名前です。アセンブリは 1 つ以上指定してください。アセンブリが複数のドメインで使用されている場合は、ドメインごとに 1 つのアセンブリを指定してください。プライベート・アセンブリの場合は、ドメインのアプリケーション・ベース・ディレクトリになければなりません。

mlsrv11 オプションの -sl dnet の詳細については、「-sl dnet オプション」 92 ページを参照してください。

.NET 同期のサンプル

このサンプルは、.NET 同期論理を使用して authenticate_user イベントを処理する方法を表示するように既存のアプリケーションを変更します。このサンプルは、*AuthUser.cs* という名前の authenticate_user 用の C# スクリプトを作成します。このスクリプトは、user_pwd_table というテーブル内でユーザのパスワードを検索し、そのパスワードに基づいてユーザを認証します。

♦ **.NET 同期スクリプトを作成するには、次の手順に従います。**

1. テーブル user_pwd_table をデータベースに追加します。Interactive SQL で次の SQL 文を実行します。

```
CREATE TABLE user_pwd_table (
  user_name  varchar(128) PRIMARY KEY NOT NULL,
  pwd        varchar(128)
)
```

2. ユーザとパスワードをテーブルに追加します。

```
INSERT INTO user_pwd_table VALUES('user1', 'myPwd')
```

3. .NET アセンブリ用のディレクトリを作成します (例：*c:¥mlexample*)。
4. 次の内容の *AuthUser.cs* というファイルを作成します。

「authenticate_user 接続イベント」 377 ページを参照してください。

```
using System;
using iAnywhere.MobiLink.Script;
namespace MLExample {

public class AuthClass {
   private DBConnection  _conn;

   /// AuthClass constructor.
   public AuthClass(DBConnectionContext cc) {
      _conn   = cc.GetConnection();
   }

   /// The DoAuthenticate method handles the 'authenticate_user'
   /// event.
   /// Note: This method does not handle password changes for
   /// advanced authorization status codes.
   public void DoAuthenticate(
      ref int authStatus,
      string user,
      string pwd,
      string newPwd)
   {
      DBCommand   pwd_command = _conn.CreateCommand();
      pwd_command.CommandText = "select pwd from user_pwd_table"
         + " where user_name = ? ";
      pwd_command.Prepare();

      // Add a parameter for the user name.
      DBParameter user_param  = new DBParameter();
      user_param.DbType       = SQLType.SQL_CHAR;

      // Set the size for SQL_VARCHAR.
      user_param.Size     = (uint) user.Length;
```



```
            user_param.Value    = user;
            pwd_command.Parameters.Add(user_param);

            // Fetch the password for this user.
            DBRowReader rr      = pwd_command.ExecuteReader();
            object[] pwd_row    = rr.NextRow();

            if (pwd_row == null) {
                // User is unknown.
                authStatus  = 4000;
            }
            else {
                if (((string) pwd_row[0]) == pwd) {
                    // Password matched.
                    authStatus = 1000;
                }
                else {
                    // Password did not match.
                    authStatus = 4000;
                }
            }
            pwd_command.Close();
            rr.Close();
            return;
        }
    }
```

MLExample.AuthClass.DoAuthenticate メソッドは、authenticate_user イベントを処理します。これはユーザ名とパスワードを受け入れ、検証の成功または失敗を示す認証ステータス・コードを返します。

5. ファイル *AuthUser.cs* をコンパイルします。コンパイルは、コマンド・ラインまたは Visual Studio で実行できます。

   たとえば、次のコマンド・ラインは *AuthUser.cs* をコンパイルし、*example.dll* という名前のアセンブリを *c:¥mlexample* に生成します。

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

6. authenticate_user イベント用の .NET コードを登録します。実行する必要があるメソッドは、ネームスペース MLExample とクラス AuthClass にあります。次の SQL を実行します。

   ```
   CALL ml_add_dnet_connection_script('ex_version', 'authenticate_user',
   'MLExample.AuthClass.DoAuthenticate');
   COMMIT
   ```

7. 次のオプションで Mobile Link サーバを実行します。このオプションによって、Mobile Link が *c:¥myexample* 内のすべてのアセンブリをロードします。

   ```
   -sl dnet (-MLAutoLoadPath=c:¥mlexample)
   ```

これで、ユーザがバージョン ex_version と同期するときに、テーブル user_pwd_table のパスワードで認証されるようになります。

.NET 用 Mobile Link サーバ API リファレンス

この項では、Mobile Link .NET のインタフェースとクラス、これらに関連するメソッド、プロパティ、コンストラクタについて説明します。これらのクラスを使用するには、*install-dir¥Assembly¥v2* にある *iAnywhere.MobiLink.Script.dll* アセンブリを参照してください。

この項では C# について説明しますが、Visual Basic.NET と C++ にも同様の説明が適用されます。

DBCommand インタフェース

構文

interface **DBCommand**
Member of **iAnywhere.MobiLink.Script**

備考

SQL 文またはデータベース・コマンドを表します。DBCommand は更新またはクエリを表すことができます。

例

たとえば、次の C# コードは DBCommand インタフェースを使用して次の 2 つのクエリを実行します。

```
DBCommand stmt = conn.CreateCommand();
stmt.CommandText = "SELECT t1a1, t1a2 FROM table1 ";

DBRowReader rs = stmt.ExecuteReader();
printResultSet(rs);
rs.Close();

stmt.CommandText = "SELECT t2a1 FROM table2 ";
rs = stmt.ExecuteReader();
printResultSet(rs);
rs.Close();
stmt.Close();
```

次の C# サンプルは DBCommand でパラメータを指定して更新を実行します。

```
public void prepare_for_download(
    DateTime last_download,
    String ml_username)
{
    DBCommand cstmt     = conn.CreateCommand();
    cstmt.CommandText   = "CALL myProc(?,?,?,?)";
    cstmt.Prepare();

    DBParameter param   = new DBParameter();
    param.DbType        = SQLType.SQL_CHAR;
    param.Value         = "10000";
    cstmt.Parameters.Add(param);

    param               = new DBParameter();
    param.DbType        = SQLType.SQL_INTEGER;
```



```
        param.Value        = 20000;
        cstmt.Parameters.Add(param);

        param              = new DBParameter();
        param.DbType       = SQLType.SQL_DECIMAL;
        param.Precision    = 5;
        param.Value        = new Decimal(30000);
        cstmt.Parameters.Add(param);

        param              = new DBParameter();
        param.DbType       = SQLType.SQL_TIMESTAMP;
        param.Precision    = 19;
        param.Value        = last_download;
        cstmt.Parameters.Add(param);

        // Execute update
        DBRowReader rset    = cstmt.ExecuteNonQuery();
        cstmt.Close();
    }
```

Prepare メソッド

構文

void **Prepare( )**

備考

CommandText に格納されている SQL 文の実行を準備します。

ExecuteNonQuery メソッド

構文

int **ExecuteNonQuery( )**

備考

non-query の文を実行します。データベース内で SQL 文の影響を受けるローの数を返します。

ExecuteReader メソッド

構文

DBRowReader **ExecuteReader( )**

備考

結果セットを返すクエリ文を実行します。SQL 文が返す結果を取得するための DBRowReader を返します。

Close メソッド

構文

void **Close( )**

備考

現在の SQL 文またはコマンドを終了します。

CommandText プロパティ

構文

string **CommandText**

備考

値は実行する SQL 文です。

Parameters プロパティ

構文

DBParameterCollection **Parameters**

備考

この DBCommand に対する iAnywhere.MobiLink.Script.DBParameterCollection を取得します。

DBConnection インタフェース

構文

interface **DBConnection**
Member of **iAnywhere.MobiLink.Script**

備考

Mobile Link ODBC 接続を表します。

このインタフェースにより、ユーザが作成した同期論理で Mobile Link によって確立された ODBC 接続にアクセスできます。

Commit メソッド

構文

void **Commit( )**

備考

現在のトランザクションをコミットします。

Rollback メソッド

構文

void **Rollback( )**

備考

現在のトランザクションをロールバックします。

Close メソッド

構文

void **Close( )**

備考

現在の接続を閉じます。

CreateCommand メソッド

構文

DBCommand **CreateCommand( )**

備考

この接続で SQL 文またはコマンドを作成します。新しく生成された DBCommand を返します。

DBConnectionContext インタフェース

構文

interface **DBConnectionContext**
Member of **iAnywhere.MobiLink.Script**

備考

現在のデータベース接続に関する情報を取得し、アクセスするためのインタフェース。このインタフェースは、スクリプトを含むクラスのコンストラクタに渡されます。コンテキストがバックグラウンド・スレッドに必要な場合や、接続期間を超えて必要な場合は、ServerContext を使用してください。

コンストラクタの詳細については、「コンストラクタ」 623 ページを参照してください。

> **警告**
> DBConnectionContext インスタンスは、.NET コードに呼び出すスレッド以外で使用しないでください。

GetConnection メソッド

構文

iAnywhere.MobiLink.Script.DBConnection GetConnection( )
Member of **iAnywhere.MobiLink.Script.DBConnectionContext**

備考

Mobile Link 統合データベースへの既存の接続を返します。この接続は、Mobile Link が SQL スクリプトの実行に使用するものと同じです。

この接続は、Mobile Link サーバのこの接続での使用に影響する方法でコミット、終了、または変更しないでください。返される接続は、基本となる Mobile Link 接続の期間中にのみ有効です。接続に対して end_connection イベントが呼び出された後は、その接続を使用しないでください。

フル・アクセス権を持つサーバ接続が必要な場合、ServerContext.makeConnection() を使用してください。

GetDownloadData メソッド

構文

```
DownloadData GetDownloadData();
```

備考

現在の同期に対する DownloadData インスタンスを返します。ダイレクト・ロー・ハンドリング用のダウンロードを作成する場合は、DownloadData インスタンスを使用してください。

戻り値

現在の同期に対する DownloadData インスタンス

例

次の例は、_cc という DBConnectionContext インスタンスがあることを前提としています。


```
// The method used for the handle_DownloadData event.
public void HandleDownload() {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.GetDownloadData();

    // Get an array of tables to set download operations.
    DownloadTableData[] download_tables = my_dd.GetDownloadTables();

    // Get the first table in the DownloadTableData array.
    DownloadTableData my_download_table = download_tables[0];

     // ...
}
```

GetServerContext メソッド

構文

public **iAnywhere.MobiLink.Script.ServerContext.GetServerContext( )**
Member of **iAnywhere.MobiLink.Script.DBConnectionContext**

備考

この Mobile Link サーバの ServerContext を返します。

GetProperties メソッド

構文

NameValueCollection **getProperties( )**

備考

この接続のスクリプト・バージョンに基づいて、この接続のプロパティを返します。プロパティは、ml_property テーブルに格納されます。

詳細については、「ml_property」 753 ページと「ml_add_property システム・プロシージャ」 713 ページを参照してください。

GetRemoteID メソッド

構文

string **GetRemoteID( )**

備考

この接続で現在同期中のデータベースのリモート ID を返します。バージョン 10 以前のリモート・データベースの場合は、Mobile Link ユーザ名を返します。

戻り値

リモート ID

参照

- 「リモート ID」 『Mobile Link - クライアント管理』

GetVersion メソッド

構文

string **getVersion( )**

備考

スクリプト・バージョンの名前を返します。

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

DBParameter クラス

構文

class **DBParameter**
Member of **iAnywhere.MobiLink.Script**

備考

ODBC バウンド・パラメータを表します。

パラメータを指定してコマンドを実行するには、DBParameter が必要です。すべてのパラメータを指定してからコマンドを実行してください。

例

たとえば、次の C# コードは DBCommand でパラメータを指定して更新を実行します。

```
public void handleUpload(UploadData ud) {
   UploadedTableData UTDAdmin = ud.GetUploadedTableByName("Admin");
   IDataReader       AdminIns = UTDAdmin.GetInserts();
   DBCommand         stmt1    = _conn.CreateCommand();
   DBParameter       p_id     = new DBParameter();
   DBParameter       p_data   = new DBParameter();

   stmt1.CommandText         = "INSERT INTO Admin(admin_id,data) VALUES (?,?)";
   p_id.DbType               = SQLType.SQL_BIGINT;
   stmt1.Parameters.Add(p_id);
   p_data.DbType             = SQLType.SQL_VARCHAR;
   p_data.Size               = 30;
   stmt1.Parameters.Add(p_data);
```



```
        stmt1.Prepare();

        while (AdminIns.Read()) {
            p_id.Value = AdminIns.GetInt64(0);
            p_data.Value = AdminIns.GetString(1);
            stmt1.ExecuteNonQuery();
        }
        stmt1.Close();
    }
```

DbType プロパティ

構文

SQLTYPE DbType

備考

値はこのパラメータの SQLType です。

デフォルト値は SQLType.SQL_TYPE_NULL です。

Direction プロパティ

構文

System.Data.ParameterDirection Direction

備考

値はこのパラメータの入出力方向です。

デフォルト値は ParameterDirection.Input です。

IsNullable プロパティ

構文

bool **IsNullable**

備考

値は、このパラメータが NULL かどうかを示します。

デフォルト値は false です。

ParameterName プロパティ

構文

string **ParameterName**

備考

値はこのパラメータの名前です。

デフォルト値は NULL です。

Precision プロパティ

構文

uint **Precision**

備考

値はこのパラメータの 10 進数精度です。SQLType.SQL_NUMERIC パラメータと SQLType.SQL_DECIMAL パラメータにのみ使用します。

デフォルト値は 0 です。

Scale プロパティ

構文

short **Scale**

備考

値はこのパラメータの解決可能な桁数です。SQLType.SQL_NUMERIC パラメータと SQLType.SQL_DECIMAL パラメータにのみ使用します。

デフォルト値は 0 です。

Size プロパティ

構文

uint **Size**

備考

値はこのパラメータのサイズ (バイト数) です。

デフォルト値は DbType から推定されます。

Value プロパティ

構文

object **Value**

備考

値はこのパラメータの値です。

デフォルト値は NULL です。

DBParameterCollection クラス

構文

class **DBParameterCollection**
inherits from **IDataParameterCollection**, **IList**, **ICollection**, **IEnumerable**
Member of **iAnywhere.MobiLink.Script**

備考

DBParameter のコレクション。DBCommand で DBParameterCollection を作成した時点では、DBParameterCollection は空です。DBCommand を実行する前に、適切なパラメータを指定してください。

DBParameterCollection メソッド

構文

DBParameterCollection( )

備考

DBParameter の空リストを作成します。

Contains(string parameterName) メソッド

構文

bool **Contains(string** *parameterName* **)**

備考

指定した名前のパラメータがコレクションに含まれている場合は、true を返します。

パラメータ

- **parameterName**　確認するパラメータの名前

IndexOf(string parameterName) メソッド

構文

int **IndexOf(string** *parameterName* **)**

備考

パラメータのインデックスを返します。指定した名前のパラメータがない場合は、-1 を返します。

パラメータ

- **parameterName**　検索するパラメータの名前

RemoveAt(string parameterName) メソッド

構文

void **RemoveAt(string** *parameterName* **)**

備考

指定した名前のパラメータをコレクションから削除します。

パラメータ

- **parameterName**　削除するパラメータの名前

Add(object value) メソッド

構文

int **Add(object** *value* **)**

備考

指定したパラメータをコレクションに追加します。

パラメータ

- **value**　コレクションに追加する iAnywhere.MobiLink.Script.DBParameter インスタンス

戻り値

コレクションに追加されたパラメータのインデックス

Clear メソッド

構文

void **Clear( )**

備考

すべてのパラメータをコレクションから削除します。

Contains(object value) メソッド

構文

bool **Contains(object** *value* **)**

備考

指定した iAnywhere.MobiLink.Script.DBParameter がこのコレクションに含まれる場合に true を返します。

パラメータ

- **value**　確認する iAnywhere.MobiLink.Script.DBParameter

IndexOf(object value) メソッド

構文

int **IndexOf(object** *value* **)**

備考

このコレクションに含まれる指定した iAnywhere.MobiLink.Script.DBParameter のインデックスを返します。

パラメータ

- **value**　検索する iAnywhere.MobiLink.Script.DBParameter

Insert(int index, object value) メソッド

構文

void **Insert(int** *index***, object** *value* **)**

備考

指定した iAnywhere.MobiLink.Script.DBParameter を、コレクションの指定したインデックス位置に挿入します。

パラメータ

- **value**　挿入する iAnywhere.MobiLink.Script.DBParameter
- **index**　DBParameter を挿入するインデックス位置

Remove(object value) メソッド

構文

void **Remove(object** *value* **)**

備考

指定した iAnywhere.MobiLink.Script.DBParameter をコレクションから削除します。

パラメータ

- **value**　削除する iAnywhere.MobiLink.Script.DBParameter

RemoveAt(int index) メソッド

構文

void **RemoveAt(int index)**

備考

このコレクションから指定したインデックス位置の iAnywhere.MobiLink.Script.DBParameter を削除します。

パラメータ

- **index**　削除する iAnywhere.MobiLink.Script.DBParameter のインデックス位置

CopyTo(Array array, int index) メソッド

構文

void **CopyTo(Array** *array*, **int** *index* **)**

備考

コレクションの内容を、指定したインデックスで始まる特定の配列にコピーします。

パラメータ

- **array**　コレクションの内容のコピー先の配列
- **index**　コレクションの内容のコピーを開始する配列内のインデックス

GetEnumerator メソッド

構文

IEnumerator **GetEnumerator( )**

備考

コレクションの列挙子を返します。

IsFixedSize プロパティ

構文

bool **IsFixedSize**

備考

false を返します。

IsReadOnly プロパティ

構文

bool **IsReadOnly**

備考

false を返します。

Count プロパティ

構文

int **Count**

備考

コレクション内のパラメータ数。

IsSynchronized プロパティ

構文

bool **IsSynchronized**

備考

false を返します。

SyncRoot プロパティ

構文

object **SyncRoot**

備考

コレクションの同期に使用できるオブジェクト。

this[string parameterName] プロパティ

構文

object **this[string** *parameterName* **]**

備考

コレクション内で指定した名前の iAnywhere.MobiLink.Script.DBParameter を取得または設定します。

パラメータ

- **parameterName** 取得または設定する iAnywhere.MobiLink.Script.DBParameter の名前

this[int index] プロパティ

構文

object **this[int** *index* **]**

備考

コレクション内で指定したインデックス位置の iAnywhere.MobiLink.Script.DBParameter を取得または設定します。

パラメータ

- **index** 取得または設定する iAnywhere.MobiLink.Script.DBParameter のインデックス

DBRowReader インタフェース

構文

interface **DBRowReader**
Member of **iAnywhere.MobiLink.Script**

備考

データベースから読み込まれるロー・セットを表します。メソッド DBCommand.executeReader( ) を実行すると、DBRowReader が作成されます。

次のサンプルは C# コード・フラグメントです。このコードは、結果セット内で所定の DBRowReader で表されるローの関数を呼び出します。

```
DBCommand stmt = conn.CreateCommand();
stmt.CommandText = "select intCol, strCol from table1 ";
DBRowReader rset = stmt.ExecuteReader();
object[] values = rset.NextRow();

while (values != null) {
   handleRow((int) values[0], (String) values[1]);
   values = rset.NextRow();
}

rset.Close();
stmt.Close();
```

NextRow メソッド

構文

object[ ] **NextRow( )**

備考

結果セット内の次のローの値を取得して返します。結果セットにローがなくなると、NULL を返します。

「SQLType 列挙体」 663 ページを参照してください。

Close メソッド

構文

void **Close( )**

備考

この MLDBRowReader で使用されるリソースをクリーンアップします。Close( ) を呼び出した後は、この MLDBRowReader を再び使用することはできません。

ColumnNames プロパティ

構文

string[ ] **ColumnNames**

備考

結果セットに含まれるすべてのカラムの名前を取得します。値は、結果セット内のカラム名に相当する文字列の配列です。

ColumnTypes プロパティ

構文

SQLType[ ] **ColumnTypes**

備考

結果セットに含まれるすべてのカラムの型を取得します。値は、結果セット内のカラムの型に対応する SQLTypes の配列です。

DownloadData インタフェース

構文

interface **DownloadData**

備考

ダイレクト・ロー・ハンドリングで使用するダウンロード・データ操作をカプセル化します。DownloadData インスタンスを取得するには、DBConnectionContext の GetDownloadData メソッドを使用します。DownloadTableData インスタンスを返すには、DownloadData の GetDownloadTables メソッドと GetDownloadTableByName メソッドを使用します。

ダウンロード・データは DBConnectionContext を使用して取得できます。begin_synchronization イベントの前または end_download イベントの後にダウンロード・データにアクセスすることはできません。また、DownloadData にアップロード専用同期でアクセスすることもできません。

参照

- 「DownloadTableData インタフェース」 654 ページ
- 「handle_DownloadData 接続イベント」 465 ページ
- DBConnectionContext 「GetDownloadData メソッド」 640 ページ
- 「ダイレクト・ロー・ハンドリング」 679 ページ

GetDownloadTables メソッド

構文

DownloadTableData[ ] **GetDownloadTables( );**

備考

現在の同期のダウンロード・データのすべてのテーブルを含む配列を取得します。このテーブルに対して実行された操作はリモート・データベースに送信されます。

戻り値

ダウンロード・テーブル・データの配列。配列内でのテーブルの順序は、リモートのアップロード順と同じです。

例

次の例では、DownloadData.GetDownloadTables メソッドを使用して、現在の同期の DownloadTableData オブジェクトの配列を取得します。この例は、_cc という DBConnectionContext インスタンスがあることを前提としています。

```
// The method used for the handle_DownloadData event.
public void HandleDownload() {
    // Get the DownloadData for the current synchronization.
    DownloadData my_dd = _cc.GetDownloadData();

    // Get an array of tables to set download operations.
    DownloadTableData[] download_tables = my_dd.GetDownloadTables();

    // Get the first table in the DownloadTableData array.
    DownloadTableData my_download_table = download_tables[0];

    // ...
}
```

GetDownloadTableByName メソッド

構文

DownloadTableData GetDownloadTableByName(
string *table-name*);

備考

現在の同期に使用する名前付きダウンロード・テーブルを取得します。指定された名前のテーブルが現在の同期にない場合は NULL を返します。

戻り値

指定したテーブル名のダウンロード・データ。見つからなかった場合は NULL。

パラメータ

- **table-name**　ダウンロード・データの取得先テーブルの名前。

DownloadTableData インタフェース

構文

interface DownloadTableData

備考

1 つのダウンロード・テーブルの情報を同期用にカプセル化します。このインタフェースを使用して、同期クライアント・サイトにダウンロードされるデータ操作を設定します。

例

たとえば、次のテーブルがあるとします。

```
CREATE TABLE remoteOrders (
   pk INT NOT NULL,
   col1 VARCHAR(200),
   PRIMARY KEY (pk)
);
```

次の例では、DownloadData.GetDownloadTableByName メソッドを使用して、remoteOrders テーブルを表す DownloadTableData インスタンスを返します。

```
// The method used for the handle_DownloadData event
public void HandleDownload() {
   // _cc is a DBConnectionContext instance.

   // Get the DownloadData for the current synchronization.
   DownloadData my_dd = _cc.GetDownloadData();

   // Get the DownloadTableData for the remoteOrders table.
   DownloadTableData td = my_dd.GetDownloadTableByName("remoteOrders");

   // User defined-methods to set download operations.
   SetDownloadUpserts(td);
   SetDownloadDeletes(td);

   // ...
}
```

この例では、SetDownloadInserts メソッドは DownloadTableData.GetUpsertCommand を使用して、挿入または更新するローのコマンドを取得します。IDbCommand は、リモート・データベースに挿入する値の設定先となるパラメータを保持します。

```
void SetDownloadInserts(DownloadTableData td) {
   IDbCommand  upsert_cmd = td.GetUpsertCommand();
   IDataParameterCollection parameters = upsert_cmd.Parameters;

   // The following method calls are the same as the following SQL statement:
   // INSERT INTO remoteOrders(pk, col1) values(2300, "truck");
   ((IDataParameter) (parameters[0])).Value = (Int32) 2300;
   ((IDataParameter) (parameters[1])).Value = (String) "truck";

   if (upsert_cmd.ExecuteNonQuery() > 0) {
      // Insert was not filtered.
   }
   else {
      // Insert was filtered because it was uploaded
      // in the same synchronization.
```



```
    }
  }
```

SetDownloadDeletes メソッドは DownloadTableData.GetDeleteCommand を使用して、削除するローのコマンドを取得します。

```
void SetDownloadDeletes(DownloadTableData td) {
   IDbCommand  delete_cmd = t2_download_dd.GetDeleteCommand();

   // The following method calls are the same as the following SQL statement:
   // DELETE FROM remoteOrders  where pk = 2300;
   IDataParameterCollection parameters = delete_cmd.Parameters;
   ((IDataParameter) (parameters[0])).Value = (Int32) 2300;
   delete_cmd.ExecuteNonQuery();
}
```

GetDeleteCommand メソッド

構文

IDbCommand **GetDeleteCommand();**

備考

ユーザが削除操作をダウンロード・データ操作に追加できるようにするコマンドを取得します。返されるコマンドには、テーブルのプライマリ・キー・カラムと同じ数のパラメータがあります。削除をダウンロードに含めるには、プライマリ・キー・カラムにカラム値を設定し、文を ExecuteNonQuery() で実行する必要があります。

> **注意**
> ダウンロード削除操作対象のすべてのプライマリ・キー値を設定してください。

戻り値

ダウンロードでの削除に使用するコマンド

例

「DownloadTableData インタフェース」 654 ページを参照してください。

GetLastDownloadTime メソッド

構文

DateTime **GetLastDownloadTime();**

備考

このテーブルの最終ダウンロード時刻を返します。これは、テーブルごとのダウンロード・イベントの多くで渡される最終ダウンロード時刻と同じです。

最終ダウンロード時刻は、特定の同期に対してテーブル・ダウンロード・データを生成する場合に便利です。

戻り値

このダウンロード・テーブルの最終ダウンロード時刻

GetName メソッド

構文

string **GetName();**

備考

このインスタンスのテーブル名を取得します。これはユーティリティ関数です。テーブル名には、このインスタンスのスキーマを使用してもアクセスできます。

戻り値

このインスタンスのテーブル名

GetSchemaTable メソッド

構文

DataTable **GetSchemaTable();**

備考

このダウンロード・テーブルのメタデータを記述する DataTable インスタンスを取得します。

DataTable にカラム名情報を含める場合は、カラム名を送信するためのクライアント拡張オプションを指定してください。

戻り値

カラム・メタデータが記述された DataTable

参照

- dbmlsync：「SendColumnNames (scn) 拡張オプション」『Mobile Link - クライアント管理』
- Ultra Light：「Send Column Names 同期パラメータ」『Ultra Light データベース管理とリファレンス』

GetUpsertCommand メソッド

構文

IDbCommand **GetUpsertCommand();**

備考

アップサート (更新または挿入) 操作をダウンロード・データ操作に追加できるようにするコマンドを取得します。返されるコマンドには、テーブルのカラムと同じ数のパラメータがあります。挿入/更新をダウンロードに含めるには、挿入するカラム値を設定し、文を ExecuteNonQuery() を使用して実行する必要があります。コマンドの ExecuteNonQuery() は、挿入/更新操作がフィルタされた場合は 0 を返し、フィルタされていない場合は 1 を返します。

コマンドに対するパラメータの追加や削除はできません。できるのは値の設定だけです。

戻り値

ダウンロードの挿入/更新に使用するコマンド

例

「DownloadTableData インタフェース」 654 ページを参照してください。

LogCallback デリゲート

構文

delegate void **LogCallback(**
  ServerContext *sc*
  LogMessage *message*
)
Member of **iAnywhere.MobiLink.Script**

備考

Mobile Link サーバがメッセージを出力したときに呼び出されます。

LogMessage クラス

構文

class **LogMessage** : **iAnywhere.MobiLink.Script.LogMessage**
Member of **iAnywhere.MobiLink.Script**

備考

ログに出力されたメッセージに関する情報が含まれています。

MessageType 列挙体

構文

enum **MessageType**
Member of **iAnywhere.MobiLink.Script.LogMessage**

備考

LogMessage の使用可能な型の列挙体。

ERROR フィールド

構文

ERROR

備考

ログ・メッセージはエラーです。

INFO フィールド

構文

INFO

備考

ログ情報メッセージ。

WARNING フィールド

構文

WARNING

備考

ログ・メッセージは警告です。

Type プロパティ

構文

LogMessage.MessageType **Type**

備考

このインスタンスが表すログ・メッセージのタイプ。

User プロパティ

構文

string **User**

備考

このメッセージがログに記録されるユーザ。NULL の場合があります。

Text プロパティ

構文

string **Text**

備考

メッセージの本文。

ServerContext インタフェース

構文

interface **ServerContext**
Member of **iAnywhere.MobiLink.Script**

備考

Mobile Link サーバの継続期間中に存在する、すべてのコンテキストのインスタンス化。このコンテキストを静的なデータとして保持し、バックグラウンド・スレッドで使用できます。Mobile Link で起動される .NET CLR の継続期間中は有効です。

ServerContext インスタンスにアクセスするには、DBConnectionContext.getServerContext メソッドを使用します。

GetStartClassInstances メソッド

構文

object[ ] GetStartClassInstances( )
Member of **iAnywhere.MobiLink.Script.ServerContext**

備考

サーバ起動時に構築された起動クラスの配列を取得します。起動クラスがない場合、配列の長さは 0 です。

ユーザ定義起動クラスの詳細については、「ユーザ定義起動クラス」 625 ページを参照してください。

次に、getStartClassInstances() の例を示します。

```
void FindStartClass(ServerContext sc, string name) {
    object[] startClasses = sc.GetStartClassInstances();
    foreach (object obj in startClasses) {
        if (obj is MyClass) {
            // Execute some code.
        }
    }
}
```

LogCallback ErrorListener イベント

このイベントは、Mobile Link サーバがエラーを出力したときにトリガされます。

LogCallback InfoListener イベント

このイベントは、Mobile Link サーバが情報を出力したときにトリガされます。

LogCallback WarningListener イベント

このイベントは、Mobile Link サーバが警告を出力したときにトリガされます。

makeConnection メソッド

構文

iAnywhere.MobiLink.Script.DBConnection makeConnection( )
Member of **iAnywhere.MobiLink.Script.ServerContext**

備考

新しいサーバ接続を作成します。

ShutDown メソッド

構文

void **Shutdown( )**
Member of **iAnywhere.MobiLink.Script.ServerContext**

備考

サーバを強制的に停止します。

ShutdownListener メソッド

構文

event iAnywhere.MobiLink.Script.ShutdownCallback
ShutdownListener(
iAnwyhere.MobiLink.Script.ServerContext *sc***)**
Member of **iAnywhere.MobiLink.Script.ServerContext**

備考

このイベントはシャットダウン時にトリガされます。次のサンプル・コードは、このイベントの使用方法のサンプルです。

```
ShutdownCallback callback = new ShutdownCallback(shutdownHandler);
_sc.ShutdownListener += callback;

public void shutdownHandler(ServerContext sc) {
    _test_out_file.WriteLine("shutdownPerformed");
}
```

getProperties メソッド

構文

NameValueCollection **getProperties(**
string *component_name*
string *prop_set_name* **)**

備考

スクリプト・バージョンに関連する一連のプロパティを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

getPropertiesByVersion メソッド

構文

NameValueCollection **getPropertiesByVersion(** string *script_version* **)**

備考

スクリプト・バージョンに関連する一連のプロパティを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。component_name が ScriptVersion の場合、スクリプト・バージョンは property_set_name カラムに格納されます。

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

getPropertySetNames メソッド

構文

StringCollection **getPropertySetNames(** string *component_name* **)**

備考

指定したコンポーネントのプロパティ・セット名のリストを返します。これらのプロパティは、Mobile Link システム・テーブル ml_property に格納されます。

参照

- 「ml_property」 753 ページ
- 「ml_add_property システム・プロシージャ」 713 ページ

ServerException クラス

構文

public class **ServerException**
Member of **iAnywhere.MobiLink.Script**

備考

サーバでエラーが発生したため直ちにシャットダウンする必要があることを Mobile Link に通知します。

ServerException コンストラクタ

構文

public **ServerException( )**
Member of **iAnywhere.MobiLink.Script.ServerException**

備考

詳細メッセージのない ServerException を構成します。

構文

public **ServerException(string** *message* **)**
Member of **iAnywhere.MobiLink.Script.ServerException**

備考

指定されたメッセージを持つ ServerException を新規に作成します。

パラメータ

- **message**　この ServerException のメッセージ

構文

public **ServerException(string** *message*, **SystemException** *ie* **)**
Member of **iAnywhere.MobiLink.Script.ServerException**

備考

指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む ServerException を新規に作成します。

パラメータ

- **message**　この ServerException のメッセージ
- **ie**　この ServerException の原因となった例外

ShutdownCallback デリゲート

構文

sealed delegate **ShutdownCallback : System.MulticastDelegate**
Member of **iAnywhere.MobiLink.Script**

備考

Mobile Link サーバがシャットダウンするときに呼び出されます。Mobile Link サーバのシャットダウン時に呼び出される ServerContext.ShutdownListener イベントで、このデリゲートの実装を登録できます。

SQLType 列挙体

構文

enum **SQLType**
Member of **iAnywhere.MobiLink.Script**

備考

使用可能なすべての ODBC データ型の列挙体。

SQL_TYPE_NULL フィールド

構文

SQL_TYPE_NULL

備考

NULL データ型。

SQL_UNKNOWN_TYPE フィールド

構文

SQL_UNKNOWN_TYPE

備考

不定のデータ型。

SQL_CHAR フィールド

構文

SQL_CHAR

備考

シングルバイト文字列。.NET 型 String を持ちます。

SQL_NUMERIC フィールド

構文

SQL_NUMERIC

備考

設定されたサイズと精度の数値。.NET 型 decimal を持ちます。

SQL_DECIMAL フィールド

構文

SQL_DECIMAL

備考

設定されたサイズと精度の 10 進数字。.NET 型 decimal を持ちます。

SQL_INTEGER フィールド

構文

SQL_INTEGER

備考

32 ビット整数値。.NET 型 Int32 を持ちます。

SQL_SMALLINT フィールド

構文

SQL_SMALLINT

備考

16 ビット整数値。.NET 型 Int16 を持ちます。

SQL_FLOAT フィールド

構文

SQL_FLOAT

備考

ODBC ドライバで精度が定義された浮動小数点数。.NET 型 Double を持ちます。

SQL_REAL フィールド

構文

SQL_REAL

備考

単精度の浮動小数点数。.NET 型 Single を持ちます。

SQL_DOUBLE フィールド

構文

SQL_DOUBLE

備考

倍精度浮動小数点数。.NET 型 Double を持ちます。

SQL_DATE フィールド

構文

SQL_DATE

備考

日付。.NET 型 DateTime を持ちます。

SQL_DATETIME フィールド

構文

SQL_DATETIME

備考

日付と時刻。.NET 型 DateTime を持ちます。

SQL_TIME フィールド

構文

SQL_TIME

備考

時刻。.NET 型 DateTime を持ちます。

SQL_INTERVAL フィールド

構文

SQL_INTERVAL

備考

時間の間隔。.NET 型 TimeSpan を持ちます。

SQL_TIMESTAMP フィールド

構文

SQL_TIMESTAMP

備考

タイムスタンプ。.NET 型 DateTime を持ちます。

SQL_VARCHAR フィールド

構文

SQL_VARCHAR

備考

シングルバイト文字列。.NET 型 String を持ちます。

SQL_TYPE_DATE フィールド

構文

SQL_TYPE_DATE

備考

日付。.NET 型 DateTime を持ちます。

SQL_TYPE_TIME フィールド

構文

SQL_TYPE_TIME

備考

時刻。.NET 型 DateTime を持ちます。

SQL_TYPE_TIMESTAMP フィールド

構文

SQL_TYPE_TIMESTAMP

備考

タイムスタンプ。.NET 型 DateTime を持ちます。

SQL_DEFAULT フィールド

構文

SQL_DEFAULT

備考

デフォルト型。型を持ちません。

SQL_ARD_TYPE フィールド

構文

SQL_ARD_TYPE

備考

ARD オブジェクト。型を持ちません。

SQL_BIT フィールド

構文

SQL_BIT

備考

シングル・ビット。.NET 型 Boolean を持ちます。

SQL_TINYINT フィールド

構文

SQL_TINYINT

備考

8 ビット整数。.NET SByte 型を持ちます。

SQL_BIGINT フィールド

構文

SQL_BIGINT

備考

64 ビット整数。.NET 型 Int64 を持ちます。

SQL_LONGVARBINARY フィールド

構文

SQL_LONGVARBINARY

備考

ドライバ依存の最大長を持つ可変長バイナリ・データ。.NET 型 byte[ ] を持ちます。

SQL_VARBINARY フィールド

構文

SQL_VARBINARY

備考

ユーザ指定した最大長を持つ可変長バイナリ・データ。.NET 型 byte[ ] を持ちます。

SQL_BINARY フィールド

構文

SQL_BINARY

備考

固定長のバイナリ・データ。.NET 型 byte[ ] を持ちます。

SQL_LONGVARCHAR フィールド

構文

SQL_LONGVARCHAR

備考

シングルバイト文字列。.NET 型 String を持ちます。

SQL_GUID フィールド

構文

SQL_GUID

備考

グローバル・ユニーク ID (UUID とも呼びます)。.NET 型 Guid を持ちます。

SQL_WCHAR フィールド

構文

SQL_WCHAR

備考

固定サイズの Unicode 文字配列。.NET 型 String を持ちます。

SQL_WVARCHAR フィールド

構文

SQL_WVARCHAR

備考

ユーザ定義の最大長を持ち、NULL で終了する Unicode 文字列。.NET 型 String を持ちます。

SQL_WLONGVARCHAR フィールド

構文

SQL_WLONGVARCHAR

備考

ドライバ依存の最大長を持ち、NULL で終了する Unicode 文字列。.NET 型 String を持ちます。

SynchronizationException クラス

構文

class **SynchronizationException**
Member of **iAnywhere.MobiLink.Script**

備考

同期例外が発生したことと、現在の同期をロールバックして再開する必要があることを通知します。

SynchronizationException コンストラクタ

構文

SynchronizationException( )
Member of **iAnywhere.MobiLink.Script.SynchronizationException**

備考

詳細のない SynchronizationException を構成します。

構文

public **SynchronizationException(string** *message* **)**
Member of **iAnywhere.MobiLink.Script.SynchronizationException**

備考

指定されたメッセージを持つ SynchronizationException を新規に作成します。

パラメータ

- **message**　この ServerException のメッセージ

構文

SynchronizationException(string *message*, **SystemException** *ie* **)**
Member of **iAnywhere.MobiLink.Script.SynchronizationException**

備考

指定されたメッセージを持ち、このエラーの原因となった特定の内部例外を含む SynchronizationException を新規に作成します。

パラメータ

- **message**　この ServerException のメッセージ

- **ie** この ServerException の原因となった例外

UploadData インタフェース

構文

public interface **UploadData**

備考

ダイレクト・ロー・ハンドリングで使用するアップロード操作をカプセル化します。アップロード・トランザクションには、ロー操作が格納されたテーブルのセットが含まれています。単一のアップロード・トランザクションを表す UploadData インスタンスが、handle_UploadData 同期イベントに渡されます。

> **警告**
> ダイレクト・ロー・ハンドリングのアップロード操作は、handle_UploadData イベントに対して登録したメソッドで処理してください。登録されたメソッドを呼び出した後、UploadData は破棄されます。後続のイベントで使用するために新しい UploadData インスタンスを作成しないでください。

UploadedTableData インスタンスを取得するには、UploadData.GetUploadedTables メソッドまたは UploadData.GetUploadedTableByName メソッドを使用します。

リモート・データベースがトランザクション・アップロードを使用している場合を除き、同期の UploadData は 1 つです。

参照

- 「ダイレクト・ロー・ハンドリング」 679 ページ
- 「UploadedTableData インタフェース」 674 ページ
- 「handle_UploadData 接続イベント」 477 ページ
- 「ダイレクト・アップロードの処理」 685 ページ

例

「handle_UploadData 接続イベント」 477 ページを参照してください。

GetUploadedTableByName メソッド

構文

UploadedTableData **GetUploadedTableByName(**
string *table-name*);

備考

アップロード・トランザクションの名前付きアップロード・テーブル・データを取得します。指定された名前のテーブルが現在のトランザクションにない場合は NULL を返します。

パラメータ

- **table-name**　アップロード・データの取得先テーブルの名前

戻り値

指定したテーブル名のアップロード・データ。見つからなかった場合は NULL。

例

handle_UploadData 同期イベントに対して HandleUpload というメソッドを使用するものとします。次の例では GetUploadedTableByName メソッドを使用して、remoteOrders テーブルの UploadedTableData インスタンスを返します。

```
// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {
    UploadedTableData uploaded_t1 = ut.GetUploadedTableByName("remoteOrders");
    // ...
}
```

GetUploadedTables メソッド

構文

UploadedTableData[ ] **GetUploadedTables();**

備考

現在のアップロード・トランザクションのすべてのアップロード・テーブル・データを含む配列を取得します。テーブルの配列内での順序は、Mobile Link による SQL のロー・ハンドリングでの順序と同じで、参照整合性違反を防ぐ最適な順序になります。データ・ソースがリレーショナル・データベースの場合は、このテーブル順序を使用してください。

戻り値

アップロード・テーブル・データの配列。配列内でのテーブルの順序は、クライアントのアップロード順と同じです。

例

handle_UploadData 同期イベントに対して HandleUpload というメソッドを使用するものとします。次の例では GetUploadedTables メソッドを使用して、現在のアップロード・トランザクションの UploadedTableData インスタンスを返します。

```
// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ud) {
    UploadedTableData[] tables = ud.GetUploadedTables();
    //...
}
```

UpdateDataReader インタフェース

構文

interface **UpdateDataReader**

備考

1 つのテーブルの 1 つのアップロード・トランザクションのための更新操作を保持します。DataReader のモードを変更することで、新しいローにも古いローにもアクセスできます。そのほかについては、通常の DataReader と同様に使用できます。

SetNewRowValues メソッド

構文

void **SetNewRowValues();**

備考

新しいカラム値 (更新後のロー) を返すように、この DataReader のモードを設定します。これがデフォルト・モードです。

SetOldRowValues メソッド

構文

void **SetOldRowValues();**

備考

古いカラム値 (更新前のロー) を返すように、この DataReader のモードを設定します。

UploadedTableData インタフェース

構文

public interface **UploadedTableData**

備考

1 つのアップロード・テーブルの情報を同期用にカプセル化します。

挿入、更新、削除の各操作には、標準の ADO.NET IDataReader を使用してアクセスできます。テーブル・メタデータへのアクセスには GetSchemaTable() 呼び出しまたは挿入および削除データ・リーダを使用できます。削除データ・リーダには、テーブルのプライマリ・キー・カラムのみが含まれます。

GetDeletes メソッド

構文

IDataReader **GetDeletes();**

備考

このアップロード・テーブル・データに対する削除を保持した DataReader を取得します。個々の削除は、このインスタンス・テーブルのローを一意に表すプライマリ・キー値で表されます。

注意：カラムのインデックスと順序は、このテーブルのスキーマのプロパティ DataTable.PrimaryKey の配列と一致します。

戻り値

削除するローのプライマリ・キー・カラムを保持した DataReader

例

リモート・クライアントに sparse_pk というテーブルがあるものとします。次の例は DownloadTableData.GetDeletes メソッドを使用して、削除するローのデータ・リーダを取得します。この場合、削除データ・リーダには 2 つのプライマリ・キー・カラムが含まれます。各プライマリ・キー・カラムのインデックスに注目してください。

```
CREATE TABLE sparse_pk (
   pcol1  INT NOT NULL,
   col2 VARCHAR(200),
   pcol3 INT NOT NULL,
   PRIMARY KEY (pcol1, pcol3)
);

using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

   // Get an UploadedTableData for the sparse_pk table.
   UploadedTableData sparse_pk_table = ut.GetUploadedTableByName("sparse_pk");

   // Get deletes uploaded by the MobiLink client.
   IDataReader data_reader = sparse_pk_table.GetDeletes();

   while (data_reader.Read()) {
      StringBuilder row_str = new StringBuilder("( ");
      row_str.Append(data_reader.GetString(0)); // pcol1
      row_str.Append(", ");
      row_str.Append(data_reader.GetString(1)); // pcol3
      row_str.Append(" )");
      writer.WriteLine(row_str);
   }
   data_reader.Close();
}
```

GetInserts メソッド

構文

IDataReader **GetInserts();**

備考

このアップロード・テーブル・データに対する挿入を保持した DataReader を取得します。個々の挿入は、リーダが返す 1 つのローで表されています。

戻り値

このテーブル・データの挿入を保持した DataReader

例

```
CREATE TABLE sparse_pk (
   pcol1  INT NOT NULL,
   col2 VARCHAR(200),
   pcol3 INT NOT NULL,
   PRIMARY KEY (pcol1, pcol3)
);

using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

   // Get an UploadedTableData for the sparse_pk table.
   UploadedTableData sparse_pk_table = ut.GetUploadedTableByName("sparse_pk");

   // Get deletes uploaded by the MobiLink client.
   IDataReader data_reader = sparse_pk_table.GetInserts();

   while (data_reader.Read()) {
      StringBuilder row_str = new StringBuilder("( ");
      row_str.Append(data_reader.GetString(0)); // pcol1
      row_str.Append(", ");
      if (data_reader.IsDBNull(1)) {
         row_str.Append("<NULL>");
      }
      else {
         row_str.Append(data_reader.GetString(1)); // col2
      }
      row_str.Append(", ");
      row_str.Append(data_reader.GetString(2)); // pcol3
      row_str.Append(" )");
      writer.WriteLine(row_str);
   }
   data_reader.Close();
}
```

GetName メソッド

構文

string **GetName();**

備考

このインスタンスのテーブル名を取得します。これはユーティリティ関数です。テーブル名には、このインスタンスのスキーマを使用してもアクセスできます。

戻り値

このインスタンスのテーブル名

GetSchemaTable メソッド

構文

DataTable **GetSchemaTable();**

備考

このダウンロード・テーブルのメタデータを記述する DataTable を取得します。

DataTable にカラム名情報を含める場合は、カラム名を送信するためのクライアント拡張オプションを指定してください。

戻り値

カラム・メタデータが記述された DataTable

参照

- dbmlsync：「SendColumnNames (scn) 拡張オプション」『Mobile Link - クライアント管理』
- Ultra Light：「Send Column Names 同期パラメータ」『Ultra Light データベース管理とリファレンス』

GetUpdates メソッド

構文

UpdateDataReader **GetUpdates();**

備考

このアップロード・テーブル・データに対する更新を保持した DataReader を取得します。結果セット内の各ローは 1 つの更新を表します。結果セットのモードは、新しいカラム値と古いカラム値とに切り替えることができます。

戻り値

このテーブル・データに対する更新を保持した DataReader

例

次の例に、GetUpdates メソッドの使用方法を示します。

```
CREATE TABLE sparse_pk (
   pcol1  INT NOT NULL,
   col2 VARCHAR(200),
   pcol3 INT NOT NULL,
   PRIMARY KEY (pcol1, pcol3)
);

using iAnywhere.MobiLink.Script;
using System;
using System.IO;
using System.Data;
using System.Text;
...

// The method used for the handle_UploadData event.
public void HandleUpload(UploadData ut) {

   // Get an UploadedTableData for the sparse_pk table.
   UploadedTableData sparse_pk_table = ut.GetUploadedTableByName("sparse_pk");

   // Get deletes uploaded by the MobiLink client.
   UpdateDataReader data_reader = sparse_pk_table.GetInserts();

   while (data_reader.Read()) {
      data_reader.SetNewRowValues();
      StringBuilder row_str = new StringBuilder("New values ( ");
      row_str.Append(data_reader.GetString(0)); // pcol1
      row_str.Append(", ");
      if (data_reader.IsDBNull(1)) {
         row_str.Append("<NULL>");
      }
      else {
         row_str.Append(data_reader.GetString(1)); // col2
      }
      row_str.Append(", ");
      row_str.Append(data_reader.GetString(2)); // pcol3
      row_str.Append(" )");
      data_reader.SetOldRowValues();
      row_str.Append(" Old Values (");
      row_str.Append(data_reader.GetString(0)); // pcol1
      row_str.Append(", ");
      if (data_reader.IsDBNull(1)) {
         row_str.Append("<NULL>");
      }
      else {
         row_str.Append(data_reader.GetString(1)); // col2
      }
      row_str.Append(", ");
      row_str.Append(data_reader.GetString(2)); // pcol3
      row_str.Append(" )");
      writer.WriteLine(row_str);
   }
   data_reader.Close();
}
```


ダイレクト・ロー・ハンドリング

目次

ダイレクト・ロー・ハンドリングの概要

注意
ダイレクト・ロー・ハンドリングは、Mobile Link の高度な機能です。この機能を使用するには、Mobile Link アプリケーションの作成方法と Mobile Link API の使用方法を熟知していることが前提です。次の項を参照してください。

- Mobile Link - クイック・スタート
- Mobile Link - サーバ管理 1 ページ
- Mobile Link - クライアント管理

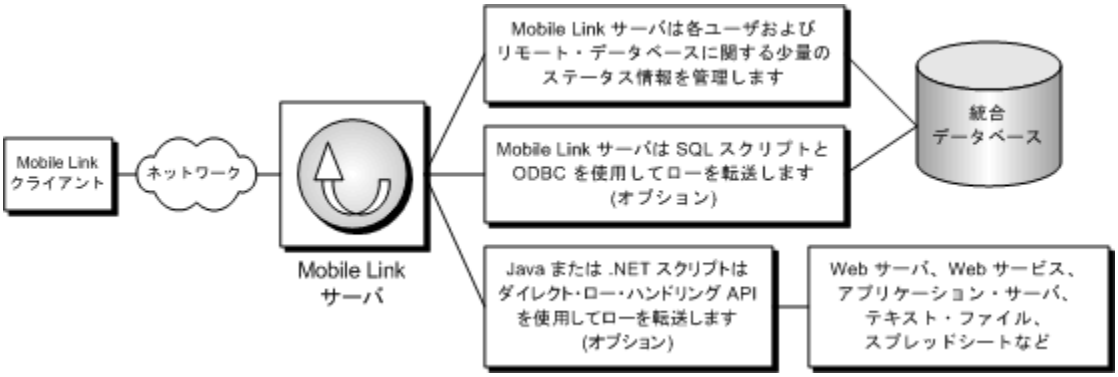
Mobile Link は、SQL とダイレクトの 2 種類のロー・ハンドリングをサポートしています。この 2 つは個別に使用することも併用することもできます。

- **SQL ロー・ハンドリング**　リモート・データをサポートされた統合データベースに同期できます。SQL ベースのイベントは、競合の解決などの同期タスクを実行するための堅牢なインタフェースを提供します。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 を返したりできます。
- **ダイレクト・ロー・ハンドリング**　リモート・データを任意の中央テータ・ソースに同期できます。ダイレクト・ロー・ハンドリングを使用することで、特別な Mobile Link イベントや Java 用および .NET 用の Mobile Link サーバ API を使用して、同期された未加工のデータにアクセスできます。

  同期可能なデータ・ソースには、アプリケーション、Web サーバ、Web サービス、アプリケーション・サーバ、テキスト・ファイル、スプレッドシート、非リレーショナル・データベースなど、ほとんどのものが該当します。また、統合データベースとして使用できない RDBMS も指定できます。ただし、Mobile Link システム・テーブルの格納には統合データベースが必要です。また、ダイレクト・ロー・ハンドリングの数多くの実装が、統合データベースと別のデータ・ソースとの両方に同期します。

  ダイレクト・ロー・ハンドリングを使用するには、Mobile Link 統合データベースの作成方法、同期スクリプトの追加方法、Mobile Link リモート・ユーザの作成方法に関する知識が必要です。

次の図は、Mobile Link の基本アーキテクチャを示しています。



ダイレクト・ロー・ハンドリングのコンポーネント

ダイレクト・ロー・ハンドリングを実装するには、2 つの同期イベントと、Java と .NET 用の Mobile Link サーバ API のいくつかのインタフェースとメソッドを使用します。

ダイレクト同期イベント

ダイレクト・ロー・ハンドリングを使用すると、アップロード・ストリームやダウンロード・ストリームに直接アクセスできます。これを行うには、handle_UploadData と handle_DownloadData 同期イベントを処理する Java または .NET メソッドを作成します。

- **handle_UploadData** UploadData パラメータを 1 つ受け取ります。このパラメータは、1 つのアップロード・トランザクションに対して Mobile Link クライアントによってアップロードされる操作をカプセル化します。次の項を参照してください。

  - 「ダイレクト・アップロードの処理」 685 ページ
  - 「handle_UploadData 接続イベント」 477 ページ

- **handle_DownloadData** DownloadData インタフェースを使用したダウンロード操作を設定します。次の項を参照してください。

  - 「ダイレクト・ダウンロードの処理」 692 ページ
  - 「handle_DownloadData 接続イベント」 465 ページ

ダイレクト・ロー・ハンドリングに使用する Mobile Link サーバ API のコンポーネント

Java API の場合

- DBConnectionContext 「getDownloadData メソッド」 571 ページ
- 「DownloadData インタフェース」 575 ページ
- 「DownloadTableData インタフェース」 577 ページ
- 「UpdateResultSet」 605 ページ
- 「UploadData インタフェース」 607 ページ
- 「UploadedTableData インタフェース」 609 ページ

.NET API の場合

- DBConnectionContext 「GetDownloadData メソッド」 640 ページ
- 「DownloadData インタフェース」 652 ページ
- 「DownloadTableData インタフェース」 654 ページ
- 「UpdateDataReader インタフェース」 674 ページ
- 「UploadedTableData インタフェース」 674 ページ
- 「UploadData インタフェース」 672 ページ

クイック・スタート

ダイレクト・ロー・ハンドリングを使用するには、Mobile Link 統合データベースの作成方法、同期スクリプトの追加方法、Mobile Link リモート・ユーザの作成方法に関する知識が必要です。

統合データベース以外のデータ・ソースと同期するには、以下の手順に従います。

♦ ダイレクト・ロー・ハンドリングの設定の概要

1. 統合データベースがない場合は、設定します。

   統合データベースと同期するかどうかに関係なく、Mobile Link システム・テーブルを保持するためには統合データベースが必要です。

   「Mobile Link 統合データベース」 3 ページを参照してください。

2. アップロードを処理するには、UploadData インタフェースを使用してパブリック・メソッドを作成し、handle_UploadData 接続イベントに登録します。

   「ダイレクト・アップロードの処理」 685 ページを参照してください。

3. ダウンロードを処理するには、DownloadData インタフェースを使用してパブリック・メソッドを作成し、handle_DownloadData 接続イベント (または他のイベント) に登録します。

   「ダイレクト・ダウンロードの処理」 692 ページを参照してください。

4. ロー・ハンドリング API を使用してカラムを (インデックスではなく) 名前で参照する場合は、クライアントで、アップロードとともにカラム名が送信されるように指定します。次の項を参照してください。

   - SQL Anywhere クライアント：「SendColumnNames (scn) 拡張オプション」 『Mobile Link - クライアント管理』
   - Ultra Light：「Send Column Names 同期パラメータ」 『Ultra Light データベース管理とリファレンス』

クイック・スタートのためのその他の資料

- 「チュートリアル：ダイレクト・ロー・ハンドリングの概要」 『Mobile Link - クイック・スタート』
- http://www.sybase.com/detail?id=1058600#319
- 「Java 同期論理の設定」 555 ページ
- 「.NET 同期論理の設定」 619 ページ

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。

ダイレクト・ロー・ハンドリングに関する開発のヒント

ユニークなプライマリ・キー

Mobile Link 同期 (ダイレクト・ロー・ハンドリングを含む) では、データ・ソースは、更新されないユニークなプライマリ・キーを必要とします。スプレッドシートやテキスト・ファイルなどの非リレーショナル・データ・ソースの場合、これは、1 つのカラムに、ローを識別するユニークで不変の値が含まれている必要があることを意味します。

「ユニークなプライマリ・キーの管理」 143 ページを参照してください。

カラム名

ダイレクト・ロー・ハンドリングを使用するとき、テーブルのカラム名は、Mobile Link クライアントがカラム名を送信するように設定されている場合にのみ使用できます。または、カラム・インデックスを使用してローの情報にアクセスすることもできます。

カラム名を使用するには、次の項を参照してください。

- SQL Anywhere リモート：「SendColumnNames (scn) 拡張オプション」 『Mobile Link - クライアント管理』
- Ultra Light リモート：「Send Column Names 同期パラメータ」 『Ultra Light データベース管理とリファレンス』

ダウンロードに最終ダウンロード時刻を使用する

可能なかぎり、タイムスタンプベースの SQL アプリケーションのようにダイレクト・ロー・ハンドリング・アプリケーションを設定してください。つまり、last_modified カラムを維持し、そのカラムに基づいてデータをダウンロードします。この方法によって、異なるダウンロード方法を使用した場合に発生する可能性がある予期しない問題を回避できます。

「タイムスタンプベースのダウンロード」 133 ページを参照してください。

アップロードのトランザクション管理

Mobile Link 統合データベースを使用してトランザクションをコミットすることはできません。一方、ダイレクト・ロー・ハンドリング・データ・ソースを使用してトランザクションをコミットすることは可能です。トランザクション管理を設定する場合は、次のヒントを参考にしてください。

- **アップロードは Mobile Link によってコミットされる前にコミットする** Mobile Link は、アップロードを適用するとき、end_upload イベントの最後で変更をコミットします。保持するすべてのアップロード変更は、end_upload スクリプトが終了する前にコミットしてください。これを実行しなかった場合、エラーや失敗が発生したときに、アプリケーションはアップロードが適用されたとみなしていても Mobile Link はデータを適用していない状態になり、その結果、データが喪失する可能性があります。

- **冗長なアップロードを処理する**　アップロードされたローをアプリケーションがコミットしてから Mobile Link サーバがコミットするまでにエラーや失敗が発生すると、Mobile Link サーバとデータソースの一貫性が失われる場合があります。この問題を解決するには、冗長なアップロードを許可し、冗長なアップロードが適切に適用されるようにするための論理を用意します。特に、アプリケーションがアップロードをもう一度送信する場合は、アップロードが再度適用されないようにしてください。

エラーの処理

エラーを処理するには、上記のように、適切なトランザクション管理を使用します。また、ローを処理する Java コードおよび .NET コードでは、発生するすべての例外を Mobile Link サーバに送信してください。Mobile Link サーバまたはアプリケーションが変更をコミットする前にエラーが発生した場合、Mobile Link は、トランザクションをロールバックし、アプリケーションとの一貫性がある状態を維持します。

クラス・インスタンス

ダイレクト・ロー・ハンドリングでは、Mobile Link は、データベース接続ごとにクラス・インスタンスを 1 つ作成します。クラス・インスタンスが破棄されるのは、同期の最後ではなく、データベース接続が閉じられたときです。クラス・レベル変数は、以前の同期から値を保持します。

ダイレクト・アップロードの処理

ダイレクト・アップロードを処理するには、次の手順を実行します。

♦ ダイレクト・アップロードを処理するには

1. 「handle_UploadData 接続イベント」 477 ページ用の Java または .NET メソッドを登録します。
2. handle_UploadData 同期イベント用のメソッドを作成します。このイベントは、UploadData パラメータを 1 つ受け取ります。次の項を参照してください。
   - Java サーバ API：「UploadData インタフェース」 607 ページ
   - .NET サーバ API：「UploadData インタフェース」 672 ページ

通常、handle_UploadData イベントは、同期のたびに 1 度呼び出されます。ただし、トランザクション・レベル・アップロードを使用する SQL Anywhere クライアントでは、同期のたびに複数回のアップロードが発生する可能性がありますが、その場合は handle_UploadData をトランザクションごとに 1 回呼び出すことで対応できます。

dbmlsync トランザクション・レベル・アップロードの詳細については、「-tu オプション」『Mobile Link - クライアント管理』を参照してください。

Java または .NET 同期スクリプトの作成に関する一般的な情報については、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」 553 ページ
- 「.NET での同期スクリプトの作成」 617 ページ

接続レベル・イベントの登録の詳細については、次の項を参照してください。

- 「ml_add_java_connection_script システム・プロシージャ」 705 ページ
- 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ

ダイレクト・アップロードに使用するクラス

Java と .NET 用の Mobile Link サーバ API には、ダイレクト・アップロードの処理用に次のインタフェースが用意されています。

- **UploadData** 1 つのアップロード・トランザクションをカプセル化します。アップロード・トランザクションには、ロー操作が格納されたテーブルのセットが含まれています。次の項を参照してください。
  - Java API：「UploadData インタフェース」 607 ページ
  - .NET API：「UploadData インタフェース」 672 ページ
- **UploadedTableData** Mobile Link クライアントによってアップロードされた、テーブルの挿入、更新、削除操作をカプセル化します。Java の場合、UploadedTableData メソッドは UpdateResultSet のインスタンスを返します。.NET の場合、UploadedTableData メソッドは UpdateDataReader インタフェースのインスタンスを返します。返された結果セット

IDataReaders を参照して、アップロードされたローに対する操作を処理します。次の項を参照してください。

- Java API：「UploadedTableData インタフェース」 609 ページ
- .NET API：「UploadedTableData インタフェース」 674 ページ

- **UpdateResultSet** Java の場合、このクラスは、UploadedTableData の getUpdates メソッドによって返された更新結果セットを表します。このクラスでは、更新ローの新しいバージョンと古いバージョンを取得するための特別なメソッドが含まれるように java.sql.ResultSet が拡張されています。

  「UpdateResultSet」 605 ページを参照してください。

  .NET の場合、UpdateDataReader インタフェースは UploadedTableData GetUpdates メソッドによって返されたロー・セットを表します。このインタフェースでは、更新ローの新しいバージョンと古いバージョンを取得するための特別なメソッドが含まれるように IDataReader が拡張されています。

  「UpdateDataReader インタフェース」 674 ページを参照してください。

例

「handle_UploadData 接続イベント」 477 ページを参照してください。

ダイレクト・アップロードでの競合の処理

Mobile Link クライアントが更新済みのローを Mobile Link サーバに送信するときは、更新された値 (更新後または新しいイメージ・ロー) だけでなく、Mobile Link サーバとの最後の同期で得た古いローの値 (更新前または古いイメージ・ロー) のコピーも含まれています。更新前イメージ・ローが中央データ・ソースの現在の値と一致しないと、競合が検出されます。

SQL ベースの競合解決

SQL ベースのアップロードの場合、Mobile Link 統合データベースが中央データ・ソースであり、Mobile Link は競合の検出と解決用に特別なイベントを提供しています。

「競合の解決」 150 ページを参照してください。

ダイレクト・ロー・ハンドリングを使用した競合解決

ダイレクト・アップロードの場合、新しいローと古いローにプログラムを使用してアクセスして、競合の検出と解決に使用できます。

UpdateResultSet (UploadedTableData.getUpdates メソッドが返す) は、競合処理に使用する特別なメソッドが含まれるように、Java または .NET 標準の結果セットを拡張したものです。setNewRowValues は、リモート・クライアントからの新しい更新された値を返すように UpdateResultSet を設定します (デフォルト・モード)。setOldRowValues は、古いロー値を返すように UpdateResultSet を設定します。

ダイレクト・ロー・ハンドリングを使用した競合検出

UpdateResultSet のメソッド .setOldRowValues を使用することによって、リモートのローの変更される前の値を取得します。返された値をデータ・ソースの既存のロー値と比較します。比較したローが同じではない場合、競合が存在しています。

ダイレクト・ロー・ハンドリングを使用した競合解決

アップロード中に競合を検出したら、カスタム・ビジネス論理を使用して競合を解決できます。競合は、Java コードまたは .NET コードによって処理されます。

例

XML ドキュメント内の在庫を追跡していて、XML ドキュメントを中央データ・ソースとして使用すると仮定します。User1 は、リモート・データベースの 1 つである Remote1 を使用します。User2 は、別のデータベースである Remote2 を使用します。

XML ドキュメント、User1、User2 の最初の在庫数はすべて 10 個です。User1 が、3 個を販売し、Remote1 にある在庫数を 7 個に更新します。User2 は 4 個販売し、Remote2 にある在庫数を 6 個に更新します。Remote1 が同期を実行すると、中央データベースは 7 個に更新されます。Remote2 が同期を実行すると、在庫の値が 10 個ではなくなっているため、競合が検出されます。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。

- 中央データ・ソースにある現在の値。
- Remote2 がアップロードした新しいローの値。
- Remote2 が最後の同期中に取得した古いローの値。

この場合、ビジネス論理は新しい在庫数を計算し、競合を解決するために次の式を使用します。

```
current data source - (old remote - new remote)
-> 7 - (10-6) = 3
```

Java と .NET の次のプロシージャでは、ダイレクト・アップロードでのこのような競合を解決する方法を、次のテーブルを例に示しています。

```
CREATE TABLE remoteOrders
(
   pk integer primary key not null,
   inventory integer not null
);
```

♦ 直接競合を処理するには、次の手順に従います (Java)。

1. handle_UploadData 接続イベント用の Java または .NET メソッドを登録します。

   「handle_UploadData 接続イベント」 477 ページを参照してください。

   たとえば、次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handle_UploadData 接続イベントに対して HandleUpload という Java メソッドを登録します。Mobile Link 統合データベースに対してこのストアド・プロシージャを実行します。

   ```
   call ml_add_java_connection_script( 'ver1',
     'handle_UploadData',
     'OrderProcessor.HandleUpload' )
   ```

同期イベント用のメソッド登録の詳細については、次の項を参照してください。

- 「スクリプトの追加と削除」 345 ページ
- 「ml_add_java_connection_script システム・プロシージャ」 705 ページ

2. アップロード内のテーブルの UpdateResultSet を取得します。

   OrderProcessor.HandleUpload メソッドは、remoteOrders テーブルの UpdateResultSet を取得します。

```
// method for handle_UploadData event
 public void HandleUpload( UploadData u_data )
 {

   // Get UploadedTableData for the remoteOrders table.
     UploadedTableData u_table = u_data.getUploadedTableByName("remoteOrders");

   // Get an UpdateResultSet for the remoteOrders table.
   UpdateResultSet update_rs = u_table.getUpdates();

   // (Continued...)
```

3. 更新ごとに、中央データ・ソースの現在の値を取得します。

   次の例では、UpdateResultSet の getInt メソッドが、プライマリ・キー・カラム (先頭カラム) の整数値を返します。getMyCentralData メソッドを実装し、使用することにより、中央データ・ソースからデータを取得できます。

```
while( update_rs.next() )
{
  // Get central data source values.

  // Get the primary key value.
  int pk_value = update_rs.getInt(1);

  // Get central data source values.
  int central_value = getMyCentralData(pk_value);

  // (Continued...)
```

4. 更新ごとに、Mobile Link クライアントによってアップロードされた古い値と新しい値を取得します。

   次の例では、UpdateResultSet の setOldRowValues と setNewRowValues を使用して、それぞれ古い値と新しい値を取得しています。

```
  // Set mode for old row values.
  update_rs.setOldRowValues();

  // Get the _old_ stored value on the remote.
  int old_value = update_rs.getInt(2);

  // Set mode for new row values.
  update_rs.setNewRowValues();

  // Get the _new_ updated value on the remote.
  int new_value = update_rs.getInt(2);

  // (Continued...)
```

5. 更新ごとに、競合がないかどうかを確認します。

古いロー値が中央データ・ソースの現在の値と一致しない場合、競合になります。競合を解決するため、ビジネス論理を使用して解決後の値を計算します。競合がなかった場合、中央データ・ソースは新しいリモート値で更新されます。setMyCentralData メソッドを実装し、使用することにより、更新を実行します。

```
// Check if there is a conflict.

if(old_value == central_value)
{
  // No conflict.
  setMyCentralData(pk_value, new_value);

}
else
{
  // Handle the conflict.
  int inventory = old_value - new_value;
  int resolved_value = central_value - inventory;

  setMyCentralData(pk_value, resolved_value);

}
}
```

♦ 直接競合を処理するには、次の手順に従います (.NET)。

1. handle_UploadData 接続イベント用のメソッドを登録します。

   たとえば、次のストアド・プロシージャ・コールは、スクリプト・バージョン ver1 を同期するときに、handle_UploadData 接続イベントに対して HandleUpload という .NET メソッドを登録します。Mobile Link 統合データベースに対してこのストアド・プロシージャを実行します。

   ```
   call ml_add_dnet_connection_script( 'ver1',
     'handle_UploadData',
     'MyScripts.OrderProcessor.HandleUpload' )
   ```

   同期イベント用のメソッド登録の詳細については、次の項を参照してください。

   - 「スクリプトの追加と削除」 345 ページ
   - 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ

2. アップロード内のテーブルの UpdateDataReader を取得します。

   MyScripts.OrderProcessor.HandleUpload メソッドは、remoteOrders テーブルの UpdateResultSet を取得します。

   ```
   // method for handle_UploadData event
    public void HandleUpload( UploadData u_data )
    {

      // Get UploadedTableData for the remoteOrders table.
        UploadedTableData u_table = u_data.GetUploadedTableByName("remoteOrders");

      // Get an UpdateDataReader for the remoteOrders table.
      UpdateDataReader update_dr = u_table.GetUpdates();

      // (Continued...)
   ```

3. 更新ごとに、中央データ・ソースの現在の値を取得します。

   次の例では、UpdateDataReader の GetInt32 メソッドが、プライマリ・キー・カラム (先頭カラム) の整数値を返します。getMyCentralData メソッドを実装し、使用することにより、中央データ・ソースからデータを取得できます。

```
while( update_dr.Read() )
{
  // Get central data source values.

  // Get the primary key value.
  int pk_value = update_dr.GetInt32(0);

  // Get central data source values.
  int central_value = getMyCentralData(pk_value);

  // (Continued...)
```

4. 更新ごとに、Mobile Link クライアントによってアップロードされた古い値と新しい値を取得します。

   次の例では、UpdateResultSet の setOldRowValues と setNewRowValues を使用して、それぞれ古い値と新しい値を取得しています。

```
// Set mode for old row values.
update_dr.SetOldRowValues();

// Get an _old_ value.
int old_value = update_dr.GetInt32(1);

// Set mode for new row values.
update_dr.SetNewRowValues();

// Get the _new_ updated value.
int new_value = update_dr.GetInt32(1);

// (Continued...)
```

5. 更新ごとに、競合がないかどうかを確認します。

   古いロー値が中央データ・ソースの現在の値と一致しない場合、競合になります。競合を解決するため、ビジネス論理を使用して解決後の値を計算します。競合がなかった場合、中央データ・ソースは新しいリモート値で更新されます。setMyCentralData メソッドを実装し、使用することにより、更新を実行します。

```
// Check if there is a conflict.

if(old_value == central_value)
{
  // No conflict.
  setMyCentralData(pk_value, new_value);

}
else
{
  // Handle the conflict.
  int inventory = old_value - new_value;
  int resolved_value = central_value - inventory;

  setMyCentralData(pk_value, resolved_value);
```



```
    }
}
```

ダイレクト・ダウンロードの処理

ダイレクト・ダウンロードを処理するには、次の手順を実行します。

♦ ダイレクト・ダウンロードを処理するには

1. 「handle_DownloadData 接続イベント」 465 ページ用の Java または .NET メソッドを登録します。
2. handle_DownloadData 同期イベント用のメソッドを作成します。このイベントでは、DBConnectionContext のインスタンスを使用して、現在の同期に対する DowloadData インスタンスを取得します。次の項を参照してください。
   - Java：「DBConnectionContext インタフェース」 570 ページ
   - Java：「DownloadData インタフェース」 575 ページ
   - .NET：「DBConnectionContext インタフェース」 639 ページ
   - .NET：「DownloadData インタフェース」 652 ページ

handle_DownloadData 同期イベントでダイレクト・ダウンロード全体を作成できます。また、他の同期イベントを使用してダイレクト・ダウンロード操作を設定することもできます。ただし、handle_DownloadData スクリプトを (そのメソッドが何も処理しない場合でも) 作成する必要があります。ダイレクト・ダウンロードを handle_DownloadData 以外のイベントで処理する場合、そのイベントは begin_synchronization より前または end_download より後に実装できません。

イベントの順序の詳細については、「Mobile Link の完全なイベント・モデル」 365 ページを参照してください。

ダイレクト・ダウンロードに使用するクラス

Java と .NET 用の Mobile Link サーバ API には、ダイレクト・ダウンロードの作成用に次のクラスが用意されています。

- **DownloadData**　同期中にリモート・クライアントへ送信する操作を含むダウンロード・テーブルをカプセル化します。次の項を参照してください。
  - Java：「DownloadData インタフェース」 575 ページ
  - .NET：「DownloadData インタフェース」 652 ページ
- **DownloadTableData**　Mobile Link クライアントにダウンロードするアップサート (更新と挿入) と削除操作をカプセル化します。

  Java の場合、DownloadTableData メソッドは JDBC PreparedStatement のインスタンスを返します。Java の場合、ダウンロードにローを追加するには、準備文のカラム値を設定してから、準備文を実行します。

  .NET の場合、DownloadTableData メソッドは .NET IDbCommand のインスタンスを返します。.NET の場合、ダウンロードにローを追加するには、コマンドのカラム値を設定してから、コマンドを実行します。

次の項を参照してください。

- Java：「DownloadTableData インタフェース」 577 ページ
- .NET：「DownloadTableData インタフェース」 654 ページ

例

「handle_DownloadData 接続イベント」 465 ページを参照してください。

Mobile Link リファレンス

この項では、Mobile Link リファレンス情報を示します。

Mobile Link サーバ・システム・プロシージャ

目次

Mobile Link システム・プロシージャ

Mobile Link には、アプリケーションの作成に役立つ、以下のストアド・プロシージャが用意されています。

スクリプトを追加または削除するためのシステム・プロシージャ

同期スクリプトは、統合データベース内のシステム・テーブルに追加しないと使用できません。次のシステム・プロシージャを使用して、同期スクリプトを統合データベースに追加するか、または統合データベースから削除します。

- 「ml_add_connection_script システム・プロシージャ」 702 ページ
- 「ml_add_table_script システム・プロシージャ」 716 ページ
- 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ
- 「ml_add_dnet_table_script システム・プロシージャ」 704 ページ
- 「ml_add_java_connection_script システム・プロシージャ」 705 ページ
- 「ml_add_java_table_script システム・プロシージャ」 706 ページ

Java または .NET 用 Mobile Link サーバ API を使用する場合は、これらのシステム・プロシージャを使用して、イベント用のスクリプトとしてメソッドを登録して、イベントが発生したときにメソッドが実行されるようにします。これらのシステム・プロシージャを使用して、メソッドを登録解除することもできます。

システム・プロシージャを使用してスクリプトを追加する場合、スクリプトは 1 つの文字列になります。スクリプト内の文字列はすべて、エスケープする必要があります。SQL Anywhere の場合、文字列が切れないように各引用符 (') を 2 つ重ねる必要があります。

Adaptive Server Enterprise 11.5 以前のバージョンに 255 バイトを超えるスクリプトを追加する場合は、システム・プロシージャを使用できません。長いスクリプトを定義する場合、Sybase Central を使用するか、直接挿入します。

DB2 メインフレームのバージョン 8.1 は下位互換性モードをサポートしており、カラム名とその他の識別子が最大 18 文字までに制限されています。この環境をサポートするためには、DB2 メインフレーム内のすべての Mobile Link システム・オブジェクトの名前を 18 文字以下にする必要があります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

バージョン 6 より前の IBM DB2 LUW では、カラム名とその他の識別子は 18 文字までしかサポートされないため、名前がトランケートされます。たとえば、ml_add_connection_script は ml_add_connection_ に短縮されます。

その他のシステム・プロシージャ

- 「ml_add_property システム・プロシージャ」 713 ページ
- 「ml_delete_sync_state_before システム・プロシージャ」 721 ページ
- 「ml_reset_sync_state システム・プロシージャ」 722 ページ

IBM DB2 メインフレームのシステム・プロシージャ名の変換

IBM DB2 メインフレーム統合データベースでは、カラム名と他の識別子に 18 文字までしか使用できません。次の表は、DB2 メインフレーム統合データベースのシステム・プロシージャ名が、他のすべてのタイプの統合データベースのシステム・プロシージャ名にどのようにマッピングされるのかを示します。

システム・プロシージャ名が次の表にない場合、変換は必要ありません。

システム・プロシージャ名	DB2 メインフレーム統合データベースのシステム・プロシージャ名
「ml_add_connection_script システム・プロシージャ」 702 ページ	ml_add_cs
「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ	ml_add_dcs
「ml_add_dnet_table_script システム・プロシージャ」 704 ページ	ml_add_dts
「ml_add_java_connection_script システム・プロシージャ」 705 ページ	ml_add_jcs
「ml_add_java_table_script システム・プロシージャ」 706 ページ	ml_add_jts
「ml_add_lang_connection_script システム・プロシージャ」 708 ページ	ml_add_lcs
「ml_add_lang_connection_script_chk システム・プロシージャ」 708 ページ	ml_add_lcs_chk
「ml_add_lang_table_script システム・プロシージャ」 708 ページ	ml_add_lts
「ml_add_lang_table_script_chk システム・プロシージャ」 708 ページ	ml_add_lts_chk
「ml_add_passthrough システム・プロシージャ」 708 ページ	ml_add_pt
「ml_add_passthrough_repair システム・プロシージャ」 710 ページ	ml_add_pt_repair
「ml_add_passthrough_script システム・プロシージャ」 711 ページ	ml_add_pt_script

システム・プロシージャ名	DB2 メインフレーム統合データベースのシステム・プロシージャ名
「ml_add_table_script システム・プロシージャ」 716 ページ	ml_add_ts
「ml_delete_passthrough システム・プロシージャ」 717 ページ	ml_del_pt
「ml_delete_passthrough_repair システム・プロシージャ」 718 ページ	ml_del_pt_repair
「ml_delete_passthrough_script システム・プロシージャ」 719 ページ	ml_del_pt_script
「ml_delete_sync_state システム・プロシージャ」 720 ページ	ml_del_sstate
「ml_delete_sync_state_before システム・プロシージャ」 721 ページ	ml_del_sstate_b4
「ml_reset_sync_state システム・プロシージャ」 722 ページ	ml_reset_sstate

ml_add_column system procedure

リモート・データベースのカラムに関する情報を登録します。この情報は名前付きカラム・パラメータで使用されます。

構文

ml_add_column (
'*version***',**
'*table***',**
'*column***',**
'*type***'**
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
table	VARCHAR(128)。テーブル名。
column	VARCHAR(128)。カラムの名前。

構文	説明
type	VARCHAR(128)。今後の使用のために予約されています。NULL に設定されます。

備考

このプロシージャは、リモート・データベースのカラムに関する情報を Mobile Link システム・テーブル ml_column に設定します。この情報は名前付きロー・パラメータで使用されます。

警告
ml_add_column の呼び出しは、リモート・データベース・テーブルでのカラム順と同じ順序で実行される必要があります。そうしないと、データが正しくなくなることがあります。

次の両方に該当する場合にこのシステム・プロシージャを実行する必要があります。

- SQL スクリプトに、カラムの名前付きパラメータが含まれる (o.column-name、r.column-name など)。
- **同期モデル作成ウィザード**を使用していない。

同期モデル作成ウィザードを使用している場合でも、モデル・モード以外でリモート・スキーマを変更した場合は、このストアド・プロシージャを使用して、ml_column に登録されていないカラムに関する情報を送信する必要があります。

特定のスクリプト・バージョン内のテーブル名のエントリをすべて削除するには、カラム名を NULL に設定します。

参照

- 「ml_column」 738 ページ
- 「スクリプトのパラメータ」 336 ページ

例

次のストアド・プロシージャ・コールは、スクリプト・バージョン Version1 で、MyTable の col1 に関する情報を Mobile Link システム・テーブル ml_column に設定します。このコールによって、Version1 スクリプト・バージョンで、MyTable1 のテーブル・スクリプトに名前付きロー・パラメータ r.col1 と o.col1 を使用できます。

```
CALL ml_add_column( 'Version1', 'MyTable1', 'col1', NULL )
```

次のストアド・プロシージャ・コールは、スクリプト・バージョン Version1 で、Mobile Link システム・テーブル ml_column 内の MyTable1 のエントリをすべて削除します。

```
CALL ml_add_column( 'Version1', 'MyTable1', NULL, NULL )
```

ml_add_connection_script システム・プロシージャ

このシステム・プロシージャを使用して、SQL 接続スクリプトを統合データベースに追加するか、または統合データベースから削除します。

構文

ml_add_connection_script (
'*version*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。

備考

接続スクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトを追加すると、スクリプトが ml_script テーブルに挿入され、このスクリプトを指定のイベントとスクリプト・バージョンに関連付ける適切な参照が定義されます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_cs という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「スクリプトを追加または削除するためのシステム・プロシージャ」 698 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「ml_add_table_script システム・プロシージャ」 716 ページ
- 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ
- 「ml_add_dnet_table_script システム・プロシージャ」 704 ページ
- 「ml_add_java_connection_script システム・プロシージャ」 705 ページ
- 「ml_add_java_table_script システム・プロシージャ」 706 ページ

例

次の文は、begin_synchronization イベントと関連付けられた接続スクリプトを SQL Anywhere 統合データベースのスクリプト・バージョン custdb に追加します。追加されるスクリプト自体は、@EmployeeID 変数を設定する単一の文です。


```
call ml_add_connection_script( 'custdb',
 'begin_synchronization',
 'set @EmployeeID = {ml s.username}' )
```

ml_add_dnet_connection_script システム・プロシージャ

このシステム・プロシージャを使用して、.NET メソッドを接続イベント用のスクリプトとして登録したり、登録解除したりします。

構文

ml_add_dnet_connection_script (
'*version*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。

備考

メソッドを登録解除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトの内容の値は、たとえば MyClass.MyMethod などの .NET アセンブリに含まれるクラス内のパブリック・メソッドです。

ml_add_dnet_connection_script を呼び出すと、メソッドが指定のイベントとスクリプト・バージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_dcs という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「スクリプトを追加または削除するためのシステム・プロシージャ」 698 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページ
- 「ml_add_dnet_table_script システム・プロシージャ」 704 ページ
- 「ml_add_connection_script システム・プロシージャ」 702 ページ
- 「ml_add_table_script システム・プロシージャ」 716 ページ
- 「ml_add_java_table_script システム・プロシージャ」 706 ページ
- 「メソッド」 624 ページ
- 「.NET での同期スクリプトの作成」 617 ページ

例

次の例は、ExampleClass クラスの beginDownloadConnection メソッドを begin_download イベントに対して登録します。

```
call ml_add_dnet_connection_script( 'ver1',
'begin_download',
'ExamplePackage.ExampleClass.beginDownloadConnection' );
```

ml_add_dnet_table_script システム・プロシージャ

このシステム・プロシージャを使用して、.NET メソッドをテーブル・イベント用のスクリプトとして登録したり、登録解除したりします。

構文

ml_add_dnet_table_script (
'*version***',**
'*table***',**
'*event***',**
'*script***'**
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
table	VARCHAR(128)。テーブル名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。

備考

メソッドを登録解除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトの内容の値は、たとえば MyClass.MyMethod などの .NET アセンブリに含まれるクラス内のパブリック・メソッドです。

ml_add_dnet_table_script を呼び出すと、メソッドが指定のテーブル、イベント、スクリプト・バージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_dts という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「スクリプトを追加または削除するためのシステム・プロシージャ」 698 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ
- 「ml_add_connection_script システム・プロシージャ」 702 ページ
- 「ml_add_table_script システム・プロシージャ」 716 ページ
- 「ml_add_java_connection_script システム・プロシージャ」 705 ページ
- 「メソッド」 624 ページ
- 「.NET での同期スクリプトの作成」 617 ページ

例

次の例では、EgClass クラスの empDownloadCursor メソッドを emp テーブルの download_cursor イベントに割り当てます。

```
call ml_add_dnet_table_script( 'ver1', 'emp',
'download_cursor','EgPackage.EgClass.empDownloadCursor' )
```

ml_add_java_connection_script システム・プロシージャ

このシステム・プロシージャを使用して、Java メソッドを接続イベント用のスクリプトとして登録したり、登録解除したりします。

構文

ml_add_java_connection_script (
'*version***',**
'*event***',**
'*script***'**
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。

構文	説明
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。

備考

メソッドを登録解除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトの内容の値は、たとえば MyClass.MyMethod などの Mobile Link サーバのクラスパスに含まれるクラス内のパブリック・メソッドです。

ml_add_java_connection_script を呼び出すと、メソッドが指定のイベントとスクリプト・バージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_jcs という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページ を参照してください。

参照

- 「スクリプトを追加または削除するためのシステム・プロシージャ」 698 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「ml_add_connection_script システム・プロシージャ」 702 ページ
- 「ml_add_table_script システム・プロシージャ」 716 ページ
- 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ
- 「ml_add_dnet_table_script システム・プロシージャ」 704 ページ
- 「ml_add_java_table_script システム・プロシージャ」 706 ページ
- 「メソッド」 559 ページ
- 「Java による同期スクリプトの作成」 553 ページ

例

次の例は、CustEmpScripts クラスの endConnection メソッドを end_connection イベントに対して登録します。

```
call ml_add_java_connection_script( 'ver1',
'end_connection',
'CustEmpScripts.endConnection' )
```

ml_add_java_table_script システム・プロシージャ

このシステム・プロシージャを使用して、Java メソッドをテーブル・イベント用のスクリプトとして登録したり、登録解除したりします。

構文

ml_add_java_table_script (
'*version***',**
'*table***',**
'*event***',**
'*script***'**
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
table	VARCHAR(128)。テーブル名。
event	VARCHAR(128)。イベント名。
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。

備考

メソッドを登録解除するには、スクリプトの内容パラメータを NULL に設定します。

script 値は、たとえば MyClass.MyMethod などの Mobile Link サーバのクラスパスに含まれるクラス内のパブリック・メソッドです。

ml_add_java_table_script を呼び出すと、メソッドが指定のテーブル、イベント、スクリプト・バージョンに関連付けられます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_jts という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「スクリプトを追加または削除するためのシステム・プロシージャ」 698 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「ml_add_connection_script システム・プロシージャ」 702 ページ
- 「ml_add_table_script システム・プロシージャ」 716 ページ
- 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ
- 「ml_add_dnet_table_script システム・プロシージャ」 704 ページ
- 「ml_add_java_connection_script システム・プロシージャ」 705 ページ
- 「メソッド」 559 ページ
- 「Java による同期スクリプトの作成」 553 ページ

例

次の例は、CustEmpScripts クラスの empDownloadCursor メソッドを emp テーブルの download_cursor イベントに対して登録します。

```
call ml_add_java_table_script( 'ver1', 'emp',
'download_cursor','CustEmpScripts.empDownloadCursor' )
```

ml_add_lang_connection_script システム・プロシージャ

このプロシージャは内部でのみ使用されます。

ml_add_lang_connection_script_chk システム・プロシージャ

このプロシージャは内部でのみ使用されます。

ml_add_lang_table_script システム・プロシージャ

このプロシージャは内部でのみ使用されます。

ml_add_lang_table_script_chk システム・プロシージャ

このプロシージャは内部でのみ使用されます。

ml_add_passthrough システム・プロシージャ

このシステム・プロシージャを使用して、スクリプトを実行するリモート・データベースを識別します。このプロシージャは、ml_passthrough システム・テーブルにエントリを追加します。指定の remote_id と run_order を持つエントリがすでにテーブルに存在している場合、このプロシージャはエントリを更新します。

構文

ml_add_passthrough (
'*remote_id***',**
'*script_name***',**
run_order
)

パラメータ

構文	説明
remote_id	VARCHAR(128)。スクリプトを実行するデータベースのリモート ID。この値は、特定のクライアントに適用する ml_database テーブル内の有効なリモート ID を指定できます。また、ml_database テーブルでリストされたすべてのスクリプト・クライアントに適用する場合は NULL を指定できます。 **警告** スクリプトをすべてまたは多数のリモートに適用する場合は、十分注意してください。スクリプトが適切に記述されていないと、ほとんどまたはすべてのリモートが破損したり、無効になったりする可能性があります。
script_name	VARCHAR(128)。サブスクライブされるスクリプトの名前。この値は、ml_passthrough_script テーブルで定義されている有効なスクリプト名にしてください。
run_order	INTEGER。run_order パラメータによって、スクリプトをリモート・データベースに適用する順序が決まります。スクリプトは常に、run_order に従って順序どおりに適用されます。各リモート・ストアは、適用を試みたがダウンロードまたは実行していない、より小さい run_order のスクリプトがない、最後のスクリプトの run_order を保存します。 この値は負でない整数にするか、NULL にします。

備考

run_order を NULL と定義した場合、プロシージャによって、remote_id の値に基づいた整数が代入されます。remote_id が NULL の場合、プロシージャによって、ml_passthrough 内の run_order の値に 10 を加えた値が代入されます。remote_id が NULL ではない場合、プロシージャによって、ml_passthrough テーブル内の remote_id のローの run_order の最大値に 10 を加えた値が代入されます。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_pt という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「ml_database」 740 ページ
- 「ml_passthrough」 747 ページ
- 「ml_passthrough_script」 750 ページ

ml_add_passthrough_repair システム・プロシージャ

このシステム・プロシージャを使用して、スクリプトのエラーを処理するためのルールを定義します。各ルールは、特定のスクリプトによって指定のエラー・コードが生成されたときにクライアントが実行するアクションを定義します。このプロシージャは、ml_passthrough_repair システム・テーブルにエントリを追加します。指定の failed_script_name と error_code を持つエントリがすでにテーブルに存在している場合、プロシージャはエントリを更新します。

構文

ml_add_passthrough_repair (
'*failed_script_name***',**
***error_code*,**
'*new_script_name***',**
'*action***'**
)

パラメータ

構文	説明
failed_script_name	VARCHAR(128)。このルールが適用される失敗したスクリプトの名前。この値は、ml_passthrough_script テーブル内の有効なスクリプト名にしてください。
error_code	INTEGER。このルールが処理する SQL Anywhere エラー・コード。
new_script_name	VARCHAR(128)。action が R のときは、失敗したスクリプトを置き換えるスクリプトの名前。action が S、P、または H の場合、この値は NULL にすることができます。action が R の場合、この値は ml_passthrough_script テーブル内の有効なスクリプト名にしてください。failed_script_name と同じにすることもできます。

構文	説明
action	CHAR(1)。failed_script_name のために error_code が生成されたときにクライアントが実行するアクション。この値は次のいずれかにしてください。 ● **R** (置換) 失敗したスクリプトを **new script name** によって指定されたスクリプトに置き換える必要があること、および新しいスクリプトの実行を試みる必要があることを示します。失敗したスクリプトを再実行するには、**new script name** を **failed script name** と同じにします。 ● **P** (パージ) リモート・データベースが受信したすべてのスクリプトを破棄し、その後に通常どおりスクリプトの実行を続行する必要があることを示します。 ● **S** (省略) リモート・データベースが失敗したスクリプトを無視し、成功した場合のようにスクリプトの実行を続行する必要があることを示します。 ● **H** (停止) リモート・データベースが追加の指示を受け取るまでそれ以上のスクリプトを実行しないようにすることを示します。

備考

スクリプトを徹底的にテストすることによって、SQL パススルー・スクリプトが失敗することをできるかぎり回避してください。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_pt_repair という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

● 「ml_passthrough_repair」 748 ページ
● 「ml_passthrough_script」 750 ページ

ml_add_passthrough_script システム・プロシージャ

このシステム・プロシージャを使用して、パススルー・スクリプトを作成します。このプロシージャは、ml_passthrough_script システム・テーブルにエントリを追加します。

構文

ml_add_passthrough_script (
'*script_name***',**
'*flags***',**
'*affected_pubs***',**
'*script***',**
'*description***'**
)

パラメータ

構文	説明
script_name	VARCHAR(128)。スクリプト名。この値は、ユニークにする必要があります。
flags	VARCHAR(256)。クライアントにスクリプトの実行方法を指定する値。この値は、NULL にしたり、セミコロンで区切られたリストによる次のキーワードの組み合わせを含めたりすることができます。 ● **manual** スクリプトが手動実行モードでのみ実行できることを示します。デフォルトでは、すべてのスクリプトは自動実行モードと手動実行モードのどちらでも実行できます。 ● **exclusive** スクリプトがすべての同期対象テーブルに対する排他ロックが取得された、同期の最後に自動的に実行できることを示します。affected_pubs 値にパブリケーションが 1 つもリストされていない場合、このオプションは無視されます。このオプションは、SQL Anywhere リモートでのみ有効です。 ● **schema_diff** スクリプトがスキーマ diff モードでのみ実行する必要があることを示します。このモードでは、スクリプトに記述されているスキーマに合わせてデータベース・スキーマが変更されます。たとえば、既存のテーブルに対する create 文は alter 文として扱われます。このフラグは、Ultra Light リモートで実行されるスクリプトにのみ適用されます。 次に例を示します。 'manual;exclusive;schema_diff'
affected_pubs	TEXT。スクリプトの実行前に同期する必要があるパブリケーションのリスト。空の文字列および NULL は、同期の必要がないことを示します。この値は、SQL Anywhere クライアントでのみ有効です。Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。

構文	説明

script	TEXT。パススルー・スクリプトの内容。この値は NULL にすることはできません。Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。
	
	スクリプトの内容を NULL にすることはできません。Ultra Light リモートの場合、**script** の内容は、単語 **go** で区切られた SQL 文のコレクションである必要があります。単語 **go** は、個別の行に記述してください。SQL Anywhere リモートの場合、**script** の内容は、**begin…end** ブロックで囲まれると有効になる、任意の SQL 文のコレクションにすることができます。
	
	SQL Anywhere リモートでの **script** の内容の例
	
	`DECLARE val INTEGER;`
	`SELECT c1 INTO val FROM t1 WHERE pk = 5;`
	`IF val > 100 THEN`
	`  INSERT INTO t2 VALUES ('c1 is big');`
	`ENDIF`
	
	Ultra Light リモートでの **script** の内容の例
	
	`CREATE TABLE myScript (c1 INT NOT NULL PRIMARY KEY)`
	`GO`
	`INSERT INTO myScript VALUES (1)`
	`GO`

description	VARCHAR(2000)。スクリプトのコメントまたは説明。この値は NULL にすることもできます。

備考

指定された script_name がすでに ml_passthrough_script に存在する場合、このプロシージャはエラーを生成します。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_pt_script という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「ml_passthrough_script」 750 ページ

ml_add_property システム・プロシージャ

このシステム・プロシージャを使用して、Mobile Link のプロパティを追加または削除します。このシステム・プロシージャは、ml_property システム・テーブルのローを変更します。

構文

ml_add_property (
'*comp_name*',

'*prop_set_name*',
'*prop_name*',
'*prop_value*'
)

パラメータ

構文	説明
comp_name	VARCHAR(128)。コンポーネント名。スクリプト・バージョンごとにプロパティを保存するには、ScriptVersion に設定します。Mobile Link サーバ・プロパティの場合は、MLS に設定します。サーバ起動同期のプロパティの場合は、SIS に設定します。
prop_set_name	VARCHAR(128)。プロパティ・セット名。 コンポーネント名が ScriptVersion である場合、このパラメータはスクリプト・バージョンの名前です。 コンポーネント名が MLS である場合、このパラメータは Mobile Link ユーザに対して冗長性を指定する ml_user_log_verbosity、またはリモート ID に対して冗長性を指定する ml_remote_id_log_verbosity にすることができます。 コンポーネント名が SIS である場合、このパラメータはプロパティを設定している Notifier、ゲートウェイ、または Carrier の名前です。
prop_name	VARCHAR(128)。プロパティ名。 コンポーネント名が ScriptVersion である場合、このパラメータは定義するプロパティです。DBConnectionContext の場合は getVersion や getProperties、ServerContext の場合は getPropertiesByVersion、getProperties、getPropertySetNames を使用して、このプロパティを参照できます。 コンポーネント名が MLS である場合、このパラメータは定義する Mobile Link ユーザ名またはリモート ID です。
prop_value	TEXT。プロパティの値。 prop_set_name が ml_user_log_verbosity または ml_remote_id_log_verbosity の場合は、有効な mlsrv -v オプションである必要があります。 Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。プロパティを削除するには、NULL に設定します。

対象 Mobile Link ユーザとリモート ID に対する ログの冗長性

対象とする Mobile Link ユーザまたはリモート ID に対して、Mobile Link サーバが異なるログの冗長性を使用するように設定できます。Mobile Link サーバは、5 分ごとに ml_property テーブルをチェックし、Mobile Link ユーザまたはリモート ID の冗長性設定を調べます。冗長性設定が存在する場合は、指定された Mobile Link ユーザまたはリモート ID に対する出力メッセージを記録

するために新しい設定が使用されます。これにより、サーバ・ファームに悪影響を与えるような高い冗長性設定を使用したり、ファーム内の各サーバを再起動したりしなくても、特定のユーザまたはリモート ID に関する詳細を確認できます。

対象 Mobile Link ユーザ (たとえば *ml_user1*) の冗長性を最大にするには、統合データベースにログインして次の SQL コマンドを発行します。

```
call ml_add_property( 'MLS', 'ml_user_log_verbosity', 'ml_user1', '-v+' )
```

対象リモート ID (たとえば *rid_1*) の冗長性を最大にするには、統合データベースにログインして次の SQL コマンドを発行します。

```
call ml_add_property( 'MLS', 'ml_remote_id_log_versity', 'rid_1', '-v+' )
```

verbose_setting は、Mobile Link サーバで有効な -v オプションにする必要があります。たとえばロー・データや未定義テーブル・スクリプトをログに記録するには、verbose_setting を -vru または vru に設定できます。Mobile Link サーバは 5 分後に ml_user1 または rid_1 に対してこの冗長性設定を使用します。「-v オプション」 104 ページを参照してください。

Mobile Link ユーザのログの冗長性を無効にするには、統合データベースにログインして次の SQL コマンドを発行します。

```
call ml_add_property( 'MLS', 'ml_user_log_verbosity', 'ml_user', NULL )
```

リモート ID のログの冗長性を無効にするには、統合データベースにログインして次の SQL コマンドを発行します。

```
call ml_add_property( 'MLS', 'ml_remote_id_log_verbosity', 'rid_1', NULL )
```

Mobile Link サーバは 5 分後に *ml_user* または *rid_1* に対して先の冗長性設定を使用停止します。

指定された Mobile Link ユーザやリモート ID に対して **ml_user_log_verbosity** と **ml_remote_id_log_verbosity** の「両方」が設定されている場合、および同期における Mobile Link ユーザ名やリモート ID が指定された対象 Mobile Link ユーザやリモート ID と同一である場合、Mobile Link サーバは出力メッセージを記録するときに **ml_remote_id_log_verbosity** 設定を使用します。

サーバ起動同期

サーバ起動同期では、ml_add_property システム・プロシージャを使用すると、Notifier、ゲートウェイ、Carrier のプロパティを設定できます。

たとえば、x という SMTP ゲートウェイのプロパティ server=mailserver1 を追加するには、次のように入力します。

```
ml_add_property( 'SIS','SMTP(x)','server','mailserver1' );
```

冗長なプロパティがすべての Notifier とゲートウェイに適用されるため、特定のプロパティ・セット名を指定することはできません。冗長性の設定を変更するには、次のようにプロパティ・セット名を空のままにします。

```
ml_add_property( 'SIS','','verbosity',2 );
```

スクリプト・バージョン

通常の Mobile Link 同期では、このシステム・プロシージャを使用して、プロパティをスクリプト・バージョンに関連付けることができます。この場合、component_name を ScriptVersion に設定します。任意のプロパティを指定し、Java クラスまたは .NET クラスを使用してプロパティにアクセスできます。

たとえば、LDAP サーバを MyVersion というスクリプト・バージョンに関連付けるには、次のように入力します。

```
ml_add_property( 'ScriptVersion','MyVersion','ldap-server','MyServer' )
```

参照

- 「ml_property」 753 ページ
- 「サーバ起動同期の Mobile Link サーバ設定」 『Mobile Link - サーバ起動同期』
- 「サーバ起動同期の Mobile Link サーバ設定」 『Mobile Link - サーバ起動同期』
- Java API DBConnectionContext：「getProperties メソッド」 572 ページと「getVersion メソッド」 574 ページ
- .NET API DBConnectionContext：「GetProperties メソッド」 641 ページと「GetVersion メソッド」 642 ページ
- Java API ServerContext：「getPropertiesByVersion メソッド」 597 ページ、「getProperties メソッド」 596 ページ、「getPropertySetNames メソッド」 597 ページ
- .NET ServerContext：「getPropertiesByVersion メソッド」 661 ページ、「getProperties メソッド」 661 ページ、「getPropertySetNames メソッド」 662 ページ

ml_add_table_script システム・プロシージャ

このシステム・プロシージャを使用して、SQL テーブル・スクリプトを統合データベースに追加するか、または統合データベースから削除します。

構文

ml_add_table_script (
'*version*',
'*table*',
'*event*',
'*script*'
)

パラメータ

構文	説明
version	VARCHAR(128)。バージョン名。
table	VARCHAR(128)。テーブル名。
event	VARCHAR(128)。イベント名。

構文	説明
script	TEXT。スクリプトの内容。Adaptive Server Enterprise の場合は VARCHAR(16384)。DB2 LUW の場合は VARCHAR(4000)。Oracle の場合は CLOB。

備考

テーブル・スクリプトを削除するには、スクリプトの内容パラメータを NULL に設定します。

スクリプトを追加すると、スクリプトが ml_script テーブルに挿入され、このスクリプトを指定のテーブル、イベント、スクリプト・バージョンに関連付ける適切な参照が定義されます。新しいバージョン名は、ml_version テーブルに自動的に挿入されます。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_add_ts という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「スクリプトを追加または削除するためのシステム・プロシージャ」 698 ページ
- 「スクリプトの追加と削除」 345 ページ
- 「ml_add_connection_script システム・プロシージャ」 702 ページ
- 「ml_add_dnet_connection_script システム・プロシージャ」 703 ページ
- 「ml_add_dnet_table_script システム・プロシージャ」 704 ページ
- 「ml_add_java_connection_script システム・プロシージャ」 705 ページ
- 「ml_add_java_table_script システム・プロシージャ」 706 ページ

例

次のコマンドは、Customer テーブルの upload_insert イベントに対応するテーブル・スクリプトを追加します。

```
call ml_add_table_script( 'default', 'Customer', 'upload_insert',
  'INSERT INTO Customer( cust_id, name, rep_id, active )
    VALUES ( {ml r.cust_id}, {ml r.name}, {ml r.rep_id}, 1 )' )
```

ml_add_user システム・プロシージャ

このプロシージャは内部でのみ使用されます。

ml_delete_passthrough システム・プロシージャ

このストアド・プロシージャは、指定の実行順序を持つ指定のスクリプトを指定のリモートにダウンロードさせる、ml_passthrough テーブル内のローを削除します。スクリプトは、削除される前にリモートにダウンロードされた場合、リモートから削除されずに通常どおり実行されます。

構文

ml_delete_passthrough (
'*remote_id***',**
'*script_name***',**
'*run_order***'**
)

パラメータ

構文	説明
remote_id	VARCHAR(128)。リモート ID。**remote_id** が NULL の場合、指定されたスクリプト名と実行順序について、ml_passthrough テーブル内のすべてのローが削除されます。
script_name	VARCHAR(128)。スクリプト名。
run_order	INTEGER。リモート・データベースで適用されるスクリプトの実行順序。**run_order** が NULL の場合、指定された **remote_id** と **script_name** について、実行順序に関係なくすべてのローが ml_passthrough テーブルから削除されます。

備考

Mobile Link サーバは、エントリを ml_passthrough テーブルから自動的には削除しません。古いパススルー・スクリプトを削除するには、このプロシージャを使用してください。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_del_pt という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「ml_passthrough」 747 ページ

ml_delete_passthrough_repair システム・プロシージャ

このシステム・プロシージャを使用して、修復ルールを ml_passthrough_repair システム・テーブルから削除します。

構文

ml_delete_passthrough_repair (
'*failed_script_name***',**
error_code
)

パラメータ

構文	説明
failed_script_name	VARCHAR(128)。このルールが適用されるスクリプトの名前。
error_code	INTEGER。ルールが適用されるエラー・コード。

備考

Mobile Link サーバは、エントリを ml_passthrough_repair テーブルから自動的には削除しません。古いパススルー修復スクリプトを削除するには、このプロシージャを使用してください。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_del_pt_repair という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「ml_passthrough_repair」 748 ページ

ml_delete_passthrough_script システム・プロシージャ

このシステム・プロシージャを使用して、パススルー・スクリプトを ml_passthrough_script システム・テーブルから削除します。

構文

ml_delete_passthrough_script (
'*script_name***'**
)

パラメータ

構文	説明
script_name	VARCHAR(128)。削除するスクリプトの名前。

備考

スクリプトは、ml_passthrough システム・テーブルまたは ml_passthrough_repair システム・テーブルで参照されている場合、削除できません。

Mobile Link サーバは、エントリを ml_passthrough_script テーブルから自動的には削除しません。古いパススルー・スクリプトを削除するには、このプロシージャを使用してください。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_del_pt_script という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「ml_passthrough」 747 ページ
- 「ml_passthrough_repair」 748 ページ
- 「ml_passthrough_script」 750 ページ

ml_delete_sync_state システム・プロシージャ

このプロシージャを使用して、未使用または不要な同期ステータスを削除します。

構文

ml_delete_sync_state (
'*user***',**
'*remote_id***'**
)

パラメータ

構文	説明
user	VARCHAR(128)。Mobile Link ユーザ名。
remote_id	VARCHAR(128)。リモート ID。

備考

これらのパラメータには NULL を指定できます。すべてのパラメータが NULL の場合、プロシージャは何も処理を実行しません。

このストアド・プロシージャは、指定された Mobile Link ユーザ名とリモート ID について、ml_subscription テーブルからすべてのローを削除します。指定されたリモート ID が ml_subscription テーブルのどのローからも参照されなくなった場合は、ml_database テーブルからそのリモート ID も削除します。

リモート ID が NULL で、Mobile Link ユーザ名が NULL でない場合、指定された Mobile Link ユーザ名で参照されるすべてのローを ml_subscription テーブルから削除します。また、ml_subscription テーブル内のどのローからも参照されなくなったすべてのリモート ID を ml_database テーブルから削除します。

Mobile Link ユーザ名が NULL で、リモート ID が NULL でない場合、このストアド・プロシージャは、指定されたリモート ID について、ml_subscription テーブルと ml_database テーブルからすべてのローを削除します。

すべてのリモート ID が ml_database テーブルから削除され、この Mobile Link ユーザが ml_subscription テーブル内のどのローからも参照されなくなっても、このユーザはこのストアド・プロシージャによって削除されません。この Mobile Link ユーザを削除する必要がある場合は、次のようなコマンドを発行して削除できます。

```
delete from ml_user where name = 'user_name'
```

ここで、*user_name* は、削除する Mobile Link ユーザです。

このストアド・プロシージャは、細心の注意を払って使用してください。次回 Mobile Link クライアントがこのリモート ID の同期を要求したとき、Mobile Link サーバは、同期ステータスをチェックしないで、このリモート ID を ml_database テーブルと ml_subscription テーブルに自動的に追加します。前回行われた同期が成功しなかったリモート ID の同期ステータスを削除すると、データの不整合が発生する場合があります。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_del_sstate という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「ml_subscription」 775 ページ
- 「ml_database」 740 ページ

例

次の例は、John という Mobile Link ユーザのリモート ID remote_db_for_John を持つリモート・データベースに関する Mobile Link システム・テーブル情報をクリーンアップします。

```
CALL ml_delete_sync_state( 'John', 'remote_db_for_John' )
```

ml_delete_sync_state_before システム・プロシージャ

このプロシージャを使用して、リモート・データベースを削除したときに Mobile Link システム・テーブルをクリーンアップします。

構文

ml_delete_sync_state_before (
'*ts*'
)

パラメータ

構文	説明
ts	TIMESTAMP。日時は、統合データベースで指定された順序で指定してください。統合データベースの日時のフォーマットが "yyyy/mm/dd hh:mm:ss.ssss" に設定されている場合、タイムスタンプは年、月、日、時、分、秒、秒以下の順に指定します。

備考

このストアド・プロシージャは Mobile Link システム・テーブルから、もう使用されていないリモート・データベースに関連するローを削除します。特に次の処理が行われます。

- ml_subscription システム・テーブルから、last_upload_time と last_download_time の両方が指定のタイムスタンプより前の値になっているすべてのローを削除する。

- リモート ID が ml_subscription テーブルのどのローからも参照されていない場合は、ml_database システム・テーブルからリモート ID を削除する。

実際には削除されていないリモート・データベースのローを削除する可能性がある場合は、このシステム・プロシージャを使用しないでください。もし使用すると、ml_subscription と ml_database テーブル内のローが削除されることで、アップロードが失敗して「不明なステータス」になっているリモート・データベースに問題が発生する可能性があります。その不明なステータスでは、リモート・データベースは Mobile Link システム・テーブルに依存してデータを再送します。

プロシージャはパラメータの日付／時刻フォーマットを検証しないので、このプロシージャに指定するタイムスタンプには正しい日付／時刻フォーマットを使用してください。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_del_sstate_b4 という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページを参照してください。

参照

- 「ml_subscription」 775 ページ
- 「ml_database」 740 ページ

例

次の例は、2004 年 1 月 10 日以降同期されていないリモート・データベースに関する Mobile Link システム・テーブル情報をクリーンアップします。この例は、日付／時刻フォーマットが yyyy/mm/dd hh:mm:ss.ssss である SQL Anywhere 統合データベースで使用できます。

```
CALL ml_delete_sync_state_before( '2004/01/10 00:00:00' )
```

ml_delete_user システム・プロシージャ

このプロシージャは内部でのみ使用されます。

ml_reset_sync_state システム・プロシージャ

このプロシージャを使用して、Mobile Link システム・テーブル内の同期ステータス情報をリセットします。

構文

ml_reset_sync_state (
'*user*',
'*remote_id*'
)

パラメータ

構文	説明
user	VARCHAR(128)。Mobile Link ユーザ名。
remote_id	VARCHAR(128)。リモート ID。

備考

パラメータには NULL を指定できます。両方のパラメータが NULL の場合、このプロシージャは何も処理を実行しません。

このストアド・プロシージャは、ml_subscription テーブルの progress、last_upload_time、last_download_time カラムを、指定のユーザ名とリモート ID のデフォルト値に設定します。progress のデフォルト値は 0 です。last_upload_time と last_download_time カラムのデフォルト値は '1900/01/01 00:00:00' です。

リモート ID が NULL で、Mobile Link ユーザ名が NULL でない場合、このプロシージャはこれらのカラムを、指定の Mobile Link ユーザ名によって参照される ml_subscription テーブル内のローのデフォルト値に設定します。Mobile Link ユーザ名が NULL で、リモート ID が NULL でない場合には、指定のリモート ID を持つ ml_subscription テーブル内のローのデフォルト値に設定します。

このストアド・プロシージャは、細心の注意を払って使用してください。次回 Mobile Link クライアントがこのリモート ID の同期を要求したとき、Mobile Link サーバは、このリモート ID の同期ステータスをチェックしません。前回行われた同期が成功しなかったリモート ID をリセットすると、データの不整合が発生する場合があります。

統合データベースのタイプが DB2 メインフレームの場合、このプロシージャは ml_reset_sstate という名前になります。「IBM DB2 メインフレームのシステム・プロシージャ名の変換」 699 ページ を参照してください。

ml_server_delete システム・プロシージャ

このプロシージャは内部でのみ使用されます。

ml_server_update システム・プロシージャ

このプロシージャは内部でのみ使用されます。

Mobile Link ユーティリティ

目次

Mobile Link ユーティリティの概要

Mobile Link サーバ・ユーティリティには次の 2 つがあります。

- 「Mobile Link 停止ユーティリティ (mlstop)」 727 ページ
- 「Mobile Link ユーザ認証ユーティリティ (mluser)」 729 ページ

次の項目も参照してください。

- Mobile Link クライアント・ユーティリティ：「Mobile Link クライアント・ユーティリティ」『Mobile Link - クライアント管理』
- Ultra Light ユーティリティ：「Ultra Light ユーティリティ」 『Ultra Light データベース管理とリファレンス』
- TLS 証明書を使用するためのユーティリティ：「証明書ユーティリティ」 『SQL Anywhere サーバ - データベース管理』
- SQL Anywhere のその他のユーティリティ：「データベース管理ユーティリティ」 『SQL Anywhere サーバ - データベース管理』

Mobile Link 停止ユーティリティ (mlstop)

ローカル・コンピュータ上の Mobile Link サーバを停止します。

構文

mlstop [*options*] [*name*]

オプション	説明
@data	これを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。「設定ファイルの使用」 『SQL Anywhere サーバ - データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化できます。「ファイル難読化ユーティリティ (dbfhide)」 『SQL Anywhere サーバ - データベース管理』を参照してください。
-f	強制シャットダウン。ハード・シャットダウンが機能しない場合に使用します。
-h	ハード・シャットダウン。Mobile Link がすべての同期を停止して終了します。リモートによっては、エラーをレポートする場合があります。
-q	クワイエット・モード。このモードにすると、バナーは表示されません。
-t *time*	ソフト・シャットダウン。ただし、指定の時間が経過したらハード・シャットダウン。*time* は数字で、後ろに D、H、M、または S (日、時間、分、秒) が続きます。たとえば、-t 10m と指定すると、10 分後または現在の同期が完了した時点 (早い方) で、サーバがシャットダウンされます。D、H、M、S の大文字と小文字は区別されません。
-w	コマンドから戻る前に、Mobile Link サーバがシャットダウンされるのを待機します。
name	-zs オプションで Mobile Link サーバを起動した場合は、同じサーバ名を指定してシャットダウンしてください。「-zs オプション」 123 ページを参照してください。

説明

デフォルト (-f、-h、-t のいずれも指定されていない場合) では、mlstop によってソフト・シャットダウンが行われます。

- **ソフト・シャットダウン**　これは、現在の同期が完了すると、Mobile Link サーバが新しい接続を受け入れずに終了することを意味します。
- **ハード・シャットダウン**　これは、Mobile Link サーバがすべての同期を停止して終了することを意味します。リモートによっては、エラーをレポートする場合があります。

Mobile Link ユーザ認証ユーティリティ (mluser)

統合データベース側で Mobile Link ユーザを登録します。SQL Anywhere リモートの場合は、CREATE SYNCHRONIZATION USER 文を使用して、リモート・データベース側でユーザを事前に作成しておきます。

構文

mluser [*options*] **-c** "*connection-string*"
{ **-f** *file* | **-u** *user* [**-p** *password*] }

オプション	説明
@*data*	これを使用すると、指定された環境変数または設定ファイルからオプションを読み込むことができます。同じ名前の環境変数と設定ファイルが両方存在する場合は、環境変数が使用されます。「設定ファイルの使用」『SQL Anywhere サーバ - データベース管理』を参照してください。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル難読化ユーティリティを使用して、設定ファイルの内容を難読化できます。「ファイル難読化ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』を参照してください。
-c "*keyword=value;...*"	これを使用して、データベース接続パラメータを指定します。ODBC データ・ソースを使用して統合データベースに接続するには、接続文字列にユーティリティのパーミッションを指定します。このパラメータは必須です。
-d	-f または -u で指定したユーザ名を削除します。
-f *filename*	指定したファイルから、ユーザ名とパスワードを読み込みます。ファイルは、1 行ごとにユーザ名とパスワードが 1 つずつ、空白スペースで区切って指定されているものを使用します。-f または -u を指定してください。
-fips	設定すると、FIPS のサポートがインストールされていない場合に mluser が失敗します。
-o *filename*	指定したファイルに出力メッセージのログを取ります。
-ot *filename*	ログ・ファイルをトランケートし、このファイルに出力メッセージを追加します。デフォルトでは、画面に出力を送信します。

オプション	説明
-p *password*	ユーザと関連付けるパスワード。このオプションは、-u を指定した場合のみ使用できます。
-pc *collation-id*	ユーザ名とパスワードの文字セット変換用のデータベース照合 ID を指定します。ここには、「サポートされている照合と代替照合」『SQL Anywhere サーバ - データベース管理』にリストされている SQL Anywhere 照合ラベルの 1 つを指定します。このオプションは、ファイルから読み込まれるユーザ名とパスワードが、ロケールで決定されるデフォルトの文字セットとは異なる文字セットでエンコードされている場合に必要です。
-u *ml_username*	追加するユーザ名を指定します (-d と一緒に使用した場合、ユーザを削除します)。1 行のコマンド・ラインで指定できるユーザは 1 人だけです。パスワードを使用している場合は、このオプションを -p とともに使用します。-f または -u を指定してください。
-v	冗長ロギングを指定します。

備考

ユーザとパスワードのペアを指定すると、mluser ユーティリティはまずそのユーザを追加しようとします。ユーザをすでに統合データベースに追加してある場合は、そのユーザのパスワードを更新します。

ユーザ名を統合データベースに登録するには、別の方法も使用できます。

- Sybase Central を使用する。
- mlsrv11 で -zu+ コマンド・ライン・オプションを指定する。この場合、最初に同期するときに、統合データベースに追加されていない既存の Mobile Link ユーザが追加されます。

追加されるのは、リモート・データベースにすでに存在する Mobile Link ユーザです。リモート・データベース側でユーザを追加する場合、次のオプションがあります。

- SQL Anywhere リモートの場合、CREATE SYNCHRONIZATION USER を使用して名前を設定し、そのユーザ名で同期する。
- Ultra Light リモートの場合は、ul_synch_info 構造体の user_name フィールドを使用するか、Java で、ULSynchInfo クラスの SetUserName() メソッドを使用してから同期する。

参照

- 「Mobile Link ユーザ」 『Mobile Link - クライアント管理』
- 「-zu オプション」 125 ページ
- 「CREATE SYNCHRONIZATION USER 文 [Mobile Link]」 『SQL Anywhere サーバ - SQL リファレンス』
- 「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』

Mobile Link サーバのシステム・テーブル

目次

Mobile Link システム・テーブルの概要

Mobile Link システム・テーブルには、Mobile Link ユーザ、サブスクリプション、テーブル、スクリプト、スクリプト・バージョン、その他の情報が保管されています。Mobile Link システム・テーブルは Mobile Link 同期に必須です。他のシステム・テーブルと異なり、Mobile Link システム・テーブルは修正できますが、たいていの場合は修正する必要はありません。

Mobile Link システム・テーブルは、使用している統合データベース用の Mobile Link 設定スクリプトを実行したときに作成されます。Mobile Link システム・テーブルは統合データベースに格納されます。設定スクリプトを実行するデータベース・ユーザが、スクリプトによって作成される Mobile Link システム・テーブルの所有者になります。

「統合データベースの設定」 6 ページを参照してください。

注意

- この章では、SQL Anywhere 統合データベースの Mobile Link システム・テーブルで使用されるデータ型について説明します。一部の RDBMS では、データ型が多少異なります。
- DB2 メインフレームのバージョン 8.1 は下位互換性モードをサポートしており、カラム名とその他の識別子が最大 18 文字までに制限されています。この環境をサポートするためには、DB2 メインフレーム内のすべての Mobile Link システム・オブジェクトの名前を 18 文字以下にする必要があります。「IBM DB2 メインフレームのシステム・テーブル名の変換」 736 ページを参照してください。
- IBM DB2 LUW バージョン 5.2 では、カラム名と他の識別子に 18 文字までしか使用できません。DB2 LUW 5.2 統合データベースでは、Mobile Link システム・テーブル名が必要に応じてトランケートされます。

IBM DB2 メインフレームのシステム・テーブル名の変換

IBM DB2 メインフレーム統合データベースでは、カラム名と他の識別子に 18 文字までしか使用できません。次の表は、DB2 メインフレーム統合データベースのシステム・テーブル名が、他のすべてのタイプの統合データベースのシステム・テーブル名にどのようにマッピングされるのかを示します。

システム・テーブル名が次の表にない場合、変換は必要ありません。

システム・テーブル名	DB2 メインフレーム統合データベースのシステム・テーブル名
「ml_active_remote_id」 737 ページ	ml_active_rid
「ml_connection_script」 739 ページ	ml_conn_script
「ml_passthrough」 747 ページ	ml_pt
「ml_passthrough_repair」 748 ページ	ml_pt_repair
「ml_passthrough_script」 750 ページ	ml_pt_script
「ml_passthrough_status」 752 ページ	ml_pt_status
「ml_scripts_modified」 772 ページ	ml_script_modified

ml_active_remote_id

サーバ・ファーム内の各リモート・データベースの同期ステータスを格納します。

カラム	説明
remote_id	VARCHAR(128)。現在同期中のリモート・データベースの ID を識別するユニークな整数。
server_id	INTEGER。ml_server テーブルの server_id を参照する値。
status	CHAR(1)。リモート・データベースの同期ステータス。値 O は継続中の同期を示し、C はキャンセルされた同期を示します。

備考

サーバ・ファームが実行されていない場合、このテーブルにはデータはありません。

統合データベースのタイプが DB2 メインフレームの場合、このテーブルは ml_active_rid という名前になります。「IBM DB2 メインフレームのシステム・テーブル名の変換」 736 ページを参照してください。

制約

PRIMARY KEY(remote_id)

FOREIGN KEY(server_id) REFERENCES ml_server(server_id)

ml_column

特定のスクリプト・バージョンに特定のテーブルのカラム名を格納します。

カラム	説明
version_id	INTEGER。スクリプト・バージョンを識別する番号。
table_id	INTEGER。テーブルを識別する番号。
idx	INTEGER。テーブル内のこのカラムの 1 から始まるインデックス。カラムの順序は、リモート・データベースでカラムが作成された順序である必要があります。
name	VARCHAR(128)。カラムの名前。
type	VARCHAR(128)。現在は使用されていません。

このテーブルは、SQL スクリプトに、カラムの名前付きパラメータ (o.column-name、r.column-name など) が含まれる場合にのみ必要です。ただし、カラムのインデックス (o.column-index、r.column-index など) は、この Mobile Link システム・テーブルに設定されなくても使用できます。

このテーブルは、**同期モデル作成ウィザード**で Mobile Link モデルを配備するときに設定されます。**同期モデル作成ウィザード**を使用しなかった場合、または**同期モデル作成ウィザード**を使用したが、後で Sybase Central モデル・モード以外でリモート・データベースの同期カラムのスキーマを変更した場合は、ml_add_column ストアド・プロシージャを使用してテーブルに設定できます。

注意：dbmlsync の拡張オプション SendColumnNames と Ultra Light の同期パラメータ Send Column Names はダイレクト・ロー・ハンドリングで使用されますが、名前付きロー・パラメータでは使用されません。

備考

このテーブルの内容を簡単に表示できるシステム・ビュー (ml_columns) があります。

制約

PRIMARY KEY(idx, version_id, table_id)

UNIQUE(version_id, table_id, name)

FOREIGN KEY(version_id) REFERENCES ml_script_version(version_id)

FOREIGN KEY(table_id) REFERENCES ml_table(table_id)

参照

- 「ml_add_column system procedure」 700 ページ
- 「スクリプトのパラメータ」 336 ページ

ml_connection_script

指定されたスクリプト・バージョンにおいて、このテーブルはスクリプトを指定されたイベントに関連付けます。

カラム	説明
version_id	INTEGER。スクリプト・バージョンを識別する番号。
event	VARCHAR(128)。接続スクリプトをトリガするイベントの名前。
script_id	INTEGER。スクリプトを識別する番号。接続スクリプトのテキストは、ml_script システム・テーブルに格納されます。

備考

このテーブルの内容を簡単に表示できるシステム・ビュー (ml_connection_scripts) があります。

統合データベースのタイプが DB2 メインフレームの場合、このテーブルは ml_conn_script という名前になります。「IBM DB2 メインフレームのシステム・テーブル名の変換」 736 ページを参照してください。

制約

PRIMARY KEY(version_id, event)

FOREIGN KEY(version_id) REFERENCES ml_script_version(version_id)

FOREIGN KEY(script_id) REFERENCES ml_script(script_id)

ml_database

同期された各リモート・データベースのユニークな ID を格納します。

警告
このテーブルは変更しないでください。

カラム	説明
rid	INTEGER。リモート ID を識別するユニークな整数。この値は内部で使用されます。
remote_id	VARCHAR(128)。各リモート・データベースを識別するユニークなリモート ID。
script_ldt	TIMESTAMP。パススルー・スクリプトが最後にダウンロードされた時刻。
description	VARCHAR(128)。予約。

備考

リモート ID は、各同期でクライアントによって送信されます。Mobile Link サーバは、このリモート ID を使用して各リモート・データベースのステータス情報を追跡します。

制約

PRIMARY KEY(rid)

ml_device

このテーブルは、サーバ起動同期にのみ使用されます。また、このテーブルにはデバイスの追跡に必要なデバイス名が格納されます。

カラム	説明
device_name	VARCHAR(255)。デバイスに指定された名前。dblsn -e オプションを使用して名前を指定した場合を除き、この名前はオペレーティング・システムから取り出されます。
listener_version	VARCHAR(128)。NOT NULL。デバイスにインストールされているソフトウェア用の SQL Anywhere のバージョン番号。この値を変更してもソフトウェアの動作には影響しませんが、この値は診断時に役立つことがあります。
listener_protocol	INTEGER。NOT NULL。このカラムは次の 0、1、2 のいずれかです。 ● **0** - バージョン 9.0.1 より前の SQL Anywhere の Listener の場合 ● **1** - 9.0.0 より後の Palm Listener の場合 ● **2** - 9.0.0 より後の Windows Listener の場合
info	VARCHAR(255)。NOT NULL。リスニング・デバイスのオペレーティング・システム情報。この情報は、dblsn -f オプションを使用して情報を提供することによって上書きできます。
ignore_tracking	VARCHAR(1)。NOT NULL。このカラムが **y** である場合は、追跡情報はローに書き込まれません。このカラムが **n** である場合は、追跡情報がローに書き込まれます。
source	VARCHAR(255)。NOT NULL。デバイスの自動追跡機能によってローが作成された場合、このカラムは **tracking** です。そうでない場合は、このカラムは空です。ただし、ストアド・プロシージャを使用してこのカラムを変更し、このローのデータの取得先に関する情報を追加した場合を除きます。このカラムが **tracking** に設定されていない場合は、カラムの値がソフトウェアの動作に影響することはありません。

備考

Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening には、サーバ起動同期に使用されるデバイスに関する情報が格納されています。DeviceTracker ゲートウェイは、この情報を使用して、Mobile Link ユーザ名ごとにターゲット・デバイスを処理します。

ほとんどの場合、これらのテーブルを変更する必要はありません。ただし、デバイスがデバイス追跡機能をサポートしていない場合、またはトラブルシューティングのためにデバイス追跡機能を無効にする場合は、事前に定義されたストアド・プロシージャを使用して、このシステム・テーブルのローを追加または削除できます。「デバイス・トラッキングのサポートの追加」『Mobile Link - サーバ起動同期』を参照してください。

自動追跡機能を停止する場合は、ignore_tracking を **y** に設定します。また、この場合は **tracking** 以外のソース名を使用してください。

制約

PRIMARY KEY(device_name)

参照

- 「ml_set_device システム・プロシージャ」 『Mobile Link - サーバ起動同期』
- 「ml_delete_device システム・プロシージャ」 『Mobile Link - サーバ起動同期』

ml_device_address

このテーブルは、サーバ起動同期にのみ使用されます。また、このテーブルにはデバイスの追跡に必要なアドレス情報が格納されます。

カラム	説明
device_name	VARCHAR(255)。NOT NULL。デバイスの名前。dblsn -e オプションを使用して名前を指定した場合を除き、この名前はオペレーティング・システムから取り出されます。
medium	VARCHAR(255)。NOT NULL。UDP では、**_UDP_** です。それ以外の場合、ネットワーク・プロバイダ ID です。
address	VARCHAR(255)。NOT NULL。UDP では、*ip*:*port-number* です。この場合、*ip* は IP アドレスまたはホスト名です。SMS では、電話番号です。
active	VARCHAR(1)。NOT NULL。アクティブな場合は **y** で、それ以外の場合は **n** です。DeviceTracker ゲートウェイは、UDP チャネルの応答がなく、予備の SMS 配信パスがある場合に UDP チャネルを非アクティブにすることがあります。
last_modified	タイムスタンプ。NOT NULL。デフォルトのタイムスタンプ。このローが最後に更新された日時。
ignore_tracking	VARCHAR(1)。NOT NULL。このカラムが **y** である場合は、追跡情報はローに書き込まれません。このカラムが **n** である場合は、追跡情報がローに書き込まれます。
source	VARCHAR(255)。NOT NULL。デバイスの自動追跡機能によってローが作成された場合、このカラムは **tracking** です。そうでない場合は、このカラムは空です。ただし、ストアド・プロシージャを使用してこのカラムを変更し、このローのデータの取得先に関する情報を追加した場合を除きます。このカラムが **tracking** に設定されていない場合は、カラムの値がソフトウェアの動作に影響することはありません。

備考

Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening には、サーバ起動同期に使用されるデバイスに関する情報が格納されています。DeviceTracker ゲートウェイは、この情報を使用して、Mobile Link ユーザ名ごとにターゲット・デバイスを処理します。

ほとんどの場合、これらのテーブルを変更する必要はありません。ただし、デバイスがデバイス追跡機能をサポートしていない場合、またはトラブルシューティングのためにデバイス追跡機能を無効にする場合は、事前に定義されたストアド・プロシージャを使用して、このシステム・テーブルのローを追加または削除できます。「デバイス・トラッキングのサポートの追加」『Mobile Link - サーバ起動同期』を参照してください。

自動追跡機能を停止する場合は、ignore_tracking を **y** に設定します。また、この場合は **tracking** 以外のソース名を使用してください。

制約

PRIMARY KEY(device_name, medium)

FOREIGN KEY(device_name) REFERENCES ml_device(device_name)

参照

- 「ml_set_device_address システム・プロシージャ」 『Mobile Link - サーバ起動同期』
- 「ml_delete_device_address システム・プロシージャ」 『Mobile Link - サーバ起動同期』

ml_listening

このテーブルは、サーバ起動同期にのみ使用されます。また、Mobile Link ユーザ名を、デバイス追跡用のデバイス名にマッピングします。

カラム	説明
name	VARCHAR(128)。NOT NULL。通知のアドレス指定に使用する名前。このカラムは、次のいずれかの方法で値を入力できます。 ● dblsn -t+ オプションを使用する場合は、ml_user に対して定義されているエイリアスです。 ● dblsn -u オプションを使用する場合は、ml_user の名前です。 ● -t+ と -u のどちらも使用しない場合は、デフォルト名は *device-name*-**dblsn** です。ここで、*device-name* はデバイスの名前です。デバイス名は、Listener メッセージ・ウィンドウ内で検索できます。必要に応じて、dblsn -e オプションを使用してデバイス名を設定できます。 「Windows 用の Listener オプション」『Mobile Link - サーバ起動同期』を参照してください。
device_name	VARCHAR(255)。NOT NULL。デバイスに指定された名前。dblsn -e オプションを使用して名前を指定した場合を除き、この名前はオペレーティング・システムから取り出されます。
listening	VARCHAR(1)。NOT NULL。アクティブな Listener の場合は **y** で、それ以外の場合は **n** です。このフィールドは、dblsn オプション -t を使用したときに設定されます。また、ストアド・プロシージャを使用して手動で設定することもできます。
ignore_tracking	VARCHAR(1)。NOT NULL。このカラムが **y** である場合は、追跡情報はローに書き込まれません。このカラムが **n** である場合は、追跡情報がローに書き込まれます。
source	VARCHAR(255)。NOT NULL。デバイスの自動追跡機能によってローが作成された場合、このカラムは **tracking** です。そうでない場合は、このカラムは空です。ただし、ストアド・プロシージャを使用してこのカラムを変更し、このローのデータの取得先に関する情報を追加した場合を除きます。このカラムの値は、ソフトウェアの動作に影響しません。

備考

Mobile Link システム・テーブル ml_device、ml_device_address、ml_listening には、サーバ起動同期に使用されるデバイスに関する追跡情報が格納されています。DeviceTracker ゲートウェイは、この情報を使用して、Mobile Link ユーザ名ごとにターゲット・デバイスを処理します。

ほとんどの場合、これらのテーブルを変更する必要はありません。ただし、デバイスがデバイス追跡機能をサポートしていない場合、またはトラブルシューティングのためにデバイス追跡機能

を無効にする場合は、事前に定義されたストアド・プロシージャを使用して、このテーブルのローを追加または削除できます。「デバイス・トラッキングのサポートの追加」 『Mobile Link - サーバ起動同期』を参照してください。

自動追跡機能を停止する場合は、ignore_tracking を **y** に設定します。また、この場合は **tracking** 以外のソース名を使用してください。

制約

PRIMARY KEY(name)

FOREIGN KEY(device_name) REFERENCES ml_device(device_name)

参照

- 「ml_set_listening システム・プロシージャ」 『Mobile Link - サーバ起動同期』
- 「ml_delete_listening システム・プロシージャ」 『Mobile Link - サーバ起動同期』

ml_passthrough

各リモート・データベースで実行するスクリプトを示すローを格納します。

カラム	説明
remote_id	VARCHAR(128)。スクリプトを実行するクライアントのリモート ID。
run_order	INTEGER。このクライアントのスクリプトの実行順序。この値は負でない必要があります。
script_id	INTEGER。ml_passthrough_script テーブルの script_id を参照する整数。この値は、実行するスクリプトを識別します。
last_modified	TIMESTAMP。パススルー情報が最後に追加または変更された時刻。

備考

システム・プロシージャ ml_add_passthrough および ml_delete_passthrough を使用して、このテーブルのエントリを追加、変更、削除できます。

統合データベースのタイプが DB2 メインフレームの場合、このテーブルは ml_pt という名前になります。「IBM DB2 メインフレームのシステム・テーブル名の変換」 736 ページを参照してください。

制約

PRIMARY KEY(remote_id, run_order)

FOREIGN KEY(remote_id) REFERENCES ml_database(remote_id)

FOREIGN KEY(script_id) REFERENCES ml_passthrough_script(script_id)

参照

- 「ml_add_passthrough システム・プロシージャ」 708 ページ
- 「ml_delete_passthrough システム・プロシージャ」 717 ページ

ml_passthrough_repair

クライアントがスクリプトのエラーを処理する方法を定義するルールを格納します。

カラム	説明
failed_script_id	INTEGER。このルールが適用されるスクリプトを識別します。この値は、ml_passthrough_script テーブルの script_id を参照します。
error_code	INTEGER。このルールが処理するエラー・コード。
new_script_id	INTEGER。ml_passthrough_script テーブルで定義され、修復スクリプトを表す script_id。action が R の場合、この値は、失敗したスクリプトを置き換えるスクリプトを識別します。値は、ml_passthrough_script テーブルの script_id カラムを参照します。action が S、P、または H の場合、この値は NULL です。
action	CHAR(1)。スクリプトが失敗したときにクライアントで実行するアクション。この値は次のいずれかにしてください。 ● **R** (置換) 失敗したスクリプトを **new script name** によって指定されたスクリプトに置き換える必要があること、および新しいスクリプトの実行を試みる必要があることを示します。失敗したスクリプトを再実行するには、**new script name** を **failed script name** と同じにします。 ● **P** (パージ) リモート・データベースが受信したすべてのスクリプトを破棄し、その後に通常どおりスクリプトの実行を続行する必要があることを示します。 ● **S** (省略) リモート・データベースが失敗したスクリプトを無視し、成功した場合のようにスクリプトの実行を続行する必要があることを示します。 ● **H** (停止) リモート・データベースが追加の指示を受け取るまでそれ以上のスクリプトを実行しないようにすることを示します。

備考

システム・プロシージャ ml_add_passthrough_repair と ml_delete_passthrough_repair を使用して、このテーブルのエントリを追加、変更、削除できます。

統合データベースのタイプが DB2 メインフレームの場合、このテーブルは ml_pt_repair という名前になります。「IBM DB2 メインフレームのシステム・テーブル名の変換」 736 ページを参照してください。

制約

PRIMARY KEY(failed_script_id, error_code)

FOREIGN KEY(failed_script_id) REFERENCES ml_passthrough_script(script_id)

参照

- 「ml_add_passthrough_repair システム・プロシージャ」 710 ページ
- 「ml_delete_passthrough_repair システム・プロシージャ」 718 ページ

ml_passthrough_script

各パススルー・スクリプトの名前と内容を格納します。

カラム	説明
script_id	INTEGER。パススルー・スクリプト名を識別するユニークな整数。この値は内部で使用されます。
script_name	VARCHAR(128)。ユニークなパススルー・スクリプト名。
flags	VARCHAR(256)。この値は、クライアントにスクリプトの実行方法を指定します。この値は、NULL にしたり、セミコロンで区切られたリストによる次のキーワードの組み合わせを含めたりすることができます。 ● **manual** スクリプトが手動実行モードでのみ実行できることを示します。デフォルトでは、すべてのスクリプトは自動実行モードと手動実行モードのどちらでも実行できます。 ● **exclusive** スクリプトがすべての同期対象テーブルに対する排他ロックが取得された、同期の最後に自動的に実行できることを示します。affected_pubs 値にパブリケーションが 1 つもリストされていない場合、このオプションは無視されます。このオプションは、SQL Anywhere リモートでのみ有効です。 ● **schema_diff** スクリプトがスキーマ diff モードでのみ実行する必要があることを示します。このモードでは、スクリプト内に記述されたスキーマと一致するようにデータベース・スキーマが変更されます。たとえば、既存のテーブルに対する create 文は alter 文として扱われます。このフラグは、Ultra Light リモートで実行されるスクリプトにのみ適用されます。 次に例を示します。 'manual;exclusive;schema_diff'
affected_pubs	TEXT。スクリプトの実行前に同期する必要があるパブリケーションのリスト。空および NULL のパブリケーション・リストは、同期の必要がないことを示します。この値は、SQL Anywhere クライアントでのみ有効です。
script	TEXT。パススルー・スクリプトの内容。
description	VARCHAR(2000)。スクリプトのコメントまたは説明。

備考

次のプロシージャを使用して、このテーブルのエントリを追加および削除できます。

- 「ml_add_passthrough_script システム・プロシージャ」 711 ページ
- 「ml_delete_passthrough_script システム・プロシージャ」 719 ページ

システム・プロシージャ ml_add_passthrough_script および ml_delete_passthrough_script を使用せずに ml_passthrough_script テーブルを直接更新しないでください。クライアントは、新しく更新されたスクリプトを受信できない場合があります。元のパススルー・スクリプトをすでにダウンロードしている場合がそうです。複数のクライアントのスクリプトの間で不一致が発生すると、スクリプト実行の管理が難しくなったり、不可能になったりします。

統合データベースのタイプが DB2 メインフレームの場合、このテーブルは ml_pt_script という名前になります。「IBM DB2 メインフレームのシステム・テーブル名の変換」 736 ページを参照してください。

制約

PRIMARY KEY(script_id)

参照

- 「ml_add_passthrough_script システム・プロシージャ」 711 ページ
- 「ml_delete_passthrough_script システム・プロシージャ」 719 ページ

ml_passthrough_status

クライアントによって実行された後の、各パススルー・スクリプトのステータスを格納します。

カラム	説明
status_id	INTEGER。ローを識別するユニークな整数。
remote_id	VARCHAR(128)。スクリプトを実行したリモート・データベースを識別します。ml_database テーブルの remote_id を参照します。
run_order	INTEGER。クライアントのスクリプトの実行順序。
script_id	INTEGER。実行されたスクリプトを識別します。ml_passthrough_script テーブルの script_id を参照します。
script_status	CHAR(1)。スクリプトのステータス。成功の場合は **S**、エラーの場合は **E**。
error_code	INTEGER。スクリプトによって生成された、リモート・データベースでの SQL コード。
error_text	TEXT。スクリプトによって生成された、リモート・データベースでのエラー・テキスト。
remote_run_time	TIMESTAMP。スクリプトが実行された、リモート・データベースでの時刻。

備考

Mobile Link サーバは、エントリを ml_passthrough_status テーブルから自動的には削除しません。このテーブルからの削除は手動で実行してください。

統合データベースのタイプが DB2 メインフレームの場合、このテーブルは ml_pt_status という名前になります。「IBM DB2 メインフレームのシステム・テーブル名の変換」 736 ページを参照してください。

制約

PRIMARY KEY(status_id)

FOREIGN KEY(remote_id) REFERENCES ml_database(remote_id)

FOREIGN KEY(script_id) REFERENCES ml_passthrough_script(script_id)

参照

- 「upload_insert テーブル・イベント」 530 ページ

ml_property

このテーブルには一部の Mobile Link プロパティが格納されます。

カラム	説明
component_name	VARCHAR(128)。ユーザ定義のプロパティの場合は **ScriptVersion** または **SIS** です。
property_set_name	VARCHAR(128)。component_name が **ScriptVersion** である場合、スクリプト・バージョンの名前です。component_name が **SIS** である場合、プロパティを設定している Notifier、ゲートウェイ、または Carrier の名前です。
property_name	VARCHAR(128)。プロパティ名。component_name が **ScriptVersion** である場合、ユーザ定義のプロパティです。component_name が **SIS** である場合、Notifier、ゲートウェイ、または Carrier のプロパティです。「サーバ起動同期の Mobile Link サーバ設定」 『Mobile Link - サーバ起動同期』を参照してください。
property_value	TEXT。プロパティの値。

備考

このテーブルには、名前と値の組み合わせが格納されます。このテーブルの一部のプロパティは、Mobile Link によって内部的に使用されます。さらに、ストアド・プロシージャ ml_add_property を使用して、このテーブルにローを追加または削除できます。

component_name **ScriptVersion** を使用すると、スクリプト・バージョンごとに情報を格納し、その情報に Java や .NET のスクリプト論理でアクセスできます。

制約

PRIMARY KEY(component_name, property_set_name, property_name)

参照

- 「ml_add_property システム・プロシージャ」 713 ページ

ml_qa_clients

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。SQL Anywhere と Oracle 統合データベースだけに存在するグローバル・テンポラリ・テーブルです。

警告
このテーブルは変更しないでください。

カラム	説明
client	VARCHAR(128)。アップロードされたメッセージの宛先のクライアント。

ml_qa_delivery

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
seqno	BIGINT。メッセージの順序を指定するために使用されます。正確なキューイングに必要です。
address	VARCHAR(255)。ターゲット受信者のアドレス。
clientaddress	VARCHAR(128)。アドレスのクライアント部分。
client	VARCHAR(128)。現在のクライアント・ステータスの対象となっているクライアント。
originator	VARCHAR(128)。送信元のクライアントの名前。
priority	INTEGER。0 ～ 9 の数字。優先度が高い数字が付いているメッセージが配信されてから、優先度の低い数字が付いているメッセージが配信されます。デフォルトは 4 です。
expires	TIMESTAMP。有効期限の日時で、これを過ぎるとメッセージが配信されなくなります。
kind	INTEGER。メッセージがバイナリ (**1**) であるか、テキスト (**2**) であるかを示します。
contentsize	BIGINT。メッセージのサイズ。バイナリ・メッセージの場合はバイト数です。テキスト・メッセージの場合は文字数です。
status	INTEGER。メッセージのステータス。1 (保留)、10 (受信中)、30 (有効期限切れ)、40 (キャンセル済み)、50 (受信不可)、60 (受信済み) のいずれかです。
statustime	TIMESTAMP。このステータスになった日時。日時は、このステータスになったクライアントのローカル時間です。
syncstatus	INTEGER。このメッセージに関する、クライアントとサーバ間の同期のステータス。0 (非同期)、1 (同期)、2 (メッセージが同期できない)、3 (同期中) のいずれかです。

カラム	説明
receiverid	VARCHAR(128)。メッセージの受信者を識別する、受信者によって設定される識別子 (存在する場合)。

制約

PRIMARY KEY(msgid, address)

FOREIGN KEY (msgid) REFERENCES ml_qa_repository(msgid)

ml_qa_delivery_archive

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
seqno	BIGINT。メッセージの順序を指定するために使用されます。正確なキューイングに必要です。
address	VARCHAR(255)。ターゲット受信者のアドレス。
clientaddress	VARCHAR(128)。アドレスのクライアント部分。
client	VARCHAR(128)。現在のクライアント・ステータスの対象となっているクライアント。
originator	VARCHAR(128)。送信元のクライアントの名前。
priority	INTEGER。0 ～ 9 の数字。優先度が高い数字が付いているメッセージが配信されてから、優先度の低い数字が付いているメッセージが配信されます。デフォルトは 4 です。
expires	TIMESTAMP。有効期限の日時で、これを過ぎるとメッセージが配信されなくなります。
kind	INTEGER。メッセージがバイナリ (**1**) であるか、テキスト (**2**) であるかを示します。
contentsize	BIGINT。メッセージのサイズ。バイナリ・メッセージの場合はバイト数です。テキスト・メッセージの場合は文字数です。
status	INTEGER。メッセージのステータス。1 (保留)、10 (受信中)、30 (有効期限切れ)、40 (キャンセル済み)、50 (受信不可)、60 (受信済み) のいずれかです。
statustime	TIMESTAMP。このステータスになった日時。日時は、このステータスになったクライアントのローカル時間です。
syncstatus	INTEGER。このメッセージに関する、クライアントとサーバ間の同期のステータス。0 (非同期)、1 (同期)、2 (メッセージが同期できない)、3 (同期中) のいずれかです。

カラム	説明
receiverid	VARCHAR(128)。メッセージの受信者を識別する、受信者によって設定される識別子 (存在する場合)。

制約

PRIMARY KEY(msgid, address)

FOREIGN KEY (msgid) REFERENCES ml_qa_repository(msgid)

ml_qa_global_props

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。このテーブルには、グローバルな *name-value* の組み合わせが含まれています。この組み合わせは、転送ルールで使用されます。

警告
このテーブルは変更しないでください。

カラム	説明
client	VARCHAR(128)。プロパティに関連付けられているクライアント。クライアント値が 'ianywhere.server.defaultClient' の場合は、すべてのクライアントにとってグローバルなプロパティを表します。
name	VARCHAR(255)。プロパティ名。
modifiers	INTEGER。プロパティを詳しく説明するためのビット・フィールド。現在は、先頭ビットのみが、同期できないプロパティを指定するために使用されています。他のすべてのビット・フィールドは、今後の使用のために確保されています。
value	LONG VARCHAR。プロパティの値。
last_modified	TIMESTAMP。値が最後に変更された日時。プロパティをクライアントといつ同期する必要があるかを指定するために必要です。

制約

PRIMARY KEY (client, name)

ml_qa_notifications

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、同期の開始を通知する QAnywhere クライアントを判別するために Notifier によって使用されます。

警告
このテーブルは変更しないでください。

カラム	説明
user_id	INTEGER。
name	VARCHAR(128)。クライアント・メッセージ・ストアをユニークに識別する QAnywhere クライアント名。

制約

PRIMARY KEY(name)

ml_qa_repository

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。このテーブルには、メッセージとそのプロパティが格納されます。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
props	LONG BINARY。メッセージのプロパティのエンコード。
content	LONG BINARY。メッセージの内容。テキスト・メッセージは、UTF-8 でエンコードされます。

制約

PRIMARY KEY(msgid)

ml_qa_repository_archive

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。このテーブルには、メッセージとそのプロパティがアーカイブされます。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
props	LONG BINARY。メッセージのプロパティのエンコード。
content	LONG BINARY。メッセージの内容。テキスト・メッセージは、UTF-8 でエンコードされます。

制約

PRIMARY KEY(msgid)

ml_qa_repository_props

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、これは ml_qa_repository テーブルの props カラムを拡張したものです。必要に応じて、プロパティは転送ルールのエンジンによってのみ拡張されます。関連付けられたルールがない場合は、プロパティは拡張されません。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
name	VARCHAR(128)。プロパティ名。プロパティ名が Unicode で指定されている場合は、データベースのネイティブの文字セットに変換されます。
value	LONG VARCHAR。プロパティの値。

制約

PRIMARY KEY(msgid, name)

FOREIGN KEY (msgid) REFERENCES ml_qa_repository(msgid)

ml_qa_repository_props_archive

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。また、これは ml_qa_repository テーブルの props カラムを拡張したものです。必要に応じて、プロパティは転送ルールのエンジンによってのみ拡張されます。関連付けられたルールがない場合は、プロパティは拡張されません。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
name	VARCHAR(128)。プロパティ名。プロパティ名が Unicode で指定されている場合は、データベースのネイティブの文字セットに変換されます。
value	LONG VARCHAR。プロパティの値。

制約

PRIMARY KEY(msgid, name)

FOREIGN KEY (msgid) REFERENCES ml_qa_repository(msgid)

ml_qa_repository_staging

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。これには、SQL Anywhere バージョン 9.0.1 を使用している QAnywhere クライアントに送信されるメッセージが格納されます。

警告
このテーブルは変更しないでください。

カラム	説明
seqno	BIGINT。メッセージ全体の順序を指定するために使用されます。正確なキューイングに必要です。
msgid	VARCHAR(255)。グローバルなユニークなメッセージ識別子。
destination	VARCHAR(128)。メッセージのアドレス。
originator	VARCHAR(128)。送信元の Mobile Link ユーザの名前。
status	VARCHAR(128)。メッセージのステータス。**pending**、**receiving**、**received**、**unreceivable**、**expired**、**cancelled** のいずれかです。デフォルトは **pending** です。
statustime	TIMESTAMP。ステータスが最後に変更された日時。
expires	TIMESTAMP。有効期限の日時で、これを過ぎるとメッセージが配信されなくなります。
priority	INTEGER。0 ～ 9 の数字。常に、大きい数字が付いているメッセージが配信されてから、小さい数字が付いているメッセージが配信されます。デフォルトは 4 です。
props	LONG BINARY。メッセージのプロパティのエンコード。
kind	INTEGER。メッセージがバイナリ (**1**) であるか、テキスト (**2**) であるかを示します。
content	LONG BINARY。メッセージの内容。テキスト・メッセージは、UTF-8 でエンコードされます。
contentsize	BIGINT。メッセージのサイズ。バイナリ・メッセージの場合はバイト数です。テキスト・メッセージの場合は文字数です。
mluser	VARCHAR(128)。Mobile Link ユーザ名。これによって、リモート・データベースがユニークに識別されます。

制約

PRIMARY KEY(msgid)

ml_qa_status_history

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。このテーブルには、メッセージ・ステータスの変更の履歴が含まれています。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
address	VARCHAR(255)。ターゲット受信者のアドレス。
status	INTEGER。メッセージのステータス。1 (保留)、10 (受信中)、30 (有効期限切れ)、40 (キャンセル済み)、50 (受信不可)、60 (受信済み) のいずれかです。
statustime	TIMESTAMP。このステータスになった日時。日時は、このステータスになったクライアントのローカル時間です。
servertime	TIMESTAMP。サーバがステータスの変更を受信した日時。
details	VARCHAR(1000)。ステータスの変更の詳細 (存在する場合)。
syncstatus	INTEGER。このメッセージに関する、クライアントとサーバ間の同期のステータス。**0** (非同期)、1 (同期)、**2** (メッセージが同期できない)、**3** (同期中) のいずれかです。

制約

PRIMARY KEY(msgid)

FOREIGN KEY (msgid) REFERENCES ml_qa_repository(msgid)

ml_qa_status_history_archive

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。このテーブルには、メッセージ・ステータスの変更の履歴が含まれています。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
address	VARCHAR(255)。ターゲット受信者のアドレス。
status	INTEGER。メッセージのステータス。1 (保留)、10 (受信中)、30 (有効期限切れ)、40 (キャンセル済み)、50 (受信不可)、60 (受信済み) のいずれかです。
statustime	TIMESTAMP。このステータスになった日時。日時は、このステータスになったクライアントのローカル時間です。
servertime	TIMESTAMP。サーバがステータスの変更を受信した日時。
details	VARCHAR(1000)。ステータスの変更の詳細 (存在する場合)。
syncstatus	INTEGER。このメッセージに関する、クライアントとサーバ間の同期のステータス。**0** (非同期)、1 (同期)、**2** (メッセージが同期できない)、**3** (同期中) のいずれかです。

制約

PRIMARY KEY(msgid)

FOREIGN KEY (msgid) REFERENCES ml_qa_repository(msgid)

ml_qa_status_staging

このテーブルは、QAnywhere アプリケーション用にのみ使用されます。これは、送信元のクライアントで SQL Anywhere バージョン 9.0.1 が使用されていた場合、送信元のクライアントとの同期ステータスが変化したときに使用される中間テーブルです。

警告
このテーブルは変更しないでください。

カラム	説明
msgid	VARCHAR(128)。グローバルなユニークなメッセージ識別子。
status	VARCHAR(255)。メッセージのステータス。**pending**、**receiving**、**received**、**unreceivable**、**expired**、**cancelled** のいずれかです。デフォルトは **pending** です。
statustime	TIMESTAMP。ステータスが最後に変更された日時。
mluser	VARCHAR(128)。Mobile Link ユーザ名。これによって、リモート・データベースがユニークに識別されます。

制約

PRIMARY KEY(msgid)

ml_script

このテーブルには、すべてのスクリプトの内容が格納されます。

カラム	説明
script_id	INTEGER。スクリプトを識別するユニークな整数。
script	TEXT。スクリプトのテキスト。
script_language	VARCHAR(128)。スクリプトに使用されるスクリプト言語。スクリプト言語は、**sql**、**java**、または **dnet** が使用できます。
checksum	VARCHAR(64)。このカラムは内部で使用されます。

制約

PRIMARY KEY(script_id)

ml_script_version

このテーブルには、各スクリプト・バージョンに関連付けられたスクリプトの名前と説明が格納されます。

カラム	説明
version_id	INTEGER。バージョンを識別するユニークな整数。
name	VARCHAR(128)。スクリプト・バージョンの名前。
description	TEXT。バージョンを説明する記述。この記述は Mobile Link が使用するものではありませんが、アプリケーション固有のコメントとして役立ちます。たとえば、指定したスクリプト・バージョンの目的を説明します。

制約

PRIMARY KEY(version_id)

ml_scripts_modified

このテーブルには、スクリプト・テーブルが最後に変更された時刻が格納されます。Mobile Link サーバは、新しいスクリプトをロードするかどうか決定するためにこのテーブルを確認します。

カラム	説明
last_modified	DATETIME。ml_script、ml_table_script、または ml_connection_script システム・テーブルが最後に変更された時刻。

備考

統合データベースのタイプが DB2 メインフレームの場合、このテーブルは ml_script_modified という名前になります。「IBM DB2 メインフレームのシステム・テーブル名の変換」 736 ページを参照してください。

制約

PRIMARY KEY(last_modified)

ml_server

サーバ・ファームで実行されている各 Mobile Link サーバの情報を格納します。

カラム	説明
server_id	INTEGER。Mobile Link サーバ名のサーバ ID を識別するユニークな整数。この値は内部で使用されます。
name	VARCHAR(128)。サーバ・ファームでアクティブに実行されている Mobile Link サーバの名前。この値は、サーバ・ファーム内でユニークにする必要があります。
version	VARCHAR(10)。Mobile Link サーバの現在のバージョン。
connection_info	VARCHAR(2048)。Mobile Link サーバの接続情報。
instance_key	VARCHAR(32)。割り当てられたインスタンス・キー。この値は内部で使用されます。
start_time	TIMESTAMP。Mobile Link サーバの起動時刻。
liveness	TIMESTAMP。クライアントがサーバに最後に応答した時刻。

備考

サーバ・ファームが実行されていない場合、このテーブルにはデータはありません。

制約

PRIMARY KEY(server_id)

ml_sis_sync_state

このテーブルは、サーバ起動同期の要求カーソルを生成するために Sybase Central によって使用されます。

警告
このテーブルは変更しないでください。

カラム	説明
remote_id	VARCHAR(128)。データベースを識別するユニークなリモート ID。
subscription_id	VARCHAR(128)。subscription_id は、リモート・データベースが生成する番号です。SQL Anywhere クライアントの場合、この値は SYS.ISYSSYNC システム・テーブルの sync_id と同じです。
publication_name	VARCHAR(128)。サブスクリプションによってサブスクライブされるパブリケーションのユーザ定義の名前。同期ごとにクライアントは各 subscription_id のパブリケーション名を送信します。 バージョン 10.0.0 より前の Ultra Light クライアントの場合、これは常に <unknown> です。Ultra Light バージョン 10 以降の場合は、パブリケーションか、文字列 ul_no_pub (パブリケーションがない場合) です。
user_name	VARCHAR(128)。Mobile Link ユーザ名。
last_upload	TIMESTAMP。指定したリモート ID と subscription_id のアップロードが、統合データベースに最後に適用された時刻。デフォルトは January 1, 1900, 00:00:00 です。
last_download	TIMESTAMP。指定したユーザと subscription_id のダウンロードが、統合データベースに最後に適用された時刻。デフォルトは January 1, 1900, 00:00:00 です。

制約

PRIMARY KEY(remote_id, subscription_id)

ml_subscription

このテーブルには、各リモートのステータス情報が格納されます。

カラム	説明
rid	INTEGER。リモート ID を識別するユニークな整数。この値は内部で使用されます。
subscription_id	VARCHAR(128)。subscription_id は、リモート・データベースが生成する番号です。SQL Anywhere クライアントの場合、この値は SYS.ISYSSYNC システム・テーブルの sync_id と同じです。 Ultra Light クライアントはサブスクリプションを使用しません。したがって、Ultra Light クライアントの場合、この値はバージョン 10.0.0 以降では Ultra Light パブリケーション ID であり、バージョン 8 と 9 では <unknown> です。
user_id	INTEGER。指定された rid と subscription_id に対して最後の同期を実行したユーザ。user_id カラムを使用して、成功した最後の同期を実行した Mobile Link ユーザを検索できます。
progress	NUMERIC(20,0)。オフセット、ステータス、シーケンス番号、進捗状況のカウンタとも呼ばれる同期の進捗状況。
publication_name	VARCHAR(128)。サブスクリプションによってサブスクライブされるパブリケーションのユーザ定義の名前。同期ごとにクライアントは各 subscription_id のパブリケーション名を送信します。 バージョン 10.0.0 より前の Ultra Light クライアントの場合、これは常に <unknown> です。Ultra Light バージョン 10 以降の場合は、パブリケーションか、文字列 ul_no_pub (パブリケーションがない場合) です。
last_upload_time	TIMESTAMP。指定したリモート ID と subscription_id のアップロードが、統合データベースに最後に適用された時刻。デフォルトは January 1, 1900, 00:00:00 です。
last_download_time	TIMESTAMP。指定したユーザと subscription_id のダウンロードが、統合データベースに最後に適用された時刻。デフォルトは January 1, 1900, 00:00:00 です。「ダウンロード・タイムスタンプの生成および使用方法」 135 ページを参照してください。

備考

SQL Anywhere クライアントでは、"progress" は、リモート・データベースのトランザクション・ログの位置を示します。これは、サブスクリプションに対するすべてのコミット済み操作がデータベースからアップロードされた点を示します。dbmlsync ユーティリティは、オフセットを使

用してどのデータをアップロードするか決定します。SQL Anywhere リモート・データベースでは、オフセットは SYS.SYSSYNC システム・テーブルの progress カラムに格納されます。

次の項を参照してください。

- 「SYSSYNC システム・ビュー」『SQL Anywhere サーバ - SQL リファレンス』
- 「進行オフセット」『Mobile Link - クライアント管理』

Ultra Light クライアントでは、"progress" は、指定されたパブリケーションの同期シーケンス番号または進行状況のカウンタです。このカウンタは、どのローが同期されたかを示します。カウンタは、パブリケーションが同期するたびに増分されます。この数字は Ultra Light データベースの内部で使用され、アクセスすることはできません。

「進行状況のカウンタ」『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。

制約

PRIMARY KEY(rid, subscription_id)

FOREIGN KEY(rid) REFERENCES ml_database(rid)

FOREIGN KEY(user_id) REFERENCES ml_user(user_id)

参照

- 「リモート ID」『Mobile Link - クライアント管理』

ml_table

このテーブルにはリモート・テーブルの名前が格納されます。このリストには、同期テーブルとしてマーク付けされたすべてのテーブルが含まれます。

カラム	説明
table_id	INTEGER。テーブルを識別するユニークな整数。
name	VARCHAR(128)。テーブルに指定された名前。

制約

PRIMARY KEY (table_id)

ml_table_script

指定されたスクリプト・バージョンにおいて、このテーブルは、テーブル・スクリプトを指定されたテーブルとイベントに関連付けます。

カラム	説明
version_id	INTEGER。スクリプト・バージョンを識別する番号。
table_id	INTEGER。テーブルを識別する番号。
event	VARCHAR(128)。イベント名。
script_id	INTEGER。スクリプトを識別する番号。スクリプトは ml_script テーブルに格納されます。

備考

ml_table_script Mobile Link システム・テーブルの内容を簡単に表示できるシステム・ビュー (ml_table_scripts) があります。

制約

FOREIGN KEY(version_id) REFERENCES ml_script_version(version_id)

FOREIGN KEY(table_id) REFERENCES ml_table(table_id)

FOREIGN KEY(script_id) REFERENCES ml_script(script_id)

ml_user

登録済みユーザと、そのハッシュされたパスワードを格納します。

カラム	説明
user_id	INTEGER。ユーザを識別するユニークな整数。この値は内部で使用されます。
name	VARCHAR(128)。登録されたユーザ名。
hashed_password	BINARY(32)。難読化した形式のパスワード。この値は NULL にすることができますが、パスワードを指定することをおすすめします。

備考

このテーブルには、Mobile Link サーバが認識している、すべての登録済みユーザが格納されます。ユーザ名は同期ごとにクライアントによって送信されます。クライアントは、認証のためにユーザのパスワードを送信することもできます。

Mobile Link サーバは独自のアルゴリズムを使用して、ユーザ・パスワードをハッシュします。

ユーザ名を NULL でないパスワードとともにこのテーブルに直接挿入しないでください。ユーザ名は、mluser ユーティリティを使用して追加できます。

制約

PRIMARY KEY (user_id)

参照

- 「Mobile Link ユーザ認証ユーティリティ (mluser)」 729 ページ

リモート・データベースと統合データベース間での Mobile Link データ・マッピング

目次

Adaptive Server Enterprise データのマッピング

Adaptive Server Enterprise の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモート・データ型がどのように Adaptive Server Enterprise の統合データ型にマッピングされるのかを示します。たとえば、リモート・データベースの FLOAT 型のカラムは、統合データベースでは REAL 型である必要があります。

最大カラム長 (MCL) は、Adaptive Server Enterprise のページ・サイズによって異なります。ページ・サイズが 2K の場合、MCL は 1954 になります。ページ・サイズが 4K の場合、MCL は 4002 になります。MCL の詳細については、Adaptive Server Enterprise のマニュアルを参照してください。

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
BIGINT	NUMERIC(20)[1] または BIGINT[2]	
BIT	BIT	
BINARY(*n*=<MCL)	BINARY(*n*)	
BINARY(*n*>MCL)	IMAGE	
CHAR(*n*=<MCL)	VARCHAR(*n*)	
CHAR(*n*>MCL)	TEXT	ダウンロード時に、値が長すぎないようにします。
DATE	DATE[3] または DATETIME[4]	Adaptive Server Enterprise の DATETIME では、年は 1753 ～ 9999 の範囲内である必要があります。 SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
DATETIME	DATETIME	Adaptive Server Enterprise の DATETIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。 ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。 DATETIME をプライマリ・キーに使用すると、競合を解決できないことがあります。DATETIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。また、年は 1753 ～ 9999 の範囲内である必要があります。
DECIMAL(*p*<39, *s*)	DECIMAL(*p*,*s*)	Adaptive Server Enterprise の NUMERIC の精度は 1 ～ 38 桁 (*p*<39) です。
DECIMAL(*p*>=39,*s*)		Adaptive Server Enterprise のデータ型に対応するデータ型がありません。
DOUBLE	DOUBLE PRECISION	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	IMAGE	
INTEGER	INTEGER	
LONG BINARY	IMAGE	
LONG NVARCHAR	UNITEXT	
LONG VARBIT	TEXT	
LONG VARCHAR	TEXT	

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
MONEY	MONEY	
NCHAR(*c*=<MCL)	UNIVARCHAR(*c*/2)	
NCHAR(*c*>MCL)	UNITEXT	ダウンロード時に、値が長すぎないようにします。
NTEXT	UNITEXT	
NUMERIC(*p*<39,*s*)	NUMERIC(*p*,*s*)	Adaptive Server Enterprise の 10 進数の精度は 1 ～ 38 桁 (*p*<39) です。
NUMERIC(*p*>=39,*s*)		
NVARCHAR(*c*=<MCL)	UNIVARCHAR(*c*/2)	
NVARCHAR(*c*>MCL)	UNITEXT	ダウンロード時に、値が長すぎないようにします。
REAL	REAL	
SMALLDATETIME	DATETIME[4]	SQL Anywhere と Ultra Light の SMALLDATETIME は TIMESTAMP として実装されます。 Adaptive Server Enterprise の DATETIME は分の精度です。29.998 秒以下の値は直近の分に切り捨てられます。29.999 秒以上の値は直近の分に切り上げられます。SQL Anywhere と Ultra Light の SMALLDATETIME はミリ秒の精度です。正常に同期させるには、SQL Anywhere または Ultra Light の SMALLDATETIME の丸め単位を分にしてください。また、年は 1753 ～ 9999 の範囲内である必要があります。
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	TEXT	

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
TIME	TIME[3] または DATETIME[4]	Adaptive Server Enterprise の TIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。TIME をプライマリ・キーに使用すると、競合を解決できないことがあります。TIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。
TIMESTAMP	DATETIME	Adaptive Server Enterprise の DATETIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。 ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。 DATETIME をプライマリ・キーに使用すると、競合を解決できないことがあります。DATETIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。また、年は 1753 ～ 9999 の範囲内である必要があります。
TINYINT	TINYINT	
UNIQUEIDENTIFIER	CHAR(36)	

SQL Anywhere または Ultra Light のデータ型	Adaptive Server Enterprise データ型	注意
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR は使用しないでください。代わりに UNIQUEIDENTIFIER を使用してください。
UNSIGNED BIGINT	NUMERIC(20)[1] または UNSIGNED BIGINT[2]	
UNSIGNED INTEGER	UNSIGNED INT	
UNSIGNED SMALLINT	UNSIGNED SMALLINT	
UNSIGNED TINYINT	TINYINT	
VARBINARY(*n*=<MCL)	VARBINARY	
VARBINARY(*n*>MCL)	IMAGE	
VARBIT(*n*=<MCL)	VARCHAR(*n*)	
VARBIT(*n*>MCL)	TEXT	
VARCHAR(*n*=<MCL)	VARCHAR(*n*)	
VARCHAR(*n*>MCL)	TEXT	
XML	TEXT	

[1] バージョン 15.0 より前の Adaptive Server Enterprise のみに該当します。

[2] バージョン 15.0 以降の Adaptive Server Enterprise のみに該当します。

[3] バージョン 12.5.1 以降の Adaptive Server Enterprise のみに該当します。

[4] バージョン 12.5.1 より前の Adaptive Server Enterprise のみに該当します。

SQL Anywhere または Ultra Light のリモート・データ型へのマッピング

次の表は、Adaptive Server Enterprise の統合データ型がどのように SQL Anywhere および Ultra Light のリモート・データ型にマッピングされるのかを示します。たとえば、統合データベースの DOUBLE PRECISION 型のカラムは、リモート・データベースでは DOUBLE 型である必要があります。

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	注意
BIGINT[1]	BIGINT	
BINARY(*n*)	BINARY(*n*)	
BIT	BIT	
CHAR(*n*)	VARCHAR(*n*)	SQL Anywhere の CHAR/NCHAR は、Adaptive Server Enterprise の CHAR/NCHAR と同等ではありません。SQL Anywhere の CHAR/NCHAR に対応するのは、VARCHAR/NVARCHAR です。同期される統合データベースのカラムでは、CHAR/NCHAR を使用しないでください。SQL Anywhere 以外の CHAR/NCHAR を使用する必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。
DATE	DATE	SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DATETIME	DATETIME	Adaptive Server Enterprise の DATETIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。 ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。競合は解決できない場合があります。DATETIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。また、年は 1753 ～ 9999 の範囲内である必要があります。
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	注意
DOUBLE PRECISION	DOUBLE	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	LONG BINARY	
INT	INT	
MONEY	MONEY	
NCHAR(*n*)	VARCHAR(*n*)	Adaptive Server Enterprise の NCHAR と NVARCHAR は、SQL Anywhere の NCHAR と NVARCHAR とは異なり、マルチバイトの各国の文字列を格納します。マルチバイト環境では、SQL Anywhere または Ultra Light の VARCHAR を使用してください。
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
NVARCHAR(*n*)	VARCHAR(*n*)	Adaptive Server Enterprise の NCHAR と NVARCHAR は、SQL Anywhere の NCHAR と NVARCHAR とは異なり、マルチバイトの各国の文字列を格納します。マルチバイト環境では、SQL Anywhere または Ultra Light の VARCHAR を使用してください。
REAL	REAL	

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	注意
SMALLDATETIME	SMALLDATETIME	SQL Anywhere と Ultra Light の SMALLDATETIME は TIMESTAMP として実装されます。 Adaptive Server Enterprise の SMALLDATETIME は分の精度です。29.998 秒以下の値は直近の分に切り捨てられます。29.999 秒以上の値は直近の分に切り上げられます。SQL Anywhere と Ultra Light の SMALLDATETIME はミリ秒の精度です。正常に同期させるには、SQL Anywhere または Ultra Light の SMALLDATETIME の丸め単位を分にしてください。また、年は 1900 ～ 2078 の範囲内である必要があります。
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	LONG VARCHAR	
TIME	TIME	Adaptive Server Enterprise の TIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。 ダウンロード時には、SQL Anywhere は Adaptive Server Enterprise からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。競合は解決できない場合があります。TIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。

Adaptive Server Enterprise データ型	SQL Anywhere または Ultra Light のデータ型	注意
TIMESTAMP	VARBINARY(8)	Adaptive Server Enterprise 内では、TIMESTAMP はローが変更されるたびに増分されるバイナリ・カウンタです。各テーブルに存在できる TIMESTAMP カラムは 1 つのみであるため、TIMESTAMP カラムを同期する意味はありません。同期に含める必要がある場合は、SQL Anywhere または Ultra Light の VARBINARY(8) データ型にマッピングします。 この TIMESTAMP カラムはサーバによって維持されるため、明示的に挿入および更新することはできません。このようなカラムがあるテーブルのアップロード・スクリプトを実装する場合は、このことを念頭に置いてください。
TINYINT	TINYINT	
UNSIGNED BIGINT[1]	UNSIGNED BIGINT	
UNSIGNED INT[1]	UNSIGNED INT	
UNSIGNED SMALLINT[1]	UNSIGNED SMALLINT	
VARBINARY(*n*)	VARBINARY(*n*)	
VARCHAR(*n*)	VARCHAR(*n*)	
UNICHAR(*n*)	NVARCHAR(*n*)	Ultra Light では使用できません。
UNITEXT[1]	LONG NVARCHAR	Ultra Light では使用できません。
UNIVARCHAR(*n*)	NVARCHAR(*n*)	Ultra Light では使用できません。

[1] バージョン 15.0 より前の Adaptive Server Enterprise のみに該当します。

IBM DB2 LUW データのマッピング

IBM DB2 LUW の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモート・データ型がどのように IBM DB2 LUW の統合データ型にマッピングされるのかを示します。たとえば、リモート・データベースの BIT 型のカラムは、統合データベースでは SMALLINT 型である必要があります。

DB2 テーブルを作成する場合は、DB2 のページ・サイズに注意する必要があります。DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、DB2 のマニュアルを参照してください。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	注意
BIGINT	BIGINT	
BINARY(*n*<MRL)	VARCHAR(*n*) FOR BIT DATA	
BINARY(*n*>=MRL)	BLOB(*n*)	
BIT	SMALLINT	
CHAR(*n*<MRL)	VARCHAR(*n*)	
CHAR(*n*>=MRL)	CLOB(*n*)	DB2 の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
DATE	DATE	SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DATETIME	TIMESTAMP	
DECIMAL(*p*<32,*s*)	DECIMAL(*p*,*s*)	SQL Anywhere の DECIMAL の精度は 1 ～ 127 です。DB2 DECIMAL の最大精度は 31 です。
DECIMAL(*p*>=32,*s*)		SQL Anywhere の DECIMAL で精度が 31 より大きいデータは、DB2 に同期できません。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	注意
DOUBLE	DOUBLE	DOUBLE は、丸め誤差が出る可能性がある概数値データ型です。種類の異なるコンピュータを使用すると、DOUBLE の基本となる記憶領域が異なることが多く、したがって、丸めの結果も異なります。プライマリ・キーは等しいかどうかの確認に使用されるので、プライマリ・キーで DOUBLE を使用することはおすすめできません。これは特に同期環境で言えます。特に、統合データベースはリモート・データベースとは異なるハードウェアで実行されることが多いからです。
FLOAT(1-24)	REAL	FLOAT はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。
FLOAT(25-53)	DOUBLE	FLOAT はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。
IMAGE	BLOB(*n*)	
INTEGER	INTEGER	
LONG BINARY	BLOB(*n*)	

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	注意
LONG NVARCHAR	CLOB(*n*)	DB2 のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合は、SQL Anywhere の LONG NVARCHAR を DB2 の CLOB に同期できます。Ultra Light には LONG NVARCHAR はありません。
LONG VARBIT	CLOB(*n*)	
LONG VARCHAR	CLOB(*n*)	
MONEY	DECIMAL(19,4)	
NCHAR(*c*)	VARCHAR(*n*) または CLOB(*n*)	DB2 のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合は、NCHAR を DB2 の VARCHAR または CLOB に同期できます。SQL Anywhere の NCHAR のサイズは文字単位で、DB2 の VARCHAR のサイズはバイト単位です。VARCHAR にマッピングする場合は、NCHAR のバイト数の合計が MRL より大きくならないようにしてください。そのようにできない場合、NCHAR は CLOB にマッピングします。NCHAR(*c*) のバイト数の計算は簡単ではありませんが、おおよそ *c*=*n*/4 です。一般的には、*c* が MRL/4 より小さい場合は VARCHAR(*n*) にマッピングし、*c* が MRL/4 以上の場合は CLOB(*n*) にマッピングします。
NUMERIC(*p*<32,*s*)	NUMERIC(*p*,*s*)	
NUMERIC(*p*>=32,*s*)		DB2 のデータ型に対応するデータ型がありません。
NTEXT	CLOB(*n*)	DB2 のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、NTEXT を DB2 の CLOB に同期できます。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	注意
NVARCHAR(*c*)	VARCHAR(*n*) または CLOB(*n*)	DB2 のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、NVARCHAR を DB2 の VARCHAR または CLOB に同期できます。SQL Anywhere の NVARCHAR のサイズは文字単位で、DB2 の VARCHAR のサイズはバイト単位です。VARCHAR にマッピングする場合は、NVARCHAR のバイト数の合計が MRL より大きくならないようにしてください。そのようにできない場合、NVARCHAR は CLOB にマッピングします。NVARCHAR(*c*) のバイト数の計算は簡単ではありませんが、おおよそ *c*=*n*/4 です。一般的には、*c* が MRL/4 より小さい場合は VARCHAR(*n*) にマッピングし、*c* が MRL/4 以上の場合は CLOB(*n*) にマッピングします。
REAL	REAL	REAL はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。
SMALLDATETIME	TIMESTAMP	
SMALLINT	SMALLINT	
SMALLMONEY	DECIMAL(10,4)	
TEXT	CLOB(*n*)	
TIME	TIMESTAMP または TIME	小数点以下の秒がある SQL Anywhere と Ultra Light の TIME 値には、DB2 の TIMESTAMP が必要です。小数点以下の秒が常に 0 の SQL Anywhere と Ultra Light の TIME 値には、DB2 の TIME を使用できます。
TIMESTAMP	TIMESTAMP	

SQL Anywhere または Ultra Light のデータ型	IBM DB2 LUW のデータ型	注意
TINYINT	SMALLINT	ダウンロードの場合、DB2 の値は負でない必要があります。
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR の DB2 での使用はおすすめしません。代わりに UNIQUEIDENTIFIER を使用してください。
UNSIGNED BIGINT	DECIMAL(20)	ダウンロードの場合、DB2 の値は負でない必要があります。
UNSIGNED INTEGER	DECIMAL(11)	ダウンロードの場合、DB2 の値は負でない必要があります。
UNSIGNED SMALLINT	DECIMAL(5)	ダウンロードの場合、DB2 の値は負でない必要があります。
UNSIGNED TINYINT	SMALLINT	ダウンロードの場合、DB2 の値は負でない必要があります。
VARBINARY(*n*<MRL)	VARCHAR(*n*) FOR BIT DATA	
VARBINARY(*n*>=MRL)	BLOB(*n*)	
VARBIT(*n*<MRL)	VARCHAR(*n*)	
VARBIT(*n*>=MRL)	CLOB(*n*)	
VARCHAR(*n*<MRL)	VARCHAR(*n*)	
VARCHAR(*n*>=MRL)	CLOB(*n*)	DB2 の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
XML	CLOB(*n*)	

SQL Anywhere または Ultra Light のリモート・データ型へのマッピング

次の表は、IBM DB2 LUW の統合データ型がどのように SQL Anywhere および Ultra Light のリモート・データ型にマッピングされるのかを示します。たとえば、統合データベースの INT 型のカラムは、リモート・データベースでは INTEGER 型である必要があります。

DB2 テーブルを作成する場合は、DB2 のページ・サイズに注意する必要があります。DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、DB2 のマニュアルを参照してください。

IBM DB2 LUW のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
BLOB	LONG BINARY	
BIGINT	BIGINT	
CHAR(*n*)	VARCHAR(*n*)	SQL Anywhere には DB2 の CHAR に対応するデータ型がありません。同期される統合データベースのカラムでは、CHAR を使用しないでください。DB2 の CHAR カラムを同期させる必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。
CHAR(*n*) FOR BIT DATA	BINARY(*n*)	
CLOB(*n*)	LONG VARCHAR	
DATE	DATE	SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DBCLOB(*n*)	LONG VARCHAR	DBCLOB(*n*) データ型は 2 バイト文字用にのみ使用します。SQL Anywhere のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、DBCLOB(*n*) は CLOB と同等です。
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	

IBM DB2 LUW のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
DOUBLE	DOUBLE	DOUBLE は、丸め誤差が出る可能性がある概数値データ型です。種類の異なるコンピュータを使用すると、DOUBLE の基本となる記憶領域が異なることが多く、したがって、丸めの結果も異なります。プライマリ・キーは等しいかどうかの確認に使用されるので、プライマリ・キーで DOUBLE を使用することはおすすめできません。これは特に同期環境で言えます。特に、統合データベースはリモート・データベースとは異なるハードウェアで実行されることが多いからです。
FLOAT	DOUBLE	FLOAT はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。
GRAPHIC(*n*)	VARCHAR(2*n*)	DB2 の GRAPHIC にはブランクが埋め込まれますが、SQL Anywhere の CHAR には埋め込まれません。このデータ型は使用しないことをおすすめします。 GRAPHIC データ型は 2 バイト文字用にのみ使用します。SQL Anywhere のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、GRAPHIC は CHAR と同等です。
INT	INTEGER	
LONG VARCHAR	VARCHAR(32700)	
LONG VARCHAR FOR BIT DATA	VARBINARY(32700)	

IBM DB2 LUW のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
LONG VARGRAPHIC(*n*)	VARCHAR(32700)	LONG VARGRAPHIC データ型は 2 バイト文字用にのみ使用します。SQL Anywhere のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、LONG VARGRAPHIC は LONG VARCHAR と同等です。
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
REAL	REAL	REAL はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。
SMALLINT	SMALLINT	
TIME	TIME	SQL Anywhere TIME 値の小数点以下の秒の値は、ダウンロード時にトランケートされます。問題を回避するには、小数点以下の秒を使用しないでください。
TIMESTAMP	TIMESTAMP	
VARCHAR(*n*)	VARCHAR(*n*)	
VARCHAR(*n*) FOR BIT DATA	VARBINARY(*n*)	
VARGRAPHIC(*n*)	VARCHAR(2*n*)	VARGRAPHIC データ型は 2 バイト文字用にのみ使用します。SQL Anywhere のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、VARGRAPHIC は VARCHAR と同等です。

IBM DB2 メインフレーム・データのマッピング

DB2 メインフレームの統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモート・データ型がどのように DB2 メインフレームの統合データ型にマッピングされるのかを示します。たとえば、リモート・データベースの BIT 型のカラムは、統合データベースでは SMALLINT 型である必要があります。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
BIGINT	DECIMAL(20)	
BINARY(*n*<MRL)	VARCHAR(*n*) FOR BIT DATA	DB2 テーブルを作成する場合は、DB2 のページ・サイズに注意する必要があります。DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、DB2 のマニュアルを参照してください。
BINARY(*n*>=MRL)	BLOB(*n*)	DB2 テーブルを作成する場合は、DB2 のページ・サイズに注意する必要があります。DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、DB2 のマニュアルを参照してください。
BIT	SMALLINT	

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
CHAR(*n*<MRL)	VARCHAR(*n*)	MRL は DB2 の最大ロー長です。 DB2 の VARCHAR が保持できるのは 32672 バイトまでです (ページ・サイズによって異なります)。 DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。詳細については、使用している DB2 のマニュアルを参照してください。 SQL Anywhere の CHAR は、SQL Anywhere の VARCHAR と同じです。
CHAR(*n*>=MRL)	CLOB(*n*)	DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。詳細については、使用している DB2 のマニュアルを参照してください。 SQL Anywhere の CHAR は、SQL Anywhere の VARCHAR と同じです。 DB2 の CLOB の値は、SQL Anywhere または Ultra Light の値より長い可能性があります。ダウンロードされる値が大きすぎないことを確認する必要があります。
DATE	DATE	SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DATETIME	TIMESTAMP	
DECIMAL(*p*<32,*s*)	DECIMAL(*p*,*s*)	SQL Anywhere の DECIMAL の精度は 1 ～ 127 です。DB2 DECIMAL の最大精度は 31 です。SQL Anywhere の DECIMAL で精度が 31 より大きいデータは、DB2 に同期できません。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
DECIMAL(*p*>=32,*s*)		SQL Anywhere の DECIMAL で精度が 31 より大きいデータは、DB2 に同期できません。
DOUBLE	DOUBLE	DOUBLE はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。 正常に同期させるには、SQL Anywhere/Ultra Light の DOUBLE 値を DB2 メインフレームの DOUBLE 値の範囲内にしてください。
FLOAT(1-24)	REAL	FLOAT はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。 正常に同期させるには、DB2 メインフレームの REAL 値を SQL Anywhere/Ultra Light の REAL 値の範囲内にしてください。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
FLOAT(25-53)	DOUBLE	FLOAT はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。 正常に同期させるには、SQL Anywhere/ Ultra Light の DOUBLE 値を DB2 メインフレームの DOUBLE 値の範囲内にしてください。
IMAGE	BLOB(*n*)	
INTEGER	INTEGER	
LONG BINARY	BLOB(*n*)	
LONG NVARCHAR	CLOB(*n*)	DB2 のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合は、SQL Anywhere の long NVARCHAR を DB2 の CLOB に同期できます。Ultra Light には LONG NVARCHAR はありません。
LONG VARBIT	BLOB(*n*)	
LONG VARCHAR	CLOB(*n*)	
MONEY	DECIMAL(19,4)	

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
NCHAR(*c*)	VARCHAR(*n*) または CLOB(*n*)	SQL Anywhere の NCHAR は、SQL Anywhere の NVARCHAR と同じです。 DB2 のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合は、NCHAR を DB2 の VARCHAR または CLOB に同期できます。SQL Anywhere の NCHAR のサイズは文字単位で、DB2 の VARCHAR のサイズはバイト単位です。VARCHAR にマッピングする場合は、NCHAR のバイト数の合計が MRL より大きくならないようにしてください。そのようにできない場合、NCHAR は CLOB にマッピングします。NCHAR(*c*) のバイト数の計算は簡単ではありませんが、おおよそ *c*=*n*/4 です。一般的には、*c* が MRL/4 より小さい場合は VARCHAR(*n*) にマッピングし、*c* が MRL/4 以上の場合は CLOB(*n*) にマッピングします。
NTEXT	CLOB(*n*)	SQL Anywhere の NTEXT は、SQL Anywhere の LONG NVARCHAR と同じです。 DB2 のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合は、NCHAR を DB2 の VARCHAR または CLOB に同期できます。SQL Anywhere の NCHAR のサイズは文字単位で、DB2 の VARCHAR のサイズはバイト単位です。VARCHAR にマッピングする場合は、NCHAR のバイト数の合計が MRL より大きくならないようにしてください。そのようにできない場合、NCHAR は CLOB にマッピングします。NCHAR(*c*) のバイト数の計算は簡単ではありませんが、おおよそ *c*=*n*/4 です。一般的には、*c* が MRL/4 より小さい場合は VARCHAR(*n*) にマッピングし、*c* が MRL/4 以上の場合は CLOB(*n*) にマッピングします。
NUMERIC(*p*<32,*s*)	NUMERIC(*p*,*s*)	

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
NUMERIC(*p*>=32,*s*)		DB2 のデータ型に対応するデータ型がありません。
NVARCHAR(*c*)	VARCHAR(*n*) または CLOB(*n*)	DB2 のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合は、NCHAR を DB2 の VARCHAR または CLOB に同期できます。SQL Anywhere の NCHAR のサイズは文字単位で、DB2 の VARCHAR のサイズはバイト単位です。VARCHAR にマッピングする場合は、NCHAR のバイト数の合計が MRL より大きくならないようにしてください。そのようにできない場合、NCHAR は CLOB にマッピングします。NCHAR(*c*) のバイト数の計算は簡単ではありませんが、おおよそ *c*=*n*/4 です。一般的には、*c* が MRL/4 より小さい場合は VARCHAR(*n*) にマッピングし、*c* が MRL/4 以上の場合は CLOB(*n*) にマッピングします。
REAL	REAL	REAL はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。 正常に同期させるには、DB2 メインフレームの REAL 値を SQL Anywhere/Ultra Light の REAL 値の範囲内にしてください。
SMALLDATETIME	TIMESTAMP	
SMALLINT	SMALLINT	
SMALLMONEY	DECIMAL(10,4)	
TEXT	CLOB(*n*)	SQL Anywhere の TEXT は、SQL Anywhere の LONG VARCHAR と同じです。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
TIME	TIMESTAMP または TIME	小数点以下の秒がある SQL Anywhere と Ultra Light の TIME 値には、DB2 の TIMESTAMP が必要です。小数点以下の秒が常に 0 の SQL Anywhere と Ultra Light の TIME 値には、DB2 の TIME を使用できます。
TIMESTAMP	TIMESTAMP	
TINYINT	SMALLINT	
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR の DB2 での使用はおすすめしません。代わりに UNIQUEIDENTIFIER を使用してください。
UNSIGNED BIGINT	DECIMAL(20)	DB2 の値は負でない必要があります。
UNSIGNED INTEGER	DECIMAL(11)	DB2 の値は負でない必要があります。
UNSIGNED SMALLINT	DECIMAL(5)	DB2 の値は負でない必要があります。
UNSIGNED TINYINT	SMALLINT	DB2 の値は負でない必要があります。
VARBINARY(*n*<MRL)	VARCHAR(*n*) FOR BIT DATA	DB2 テーブルを作成する場合は、DB2 のページ・サイズに注意する必要があります。DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、DB2 のマニュアルを参照してください。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
VARBINARY(*n*>=MRL)	BLOB(*n*)	DB2 テーブルを作成する場合は、DB2 のページ・サイズに注意する必要があります。DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、DB2 のマニュアルを参照してください。
VARBIT(*n*<MRL)	VARCHAR(*n*)	DB2 テーブルを作成する場合は、DB2 のページ・サイズに注意する必要があります。DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、DB2 のマニュアルを参照してください。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
VARBIT(*n*>=MRL)	CLOB(*n*)	DB2 テーブルを作成する場合は、DB2 のページ・サイズに注意する必要があります。DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。テーブルに含まれる全カラムの長さがこの制限を超えることはできません。テーブルに BLOB または CLOB カラムがある場合は、BLOB または CLOB データを直接カウントするのではなく、LOB ロケータを使用してロー長をカウントできます。詳細については、DB2 のマニュアルを参照してください。
VARCHAR(*n*<MRL)	VARCHAR(*n*)	DB2 の VARCHAR が保持できるのは 32672 バイトまでです (ページ・サイズによって異なります)。 DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。詳細については、使用している DB2 のマニュアルを参照してください。 SQL Anywhere の CHAR は、SQL Anywhere の VARCHAR と同じです。
VARCHAR(*n*>=MRL)	CLOB(*n*)	DB2 には、ページ・サイズに基づく最大ロー長 (MRL) があります。ページ・サイズが 4K の場合の MRL は 4005 です。8K の場合は 8101、16K の場合は 16293、32K の場合は 32677 になります。詳細については、使用している DB2 のマニュアルを参照してください。 SQL Anywhere の CHAR は、SQL Anywhere の VARCHAR と同じです。 DB2 の CLOB の値は、SQL Anywhere または Ultra Light の値より長い可能性があります。ダウンロードされる値が大きすぎないことを確認する必要があります。

SQL Anywhere または Ultra Light のデータ型	IBM DB2 メインフレームのデータ型	注意
XML	CLOB(*n*)	SQL Anywhere の XML は、SQL Anywhere の LONG VARCHAR と同じです。

SQL Anywhere または Ultra Light のリモート・データ型へのマッピング

次の表は、DB2 メインフレームの統合データ型がどのように SQL Anywhere または Ultra Light のリモート・データ型にマッピングされるのかを示します。たとえば、統合データベースの INT 型のカラムは、リモート・データベースでは INTEGER 型である必要があります。

IBM DB2 メインフレームのデータ型	SQL Anywhere または Ultra Light のデータ型	注意
BLOB	LONG BINARY	
CHAR(*n*)	VARCHAR(*n*)	SQL Anywhere には DB2 の CHAR に対応するデータ型がありません。同期される統合データベースのカラムでは、CHAR を使用しないでください。DB2 の CHAR カラムを同期させる必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。
CHAR(*n*) FOR BIT DATA	BINARY(*n*)	SQL Anywhere には DB2 の CHAR に対応するデータ型がありません。同期される統合データベースのカラムでは、CHAR を使用しないでください。DB2 の CHAR カラムを同期させる必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。
CLOB(*n*)	LONG VARCHAR	
DATE	DATE	SQL Anywhere と Ultra Light では、時刻の値は 00:00:00 の形式である必要があります。
DBCLOB(*n*)	LONG VARCHAR	DBCLOB(*n*) データ型は 2 バイト文字用にのみ使用します。SQL Anywhere のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、DBCLOB(*n*) は CLOB と同等です。

IBM DB2 メインフレームのデータ型	SQL Anywhere または Ultra Light のデータ型	注意
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
DOUBLE	DOUBLE	DOUBLE はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。 正常に同期させるには、SQL Anywhere/Ultra Light の DOUBLE 値を DB2 の DOUBLE 値の範囲内にしてください。
FLOAT	DOUBLE	FLOAT はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。 正常に同期させるには、SQL Anywhere/Ultra Light の DOUBLE 値を DB2 の DOUBLE 値の範囲内にしてください。
GRAPHIC(*n*)	VARCHAR(2*n*)	DB2 の GRAPHIC にはブランクが埋め込まれます。SQL Anywhere には DB2 の GRAPHIC に対応するデータ型がありません。同期される統合データベースのカラムでは、GRAPHIC を使用しないでください。 GRAPHIC データ型は 2 バイト文字用にのみ使用します。SQL Anywhere のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、GRAPHIC は CHAR と同等です。
INT	INTEGER	
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	

IBM DB2 メインフレームのデータ型	SQL Anywhere または Ultra Light のデータ型	注意
REAL	DOUBLE	REAL はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。 DB2 メインフレームでは、REAL、DOUBLE、FLOAT の範囲は同じであり、-7.2E+75 ～ 7.2E+75 です。負の数の最大値は約 -5.4E-79 であり、正の数の最小値は約 5.4E-79 です。SA/UL の REAL の範囲は -3.402823e+38 ～ 3.402823e+38 であり、DOUBLE の範囲は 2.22507385850721e-308 ～ 1.79769313486231e+308 です。正常に同期させるには、DB2 メインフレームの REAL は SA/UL の REAL の範囲内、SA/UL の DOUBLE は DB2 メインフレームの DOUBLE の範囲内である必要があります。
ROWID		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。ROWID は DB2 サーバによって維持されます。このデータ型は同期できません。
SMALLINT	SMALLINT	
TIME	TIME	SQL Anywhere TIME 値の小数点以下の秒の値は、ダウンロード時にトランケートされます。問題を回避するには、小数点以下の秒を使用しないでください。
TIMESTAMP	TIMESTAMP	
VARCHAR(*n*)	VARCHAR(*n*)	
VARCHAR(*n*) FOR BIT DATA	VARBINARY(*n*)	

IBM DB2 メインフレームのデータ型	SQL Anywhere または Ultra Light のデータ型	注意
VARGRAPHIC(*n*)	VARCHAR(2*n*)	VARGRAPHIC データ型は 2 バイト文字用にのみ使用します。SQL Anywhere のデータ型に対応するデータ型がありません。DB2 の文字セットが Unicode の場合、VARGRAPHIC は VARCHAR と同等です。

Microsoft SQL Server データのマッピング

Microsoft SQL Server の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモート・データ型がどのように Microsoft SQL Server の統合データ型にマッピングされるのかを示します。たとえば、リモート・データベースの DATE 型のカラムは、統合データベースでは DATETIME 型である必要があります。

SQL Anywhere または Ultra Light のデータ型	Microsoft SQL Server データ型	注意
BIGINT	BIGINT	
BINARY(*n*<=8000)	VARBINARY(*n*)	
BINARY(*n*>8000)	VARBINARY(MAX)	
BIT	BIT	
CHAR(*n*<=8000)	VARCHAR(*n*)	
CHAR(*n*>8000)	VARCHAR(MAX)	
DATE	DATE	
DATETIME	DATETIME2	SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。
DECIMAL(*p*=<38,*s*)	DECIMAL(*p*,*s*)	SQL Server の DECIMAL/NUMERIC の精度は 1 ～ 38 なので、*p* は 39 より小さい必要があります。
DECIMAL(*p*>38,*s*)		SQL Server のデータ型に対応するデータ型がありません。
DOUBLE	FLOAT(53)	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	VARBINARY(MAX)	

SQL Anywhere または Ultra Light のデータ型	Microsoft SQL Server データ型	注意
INTEGER	INT	
LONG BINARY	VARBINARY(MAX)	
LONG NVARCHAR	NVARCHAR(MAX)	
LONG VARBIT	VARCHAR(MAX)	
LONG VARCHAR	VARCHAR(MAX)	
MONEY	MONEY	
NCHAR(*n*<=4000)	NVARCHAR(*c*)	
NCHAR(*n*>4000)	NVARCHAR(MAX)	
NTEXT	NVARCHAR(MAX)	
NUMERIC(*p*=<38,*s*)	NUMERIC(*p*,*s*)	SQL Server の DECIMAL/NUMERIC の精度は 1 ～ 38 なので、p は 39 より小さい必要があります。
NUMERIC(*p*>38,*s*)		SQL Server のデータ型に対応するデータ型がありません。
NVARCHAR(*n*<=4000)	NVARCHAR(*c*)	
NVARCHAR(*n*>4000)	NVARCHAR(MAX)	
REAL	REAL	
SMALLDATETIME	SMALLDATETIME	SQL Anywhere と Ultra Light の SMALLDATETIME は TIMESTAMP として実装されます。SQL Server の SMALLDATETIME は分の精度です。29.998 秒以下の値は直近の分に切り捨てられます。29.999 秒以上の値は直近の分に切り上げられます。SQL Anywhere と Ultra Light の SMALLDATETIME はミリ秒の精度です。正常に同期させるには、SQL Anywhere または Ultra Light の SMALLDATETIME の丸め単位を分にしてください。年は、1900 ～ 2078 の範囲内である必要があります。

SQL Anywhere または Ultra Light のデータ型	Microsoft SQL Server データ型	注意
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	VARCHAR(MAX)	
TIME	TIME	SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。
TIMESTAMP	DATETIME2	SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。
TINYINT	TINYINT	ダウンロードの場合、値は負でない必要があります。
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
UNIQUEIDENTIFIERSTR	UNIQUEIDENTIFIER	
UNSIGNED BIGINT	NUMERIC(20)	ダウンロードの場合、値は負でない必要があります。
UNSIGNED INTEGER	NUMERIC(11)	ダウンロードの場合、値は負でない必要があります。
UNSIGNED TINYINT	TINYINT	ダウンロードの場合、値は負でない必要があります。
UNSIGNED SMALLINT	INT	ダウンロードの場合、値は負でない必要があります。
VARBINARY(*n*<=8000)	VARBINARY(*n*)	

SQL Anywhere または Ultra Light のデータ型	Microsoft SQL Server データ型	注意
VARBINARY(*n*>8000)	VARBINARY(MAX)	
VARBIT(*n*<=8000)	VARCHAR(*n*)	
VARBIT(*n*>8000)	VARCHAR(MAX)	
VARCHAR(*n*<=8000)	VARCHAR(*c*)	
VARCHAR(*n*>8000)	VARCHAR(MAX)	
XML	XML または VARCHAR(MAX)	SQL Server 2005 の場合は、XML を使用します。それ以外のバージョンの場合は、VARCHAR(MAX) を使用します。

SQL Anywhere または Ultra Light のリモート・データ型へのマッピング

次の表は、Microsoft SQL Server の統合データ型がどのように SQL Anywhere および Ultra Light のリモート・データ型にマッピングされるのかを示します。たとえば、リモート・データベースの TEXT 型のカラムは、統合データベースでは LONG VARCHAR 型である必要があります。

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	注意
BIGINT	BIGINT	
BINARY(*n*)	BINARY(*n*)	
BIT	BIT	

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	注意
CHAR(*n*)	VARCHAR(*n*)	Microsoft SQL Server の CHAR カラムは、ブランクが埋め込まれます。SQL Anywhere の CHAR カラムは、デフォルトではブランクが埋め込まれず、VARCHAR カラムと同等になります。そのため、Microsoft SQL Server の同期テーブルでは CHAR データ型の使用を避けるようにしてください。Microsoft SQL Server 統合データベースで CHAR データ型を使用する必要がある場合は、SQL Anywhere の CHAR と SQL Anywhere 以外の CHAR との違いを解決できるように、-b コマンド・ライン・オプションを指定して Mobile Link サーバを実行してください。
DATE	DATE	
DATETIME	TIMESTAMP または DATETIME	SQL Server の DATETIME 値は 1/300 秒の精度です。小数点以下の秒の末尾の数字は 0、3、6 のいずれかになります。その他の数字はこの 3 つのいずれかに丸められます。具体的には、0 と 1 は 0 に、2 と 3 と 4 は 3 に、5 と 6 と 7 と 8 は 6 に、9 は 10 に丸められます。ダウンロード時には、SQL Anywhere は SQL Server からの元の値を保持しますが、アップロード時には、値は元の値と完全には一致しないことがあります。DATETIME をプライマリ・キーに使用すると、競合を解決できないことがあります。DATETIME を正常に同期させるには、秒の小数点以下の丸め単位を 10 ミリ秒にすることをおすすめします。年は、1753 ～ 9999 の範囲内である必要があります。
DATETIME2	TIMESTAMP	SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	注意
DECIMAL(*p*,*s*)	DECIMAL(*p*,*s*)	
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	LONG BINARY	
INT	INT	
MONEY	MONEY	
NCHAR(*n*)	NVARCHAR(*c*)	Ultra Light では使用できません。 SQL Anywhere の NCHAR は、SQL Anywhere 以外のデータベースの NCHAR と同等ではありません。SQL Anywhere の NCHAR に対応するのは NVARCHAR です。同期される統合データベースのカラムでは、NCHAR を使用しないでください。SQL Anywhere 以外の NCHAR を使用する必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。
NTEXT	LONG NVARCHAR	Ultra Light では使用できません。
NVARCHAR(*c*)	NVARCHAR(*c*)	Ultra Light では使用できません。
NVARCHAR(MAX)	LONG NVARCHAR	
NUMERIC(*p*,*s*)	NUMERIC(*p*,*s*)	
REAL	REAL	REAL はあいまいな値なので、統合データベースとリモート・データベースとで厳密に同じ値を使用できない場合、問題を引き起こすことがあります。取り得るすべての値がテスト済みであるわけではないので、注意が必要です。問題の発生を回避するため、このような型をプライマリ・キーの一部として使用しないでください。

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	注意
SMALLDATETIME	SMALLDATETIME	SQL Anywhere と Ultra Light の SMALLDATETIME は TIMESTAMP として実装されます。SQL Server の SMALLDATETIME は分の精度です。29.998 秒以下の値は直近の分に切り捨てられます。29.999 秒以上の値は直近の分に切り上げられます。SQL Anywhere と Ultra Light の SMALLDATETIME はミリ秒の精度です。正常に同期させるには、SQL Anywhere または Ultra Light の SMALLDATETIME の丸め単位を分にしてください。年は、1900 ～ 2078 の範囲内である必要があります。
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	LONG VARCHAR	
TIME	TIME	SQL Server の DATETIME2 値と TIME 値は、100 ナノ秒の精度です。ただし TIMESTAMP 値と TIME 値は、1 マイクロ秒の精度しかありません。DATETIME2 と TIME を正常に同期させるには、秒の小数点以下の丸め単位を 1 マイクロ秒にすることをおすすめします。

Microsoft SQL Server データ型	SQL Anywhere または Ultra Light のデータ型	注意
TIMESTAMP	VARBINARY(8)	Microsoft SQL Server 内では、TIMESTAMP はローが変更されるたびに増分されるバイナリ・カウンタです。各テーブルに存在できる TIMESTAMP カラムは 1 つのみであるため、TIMESTAMP カラムを同期する意味はありません。同期に含める必要がある場合は、SQL Anywhere または Ultra Light の VARBINARY(8) データ型にマッピングします。 このタイムスタンプ・カラムはサーバによって維持されるため、明示的に挿入および更新することはできません。このようなカラムがあるテーブルのアップロード・スクリプトを実装する場合は、このことを念頭に置いてください。
TINYINT	TINYINT	
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
VARBINARY(*n*)	VARBINARY(*n*)	
VARBINARY(MAX)	LONG BINARY	
VARCHAR(*n*)	VARCHAR(*n*)	
VARCHAR(MAX)	LONG VARCHAR	
XML	XML	

MySQL データのマッピング

MySQL の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモート・データ型がどのように MySQL の統合データ型にマッピングされるのかを示します。たとえば、リモート・データベースの TEXT 型のカラムは、統合データベースでは LONGTEXT 型である必要があります。

SQL Anywhere または Ultra Light のデータ型	MySQL のデータ型	注意
BIGINT	BIGINT	
BINARY(*n*<=255)	BINARY(*n*)	
BINARY(*n*>255)	BLOB	
BIT	BIT	
CHAR(*n*<=255)	CHAR(*n*)	
CHAR(*n*>255)	TEXT(*n*)	
DATE	DATE	年は 1000 ～ 9999 である必要があります。
DATETIME	DATETIME	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。
DECIMAL(*p*<=65,*s*<=30)	DECIMAL(*p*,*s*)	
DECIMAL(*p*>65,*s*>30)		精度が 65 より大きい場合、または位取りが 30 より大きい場合、MySQL のデータ型に対応するデータ型がありません。
DOUBLE	DOUBLE	
FLOAT	FLOAT	
IMAGE	LONGBLOB	
INTEGER	INTEGER	
LONG BINARY	LONGBLOB	
LONG NVARCHAR	LONGTEXT CHARACTER SET UTF8	

SQL Anywhere または Ultra Light のデータ型	MySQL のデータ型	注意
LONG VARBIT	LONGTEXT	
LONG VARCHAR	LONGTEXT	
MONEY	NUMERIC(19,4)	
NCHAR(*n*<=255)	CHAR(*n*) CHARACTER SET UTF8	
NCHAR(*n*>255)	TEXT CHARACTER SET UTF8	
NTEXT	LONGTEXT CHARACTER SET UTF8	
NUMERIC(*p*<=65,*s*<=30)	DECIMAL(*p*,*s*)	
NUMERIC(*p*>65,*s*>30)		MySQL のデータ型に対応するデータ型がありません。
NVARCHAR(*n*)	VARCHAR(*n*) CHARACTER SET UTF8	
REAL	REAL	
SMALLDATETIME	DATETIME	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。
SMALLINT	SMALLINT	
SMALLMONEY	NUMERIC(10,4)	
TEXT	LONGTEXT	
TIME	TIME	MySQL の TIME データ型は、秒の小数点以下をサポートしていません。
TIMESTAMP	DATETIME	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。

SQL Anywhere または Ultra Light のデータ型	MySQL のデータ型	注意
TINYINT	TINYINT UNSIGNED	TINYINT は、SQL Anywhere および Ultra Light では常に符号なしです。
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	
VARBINARY(*n*)	VARCHAR(*n*)	
VARBIT(*n*<=8000)	VARCHAR(*n*)	
VARBIT(*n*>8000)	TEXT	
VARCHAR(*n*)	VARCHAR(*n*)	
XML	LONGTEXT	

SQL Anywhere または Ultra Light のリモート・データ型へのマッピング

次の表は、MySQL の統合データ型がどのように SQL Anywhere および Ultra Light のリモート・データ型にマッピングされるのかを示します。たとえば、統合データベースの BOOL 型のカラムは、リモート・データベースでは BIT 型である必要があります。

MySQL のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
BIGINT	BIGINT	
BINARY(*n*)	BINARY(*n*)	
BIT(1)	BIT	
BIT(*n*>1)	UNSIGNED BIGINT	
BLOB(*n*<=32767)	VARBINARY(*n*)	
BLOB(*n*>32767)	IMAGE	
BOOL	BIT	
CHAR(*n*)	CHAR(*n*)	
DATE	DATE	年は 1000 ～ 9999 である必要があります。

MySQL のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
DATETIME	DATETIME	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。
DOUBLE	DOUBLE	
DECIMAL	DECIMAL	
ENUM		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
GEOMETRY		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
INTEGER	INTEGER	
LINESTRING		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
LONGBLOB	IMAGE	
LONGTEXT	TEXT	
MEDIUMBLOB	IMAGE	
MEDIUMINT	INTEGER	
MEDIUMTEXT	TEXT	
MULTILINESTRING		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
MULTIPOINT		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
MULTIPOLYGON		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
NCHAR	NCHAR	Ultra Light では使用できません。
NUMERIC	NUMERIC	
NVARCHAR	NVARCHAR	Ultra Light では使用できません。

MySQL のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
POINT		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
POLYGON		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
REAL	REAL	
SET		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
SMALLINT	SMALLINT	
TEXT(*n*<=32767)	VARCHAR(*n*)	
TEXT(*n*>32767)	TEXT	
TIME	TIME	MySQL の TIME データ型は、秒の小数点以下をサポートしていません。MySQL の TIME の範囲は '-838:59:59' ～ '838:59:59' です。SQL Anywhere および Ultra Light の TIME の範囲は '00:00:00.000000' ～ '23:59:59:999999' です。
TIMESTAMP	TIMESTAMP	MySQL の DATETIME データ型は、秒の小数点以下をサポートしていません。年は 1000 ～ 9999 である必要があります。MySQL は TIMESTAMP カラムでの自動初期化と更新機能を提供していますが、SQL Anywhere および Ultra Light は自動初期化のみを提供しています。
TINYBLOB	VARBINARY	
TINYINT	SMALLINT	TINYINT は、SQL Anywhere および Ultra Light では常に符号なしです。正の値にしてください。
TINYINT UNSIGNED	TINYINT	TINYINT は、SQL Anywhere および Ultra Light では常に符号なしです。
TINYTEXT	VARCHAR	
VARBINARY(*n*<=32767)	VARBINARY(*n*)	

MySQL のデータ型	SQL Anywhere または Ultra Light のデータ型	注意
VARBINARY(*n*>32767)	IMAGE	
VARCHAR(*n*<=32767)	VARCHAR(*n*)	
VARCHAR(*n*>32767)	TEXT	
YEAR[(2|4)]	INTEGER	SQL Anywhere および Ultra Light は YEAR データ型をサポートしていません。YEAR は、リモート・データベースの INTEGER にマッピングしてください。INTEGER 値は 1000 ～ 9999 である必要があります。

Oracle データのマッピング

Oracle の統合データ型へのマッピング

次の表は、SQL Anywhere および Ultra Light のリモート・データ型がどのように Oracle の統合データ型にマッピングされるのかを示します。たとえば、リモート・データベースの BIT 型のカラムは、統合データベースでは NUMBER 型である必要があります。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	注意
BIGINT	NUMBER(20)	
BINARY(*n*<=2000)	RAW(*n*)	
BINARY(*n*>2000)	BLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
BIT	NUMBER(1)	
CHAR(*n*<=4000)	VARCHAR2(*n* byte)	Oracle の VARCHAR2 を使用すると、バイト数または文字数の最大値を指定できます。VARCHAR2 データの最大長は 4000 バイトです。文字数を指定する場合は、データの最大長が 4000 バイトを超えないようにしてください。
CHAR(*n*>4000)	CLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
DATE	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。年は、1 ～ 9999 の範囲内である必要があります。 Interactive SQL ユーティリティを使用する場合は、Return_date_time_as_string オプションをオフにしてから、SQL 文を実行します。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	注意
DATETIME	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。年は、1 ～ 9999 の範囲内である必要があります。 Interactive SQL ユーティリティを使用する場合は、Return_date_time_as_string オプションをオフにしてから、SQL 文を実行します。
DECIMAL(*p*<=38,*s*)	NUMBER(*p*, 0<=*s*<=38)	SQL Anywhere の DECIMAL では、p は 1 ～ 127 で、s は常に p 以下です。Oracle の NUMBER では、p は 1 ～ 38 で、s は -84 ～ 127 です。同期させるには、Oracle の NUMBER の位取りを 0 ～ 38 に制限する必要があります。
DECIMAL(*p*>38,*s*)		Oracle のデータ型に対応するデータ型がありません。
DOUBLE	DOUBLE PRECISION または BINARY_DOUBLE[1]	Oracle 10g の BINARY_FLOAT と BINARY_DOUBLE の特別な値 INF、-INF、NAN は、SQL Anywhere または Ultra Light とは同期できません。
FLOAT(*p*)	FLOAT(*p*)	
IMAGE	BLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
INTEGER	INT	
LONG BINARY	BLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	注意
LONG NVARCHAR	NCLOB	Oracle の CLOB と NCLOB は、最大で 4G のデータを保持できます。SQL Anywhere の LONG VARCHAR と LONG NVARCHAR が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
LONG VARBIT	CLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
LONG VARCHAR	CLOB	Oracle の CLOB と NCLOB は、最大で 4G のデータを保持できます。SQL Anywhere の LONG VARCHAR と LONG NVARCHAR が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
MONEY	NUMBER(19,4)	
NCHAR(*c*)	NVARCHAR2(*c* char) または NCLOB	SQL Anywhere の NCHAR と Oracle の NVARCHAR2 では、サイズは Unicode 文字の最大文字数を示します。Oracle の NVARCHAR2 のデータ長が 4000 バイトを超えることはできません。文字サイズから最大バイト長を計算することは困難です。一般的に、サイズが 1000 を超える場合は NCLOB に、そうでない場合は NVARCHAR2 に、それぞれマッピングします。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	注意
NTEXT	NCLOB	Oracle の NCLOB は、最大で 4G のデータを保持できます。SQL Anywhere の NTEXT (または LONG NVARCHAR) が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
NUMERIC(*p*<=38,*s*)	NUMBER(*p*, 0<=*s*<=38)	SQL Anywhere の NUMERIC では、p は 1 ～ 127 で、s は常に p 以下です。Oracle の NUMBER では、p は 1 ～ 38 で、s は -84 ～ 127 です。同期させるには、Oracle の NUMBER の位取りを 0 ～ 38 に制限する必要があります。
NUMERIC(*p*>38,*s*)		Oracle のデータ型に対応するデータ型がありません。
NVARCHAR	NVARCHAR2(*c* char) または NCLOB	SQL Anywhere の NCHAR と Oracle の NVARCHAR2 では、サイズは Unicode 文字の最大文字数を示します。Oracle の NVARCHAR2 のデータ長が 4000 バイトを超えることはできません。文字サイズから最大バイト長を計算することは困難です。一般的に、サイズが 1000 を超える場合は NCLOB に、そうでない場合は NVARCHAR2 に、それぞれマッピングします。
REAL	REAL または BINARY_FLOAT[1]	Oracle 10g の BINARY_FLOAT と BINARY_DOUBLE の特別な値 INF、-INF、NAN は、SQL Anywhere または Ultra Light とは同期できません。
SMALLDATETIME	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。年は、1 ～ 9999 の範囲内である必要があります。
SMALLINT	NUMBER(5)	

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	注意
SMALLMONEY	NUMBER(10,4)	
TEXT	CLOB	Oracle の CLOB は、最大で 4G のデータを保持できます。SQL Anywhere の TEXT (または LONG VARCHAR) が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
TIME	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。 Interactive SQL ユーティリティを使用する場合は、Return_date_time_as_string オプションをオフにしてから、SQL 文を実行します。
TIMESTAMP	DATE[2] または TIMESTAMP	Oracle の DATE データ型には小数点以下の秒がないので、このデータ型を使用すると SQL Anywhere または Ultra Light の小数点以下の秒を保持できません。問題を回避するには、小数点以下の秒を使用しないでください。年は、1 ～ 9999 の範囲内である必要があります。 Interactive SQL ユーティリティを使用する場合は、Return_date_time_as_string オプションをオフにしてから、SQL 文を実行します。
TINYINT	NUMBER(3)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNSIGNED BIGINT	NUMBER(20)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNSIGNED INTEGER	NUMBER(11)	ダウンロードの場合、Oracle の値は負でない必要があります。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	注意
UNSIGNED SMALLINT	NUMBER(5)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNSIGNED TINYINT	NUMBER(3)	ダウンロードの場合、Oracle の値は負でない必要があります。
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR の Oracle での使用はおすすめしません。代わりに UNIQUEIDENTIFIER を使用してください。
VARBINARY(*n*<=2000)	RAW(*n*)	
VARBINARY(*n*>2000)	BLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
VARBIT(*n*<=4000)	VARCHAR2(*n* byte)	
VARBIT(*n*>4000)	CLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
VARCHAR(*n*<=4000)	VARCHAR2(*n* byte)	Oracle の VARCHAR2 を使用すると、バイト数または文字数の最大値を指定できます。VARCHAR2 データの最大長は 4000 バイトです。文字数を指定する場合は、データの最大長が 4000 バイトを超えないようにしてください。
VARCHAR(*n*>4000)	CLOB	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。

SQL Anywhere または Ultra Light のデータ型	Oracle データ型	注意
XML	CLOB	Oracle の CLOB と NCLOB は、最大で 4G のデータを保持できます。SQL Anywhere の XML が保持できるのは 2G までです。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。

[1] Oracle バージョン 10g 以降にのみ該当します。

[2] Oracle バージョン 8i 以降にのみ該当します。

注意
LONG データ型は、Oracle 8、8i、9i では使用されなくなりました。

Oracle の LONG データ型を正常に同期させるには、iAnywhere Solutions Oracle ODBC ドライバの **[ODBC データ・ソース設定]** ウィンドウで **[Oracle Force Retrieval Of Long Columns]** のオプションを確認します。

SQL Anywhere または Ultra Light のリモート・データ型へのマッピング

次の表は、Oracle の統合データ型がどのように SQL Anywhere および Ultra Light のリモート・データ型にマッピングされるのかを示します。たとえば、統合データベースの LONG 型のカラムは、リモート・データベースでは LONG VARCHAR 型である必要があります。

Oracle データ型	SQL Anywhere または Ultra Light のデータ型	注意
BFILE	LONG BINARY	ダウンロードのみ。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
BINARY_DOUBLE	DOUBLE	BINARY_FLOAT の特別な値 INF、-INF、NAN は、SQL Anywhere または Ultra Light とは同期できません。Oracle の FLOAT と DOUBLE の精度は、SQL Anywhere と Ultra Light のものとは異なります。精度によっては、データの値が変わる可能性があります。

Oracle データ型	SQL Anywhere または Ultra Light のデータ型	注意
BINARY_FLOAT	REAL	BINARY_FLOAT の特別な値 INF、-INF、NAN は、SQL Anywhere または Ultra Light とは同期できません。Oracle の FLOAT と DOUBLE の精度は、SQL Anywhere と Ultra Light のものとは異なります。精度によっては、データの値が変わる可能性があります。
BLOB	LONG BINARY	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
CHAR(*n* byte)	VARCHAR(*n*)	SQL Anywhere の CHAR は、Oracle の CHAR と同等ではありません。SQL Anywhere の CHAR に対応するのは VARCHAR です。同期される統合データベースのカラムでは、CHAR/NCHAR を使用しないでください。SQL Anywhere 以外の CHAR を使用する必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。 SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。
CLOB	LONG VARCHAR	Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
DATE	TIMESTAMP	年は、1 ～ 9999 の範囲内である必要があります。
INTERVAL YEAR(*year_precision*) TO MONTH		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
INTERVAL DAY(*day_precision*) TO SECOND(*p*)		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。

Oracle データ型	SQL Anywhere または Ultra Light のデータ型	注意
LONG	LONG VARCHAR	
LONG RAW	LONG BINARY	
NCHAR(*c* char)	NVARCHAR(*c*)	SQL Anywhere の NCHAR は、Oracle の NCHAR と同等ではありません。SQL Anywhere の NCHAR に対応するのは NVARCHAR です。同期される統合データベースのカラムでは、NCHAR を使用しないでください。SQL Anywhere 以外の NCHAR を使用する必要がある場合は、-b オプションを指定して Mobile Link サーバを実行してください。 SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。
NCLOB	LONG NVARCHAR	Ultra Light では使用できません。 Oracle の値が SQL Anywhere または Ultra Light の値より長い可能性があるので、ダウンロード時には値が大きすぎないことを確認してください。
NUMBER(*p*,*s*)	NUMBER(*p*,*s*)	SQL Anywhere の NUMBER では、*p* は 1 ～ 127 であり、*s* は常に *p* 以下です。Oracle の NUMBER では、*p* は 1 ～ 38 であり、*s* は -84 ～ 127 です。同期させるには、Oracle の NUMBER の位取りを 0 ～ 38 にしてください。
NVARCHAR2(*c* char)	NVARCHAR(*c*)	Ultra Light では使用できません。 SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。
RAW	BINARY	SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。

Oracle データ型	SQL Anywhere または Ultra Light のデータ型	注意
ROWID	VARCHAR(64)	UROWID と ROWID は読み込み専用なので、同期対象になることはほとんどありません。
TIMESTAMP(*p*<=6)	TIMESTAMP	p<6 の場合、SQL Anywhere または Ultra Light の値が同じ精度になっていることを確認する必要があります。そうしないと、競合を検出できなかったり、ローの重複が発生したりする可能性が生じます。年は、1 ～ 9999 の範囲内である必要があります。
TIMESTAMP(*p*>6)		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
TIMESTAMP(*p*) WITH LOCAL TIME ZONE		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
TIMESTAMP(*p*) WITH TIME ZONE		SQL Anywhere または Ultra Light のデータ型に対応するデータ型がありません。
UROWID	VARCHAR(64)	UROWID と ROWID は読み込み専用なので、同期対象になることはほとんどありません。
VARCHAR2(*n* byte)	VARCHAR(*n*)	SQL Anywhere または Ultra Light の値が Oracle の値より長い可能性があるので、アップロード時には値が大きすぎないことを確認してください。

文字セットの考慮事項

目次

文字セットの考慮事項

テキストの各文字は、1 バイトまたはそれ以上のバイトで表現されます。文字からバイナリ・コードへのマッピングを「文字セット・エンコード」といいます。ヨーロッパ言語など、小規模なアルファベット体系を持つ言語で使われる文字セットには、シングルバイト表現が使用されるものがあります。Unicode などのように 2 バイト表現を使用する文字セットもあります。2 バイト文字セットは、各文字について記憶領域を 2 倍使用するため、はるかに多くの文字を表現できます。

変換エラーが起きたり、データが失われたりするのは、1 つの文字セットを使用しているテキストを別の文字セットに変換しなければならないときです。すべての文字がすべての文字セットで表示できるわけではありません。特に、シングルバイト文字セットは、使用可能なコードの数が限られているので、表示できる文字はマルチバイト・システムより少なくなります。

Mobile Link リモート・データベースの文字セットがユーザの統合データベースと同じ場合は、文字変換の問題は起こりません。

ディレクトリのリストのようにインデックスを構築したり順序リストを準備したりするには、テキストをソートする必要がよくあります。「ソート順」は、文字の順序を特定します。たとえば、一般的にソート順では文字 "a" は文字 "b" の前に位置し、文字 "b" は文字 "c" の前に位置します。

各データベースには、「照合順」があります。照合順は、データベースを作成するときに設定します。設定方法は、データベースのシステムによって異なります。照合順は、データベースの文字セットとソート順の両方を定義します。

ヒント
可能な場合は常に、使用しているリモート・データベースの照合順を、使用している統合データベースの照合順と同じになるように定義してください。この方法により、間違った変換をする可能性が減ります。

参照

- SQL Anywhere クライアント：「国際言語と文字セット」 『SQL Anywhere サーバ - データベース管理』
- Ultra Light クライアント：「Ultra Light 文字セット」 『Ultra Light データベース管理とリファレンス』
- RDBMS 固有の情報：「Mobile Link 統合データベース」 3 ページ

同期中の文字セット変換

同期中は、ある文字セットの文字を別の文字セットに変換しなければならないことがあります。次の変換は、文字がリモート・アプリケーションと統合データベースの間で渡されるときに起こります。

アップロード中の文字セット変換

Mobile Link クライアントは、リモート・データベースの文字セットを使用して Mobile Link サーバにデータを送ります。

1. Mobile Link サーバは、Unicode ODBC API を使用して統合データベースと通信します。通信するために、Mobile Link サーバは、リモート・データベースから受信したすべての文字を Unicode に変換し、Unicode を ODBC ドライバに送信します。
2. 必要に応じて、統合データベース・サーバの ODBC ドライバは文字を Unicode から統合データベースの文字セットに変換します。この変換を制御できるのは、使用している統合データベース・システムの ODBC ドライバだけです。したがって、2 つの異なるデータベース・システム (特に、異なる製造業者によって作成されたシステム) では、動作が異なる場合があります。Mobile Link 同期は、複数のデータベース・システムで動作可能です。詳細については、使用している統合サーバと ODBC ドライバのマニュアルを参照してください。

ダウンロード中の文字セット変換

1. 統合データベース・システムの ODBC ドライバは、文字を統合データベースのコードで受け取ります。このドライバは、これらの文字を Unicode に変換し、Unicode API をとおして Mobile Link サーバに渡します。この変換を制御できるのは、使用している統合データベース・システムの ODBC ドライバだけです。詳細については、使用している統合サーバと ODBC ドライバのマニュアルを参照してください。
2. Mobile Link サーバは、Unicode ODBC API をとおして文字を受け取ります。リモート・データベースで異なる文字セットを使用している場合は、ダウンロードする前に Mobile Link サーバが文字を変換します。

例

- Windows Mobile デバイス上の Ultra Light アプリケーションでは、Unicode 文字セットを使用します。

  Windows Mobile アプリケーションを同期するときは、Mobile Link サーバ内で文字変換は行われません。Mobile Link 同期サーバは、アプリケーションから送信されたデータがすでに Unicode であると判断し、それを ODBC ドライバに直接渡します。同様に、データのダウンロード時も文字セットの変換は必要ありません。

- Windows Mobile 以外のプラットフォーム上にあるすべての SQL Anywhere データベースとすべての Ultra Light アプリケーションは、リモート・データベースの照合順によって決定された文字セットを使用します。

  リモート・データベースを同期するときは、Mobile Link サーバがリモート・データベースの文字セットと Unicode の間の文字セット変換を実行します。

ODBC ドライバの文字セット変換の制御

統合データベースは Unicode を使用していない場合が多いので、使用している統合データベース・システムの ODBC ドライバがデータを Unicode に変換する方法と Unicode から変換する方法について理解しておくことが重要です。ODBC ドライバの中には、Mobile Link を実行しているコンピュータの言語設定を使って、使用する文字セットを決定するものがあります。この場合、Mobile Link サーバを実行しているコンピュータの言語設定とコード・ページ設定が、統合データベースの設定と一致するのが最も望ましいことです。

Sybase Adaptive Server Enterprise のドライバなどその他の ODBC ドライバでは、各接続で特定の文字セットを使用できます。変換のエラーを避けるために、Mobile Link で使用する文字セットが統合データベースの文字セットと一致するように設定してください。

使用している統合データベース・サーバの ODBC ドライバで文字セット変換がどのように行われるかについては、各製品の ODBC ドライバのマニュアルを参照してください。

Mobile Link 対応の iAnywhere Solutions ODBC ドライバ

目次

Mobile Link でサポートされる ODBC ドライバ

次の表のとおり、Mobile Link サーバは各種の統合データベースや ODBC ドライバと連携させることができます。ドライバによっては、Mobile Link と互換性があっても、使用に関する機能上の制約がある場合があります。

サポートされるバージョンの詳細については、http://www.ianywhere.jp/sas/os.html を参照してください。

データベース	ODBC ドライバ
SQL Anywhere 11	SQL Anywhere 11[1]
Oracle 10g または 11g	iAnywhere Solutions 11 - Oracle[1]
Microsoft SQL Server	Microsoft SQL Server ODBC ドライバ[2]
Sybase Adaptive Server Enterprise 12.5.1 以降	Sybase Adaptive Server Enterprise ドライバ[2]
Windows、Linux、UNIX 用の IBM DB2 LUW 8.1 または 8.2	IBM DB2 8.2 CLI ドライバ[2]
Windows、Linux、UNIX 用の IBM DB2 LUW 9.x	IBM DB2 9.1 CLI ドライバ[2]
Z/OS 用の IBM DB2 メインフレーム 8.1	IBM DB2 8.2 CLI ドライバ[2]
Z/OS 用の IBM DB2 メインフレーム 9.1	IBM DB2 9.1 CLI ドライバ[2]
MySQL 5.1	MySQL ODBC ドライバ 5.1[2]

[1] SQL Anywhere バージョン 11 に付属しています。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。

[2] SQL Anywhere バージョン 11 に付属していません。インストール方法と設定方法については、「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

iAnywhere Solutions 11 - Oracle ODBC ドライバは、iAnywhere ソフトウェアで使用するための専用に作成されたドライバです。このドライバは、サード・パーティ・ソフトウェアでは動作しません。

Mobile Link または OMNI で Oracle を使用する場合、この Oracle ドライバと同じコンピュータに Oracle クライアントをインストールする必要があります。

Oracle ドライバは、ODBC アドミニストレータ、.odbc.ini ファイル (UNIX)、または dbdsn ユーティリティを使用して設定できます。

次の表に、Oracle ドライバの設定オプションを示します。

Windows ODBC アドミニストレータ	dbdsn コマンド・ラインまたは .odbc.ini ファイルの設定	説明
[データ・ソース名]	dbdsn では、-w オプションを使用します。	データ・ソースを識別するための名前。
[ユーザ ID]	**UserID** dbdsn では、接続文字列でこのオプションを設定します。	アプリケーションが Oracle データベースへの接続に使用するデフォルトのログオン ID。このフィールドを空白にすると、接続したときに情報が要求されます。
[パスワード]	**Password** dbdsn では、接続文字列でこのオプションを設定します。	アプリケーションが Oracle データベースへの接続に使用するパスワード。このフィールドを空白にすると、接続したときに情報が要求されます。
[SID]	**SID**	Oracle インストール・ディレクトリの *network/admin/tnsnames.ora* に保存される TNS サービス名。
[Microsoft 分散トランザクションを有効にする]	dbdsn では、接続文字列で **enableMSDIC** オプションを使用します。 *.odbc.ini* では、サポートされていません。	トランザクションを Microsoft 分散トランザクション・コーディネータにエンリストする場合は、このチェックボックスをオンにします。オンにすると、Oracle ODBC ドライバに Oracle バイナリ・ファイル *oramts.dll* (Oracle 9i クライアントの場合) または *oramts10.dll* (Oracle 10g クライアントの場合) が必要になります。
[パスワードの暗号化]	dbdsn では、-pe オプションを使用します。 *.odbc.ini* では、サポートされていません。	パスワードを暗号化形式でデータ・ソースに保存するには、このチェックボックスをオンにします。

Windows ODBC アドミニストレータ	dbdsn コマンド・ラインまたは .odbc.ini ファイルの設定	説明
[プロシージャは結果を返す]	**ProcResults** dbdsn では、接続文字列でこのオプションを設定します。	ストアド・プロシージャが結果を返すことができる場合は、このフィールドを選択します。デフォルトでは、プロシージャは結果を返しません (選択されていません)。download_cursor スクリプトまたは download_delete_cursor スクリプトがストアド・プロシージャ呼び出しの場合は、このオプションを yes に設定してください。
[配列サイズ]	**ArraySize** dbdsn では、接続文字列でこのオプションを設定します。	ローのプリフェッチに使用するバイト配列のサイズ (バイト単位)。文ごとに指定します。デフォルトは 60000 です。この値を大きくすると、使用するメモリが増えますが、フェッチのパフォーマンス (Mobile Link サーバのダウンロードなど) は大幅に向上します。

Windows での設定

♦ **Windows で Oracle ドライバ用の DSN を作成するには、次の手順に従います。**

1. ODBC アドミニストレータを開きます。
   - **[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。
2. **[追加]** をクリックします。
3. **[iAnywhere Solutions 11 - Oracle]** を選択します。
4. 必要な設定オプションを指定します。フィールドは上記の説明のとおりです。
5. **[接続テスト]** をクリックし、**[OK]** をクリックします。

UNIX での設定

UNIX では、ODBC システム情報ファイル (通常、*.odbc.ini*) でドライバを設定している場合、このドライバのセクションは次のように表示されます (各フィールドに適切な値が入力されます)。

```
[sample_dsn_using_the_ias_odbc_driver_for_oracle]
Driver=full-path/libdboraodbc10_r.so
UserID=user-id
Password=password
SID=TNS-service-name
ProcResults=[yes|no]
ArraySize=bytes
```

各フィールドについては、前述の説明を参照してください。

DBDSN 設定

Oracle DSN を dbdsn ユーティリティを使用して作成するには、次の構文を使用します。

dbdsn -w *data-source-name* **-or -c** *configuration-options*

configuration-options については、前述の説明を参照してください。

次に例を示します。

```
dbdsn -w MyOracleDSN -or -pe -c
Userid=dba;Password=sql;SID=abcd;ArraySize=100000;ProcResults=y;enableMSDIC=n
```

「データ・ソース・ユーティリティ (dbdsn)」 『SQL Anywhere サーバ - データベース管理』を参照してください。

参照

- Mobile Link の推奨 ODBC ドライバ
- 「データ・ソース・ユーティリティ (dbdsn)」 『SQL Anywhere サーバ - データベース管理』

Mobile Link アプリケーションの配備

目次

Mobile Link 配備の概要

Mobile Link アプリケーションを配備するときは、以下の作業を行います。

- 運用設定への Mobile Link サーバの配備
- リダイレクタの配備 (必要に応じて)
- SQL Anywhere Mobile Link クライアントの配備
- Ultra Light Mobile Link クライアントの配備

この章では、アプリケーションのインストール・プログラムに含める必要があるファイルについて、これらの項目ごとに説明します。

Windows での配備には、**同期モデル展開ウィザード**を使用できます。「Deployment ウィザードの使用」『SQL Anywhere サーバ - プログラミング』を参照してください。

ライセンス契約の確認
ファイルの再配布はライセンス契約に従います。このマニュアル内の記述は、ライセンス契約のどの条項にも優先しません。配備について検討する前にライセンス契約を確認してください。

Mobile Link サーバの配備

Mobile Link サーバを運用環境に配備するには、SQL Anywhere のライセンス取得済みコピーを運用コンピュータにインストールするのが最も簡単です。

しかし、Mobile Link サーバを別のインストール・プログラムで再配布する場合は、ファイルのサブセットだけを含めてもかまいません。この場合、インストール環境に次のファイルを含める必要があります。

注意

- 再配布する前にクリーンなコンピュータでテストしてください。
- サンプル以外のファイルは、SQL Anywhere インストール・ディレクトリにインストールしてください。
- 特に指定がないかぎり、ファイルを同じディレクトリにインストールしてください。
- ロケーションが指定されている場合、ファイルは同じ名前のディレクトリにコピーしてください。
- UNIX の場合は、システムが SQL Anywhere アプリケーションとライブラリを見つけることができるように環境変数を設定してください。必要な環境変数を設定するためのテンプレートとして、*sa_config.sh* と *sa_config.csh* (32 ビットの環境では *install-dir/bin32* ディレクトリ内、64 ビットの環境では *install-dir/bin64* 内) のいずれかのうち、シェルに適したファイルを使用することをおすすめします。sa_config ファイルによって設定される環境変数には PATH、LD_LIBRARY_PATH、SQLANY11、SQLANYSAMP11 などがあります。
- Windows の場合は、システムが SQL Anywhere アプリケーションとライブラリを見つけることができるように環境変数を設定してください。PATH 変数に、32 ビットの環境の場合は *install-dir/bin32*、64 ビットの環境の場合は *install-dir/bin64* が含まれていることを確認します。両方のエントリが存在する場合は、環境に該当しないパスを削除します。
- Java 同期論理を使用し、グラフィカルな管理ツール (Sybase Central と Mobile Link モニタ) を使用するには、JRE 1.6.0 をインストールします。
- Sybase Central を配備する方法については、「管理ツールの配備」 『SQL Anywhere サーバ - プログラミング』を参照してください。
- Windows 用の配備ウィザードがあります。「Deployment ウィザードの使用」 『SQL Anywhere サーバ - プログラミング』を参照してください。

Windows の 32 ビット・アプリケーション

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。64 ビット Windows 環境のファイル構造の詳細については、「Windows の 64 ビット・アプリケーション」 852 ページを参照してください。

説明	Windows ファイル
Mobile Link サーバ	● *bin32¥mlodbc11.dll* ● *bin32¥mlsrv11.exe* ● *bin32¥mlsrv11.lic* ● *bin32¥mlsql11.dll* ● *bin32¥dbicu11.dll* ● *bin32¥dbicudt11.dll*
言語ライブラリ	● *bin32¥dblgen11.dll*[1]
同期ストリーム・ライブラリ (バージョン 8 と 9 のクライアントのサポートのため)	● *bin32¥mlhttp11.dll* ● *bin32¥mlsock11.dll*
Java 同期論理	● *java¥activation.jar*[2] ● *java¥imap.jar*[2] ● *java¥jodbc.jar* ● *java¥log4j.jar*[2] ● *java¥mailapi.jar*[2] ● *java¥mlscript.jar* ● *java¥mlsupport.jar* ● *java¥pop3.jar*[2] ● *java¥smtp.jar*[2] ● *bin32¥mljava11.dll* ● *bin32¥dbjodbc11.dll* ● *bin32¥mljodbc11.dll*
.NET 同期論理	● *MobiLink¥setup¥dnet¥mlDomConfig.xml* ● *bin32¥mldnet11.dll* ● *bin32¥dnetodbc11.dll* ● *Assembly¥v2¥iAnywhere.MobiLink.dll* ● *Assembly¥v2¥iAnywhere.MobiLink.Script.dll* ● *Assembly¥v2¥iAnywhere.MobiLink.Script.xml* ● *bin32¥mlDomConfig.xsd*
バージョン 10 と 11 クライアントのセキュリティ・オプション (mlsrv11 -x)[3]	● *bin32¥mlecc_tls11.dll* ● *bin32¥mlrsa_tls11.dll* ● *bin32¥mlrsa_tls_fips11.dll* ● *bin32¥sbgse2.dll*

説明	Windows ファイル
バージョン 8 と 9 クライアントのセキュリティ・オプション [3] (mlsrv11 -xo)[5]	● *bin32¥mlhttps11.dll* ● *bin32¥mlhttpsfips11.dll* ● *bin32¥mlrsafips11.dll* ● *bin32¥mljrsa11.dll* ● *bin32¥mljtls11.dll* ● *bin32¥mlrsa11.dll* ● *bin32¥mltls11.dll* ● *bin32¥defaultmem.dll* ● *bin32¥libsb.dll*
設定スクリプト (統合データベース用のファイルを配備)	● *MobiLink¥setup¥* ● *MobiLink¥upgrade¥*
mluser ユーティリティ	● *bin32¥mluser.exe* ● *bin32¥mlodbc11.dll* ● *bin32¥dbicu11.dll* ● *bin32¥dbicudt11.dll*
mlstop ユーティリティ	● *bin32¥mlstop.exe* ● *bin32¥dbicu11.dll*
Mobile Link モニタ	● *java¥mlmon.jar* ● *java¥JComponents1101.jar* ● *java¥mlstream.jar* ● *java¥jsyblib600.jar* ● *Sun¥JavaHelp-2_0¥jh.jar* ● *Sun¥jaxb1.0¥* ● *bin32¥jsyblib600.dll* ● *bin32¥mlmon.exe* Mobile Link モニタのセキュリティ用 [3] ● *bin32¥mlcecc11.dll* ● *bin32¥mlcrsa11.dll* ● *bin32¥mlcrsafips11.dll* ● *bin32¥mlczlib11.dll*
Mobile Link プラグインとモニタのオンライン・ヘルプ	● *¥documentation¥ja¥htmlhelp¥dbadmin_ja11.chm*[1] ● *¥documentation¥ja¥htmlhelp¥dbadmin_ja11.map*[1]
Mobile Link リダイレクタ	● *MobiLink¥redirector*

説明	Windows ファイル
Notifier	● *java¥activation.jar*[2] ● *java¥jodbc.jar* ● *java¥log4j.jar*[4] ● *java¥mailapi.jar*[2] ● *java¥mlnotif.jar* ● *java¥mlscript.jar* ● *java¥smtp.jar*[2] ● *bin32¥mljodbc11.dll* ● *bin32¥mljstrm11.dll*
QAnywhere で必要な Mobile Link サーバ・ファイル	● Notifier ファイル ● *java¥commons-logging.jar* ● *java¥commons-codec-1.3.jar* ● *java¥commons-httpclient-3.0.jar* ● *java¥jsyblib600.jar* ● *java¥log4j.jar*[4] ● *java¥mlscript.jar* ● *java¥mlstream.jar* ● *java¥qaconnector.jar* ● *bin32¥jsyblib600.dll*
Mobile Link リレー・サーバ Outbound Enabler	● *bin32¥rsoe.exe* Outbound Enabler のセキュリティ用 ● *bin32¥mlcrsa11.dll*

[1] ドイツ語、日本語、中国語の版では、en をそれぞれ de、ja、zh と置き換えます。

[2] アプリケーションを再配布する場合は、これらのファイルを Sun から直接入手してください。

[3] ECC および FIPS を使用するには、別途ライセンスが必要な SQL Anywhere セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。RSA セキュリティは、バージョン 10 以降の SQL Anywhere に付属しています。このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

[4] アプリケーションを再配布する場合は、このファイルを Apache から直接入手してください。

[5] *HKEY_LOCAL_MACHINE¥SOFTWARE¥Certicom¥libsb* というレジストリ・キーを作成し、expectedtag という名前の REG_BINARY 値を追加して、データを 5B0F4FA6E24AEF3B4407052EB04902711FD991B6 に設定する必要があります。

Windows の 64 ビット・アプリケーション

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。64 ビット Windows 環境のファイル構造の詳細については、「Windows の 32 ビット・アプリケーション」 849 ページを参照してください。

説明	Windows ファイル
Mobile Link サーバ	● *bin64¥mlodbc11.dll* ● *bin64¥mlsrv11.exe* ● *bin64¥mlsrv11.lic* ● *bin64¥mlsql11.dll* ● *bin64¥dbicu11.dll* ● *bin64¥dbicudt11.dll*
言語ライブラリ	● *bin64¥dblgen11.dll*[1]
同期ストリーム・ライブラリ (バージョン 8 と 9 のクライアントのサポートのため)	● *bin64¥mlhttp11.dll* ● *bin64¥mlsock11.dll*
Java 同期論理	● *java¥activation.jar*[2] ● *java¥imap.jar*[2] ● *java¥jodbc.jar* ● *java¥log4j.jar*[2] ● *java¥mailapi.jar*[2] ● *java¥mlscript.jar* ● *java¥mlsupport.jar* ● *java¥pop3.jar*[2] ● *java¥smtp.jar*[2] ● *bin64¥mljava11.dll* ● *bin64¥dbjodbc11.dll* ● *bin64¥mljodbc11.dll*
.NET 同期論理	● *MobiLink¥setup¥dnet¥mlDomConfig.xml* ● *bin64¥mldnet11.dll* ● *bin64¥dnetodbc11.dll* ● *Assembly¥v2¥iAnywhere.MobiLink.dll* ● *Assembly¥v2¥iAnywhere.MobiLink.Script.dll* ● *Assembly¥v2¥iAnywhere.MobiLink.Script.xml* ● *bin64¥mlDomConfig.xsd*
バージョン 10 と 11 クライアントのセキュリティ・オプション (mlsrv11 -x)[3]	● *bin64¥mlecc_tls11.dll* ● *bin64¥mlrsa_tls11.dll* ● *bin64¥mlrsa_tls_fips11.dll* ● *bin64¥sbgse2.dll*

説明	Windows ファイル
バージョン 8 と 9 クライアントのセキュリティ・オプション[3] (mlsrv11 -xo)[5]	● *bin64¥mlhttps11.dll* ● *bin64¥mlhttpsfips11.dll* ● *bin64¥mlrsafips11.dll* ● *bin64¥mljrsa11.dll* ● *bin64¥mljtls11.dll* ● *bin64¥mlrsa11.dll* ● *bin64¥mltls11.dll* ● *bin64¥defaultmem.dll* ● *bin64¥libsb.dll*
設定スクリプト (統合データベース用のファイルを配備)	● *MobiLink¥setup¥* ● *MobiLink¥upgrade¥*
mluser ユーティリティ	● *bin64¥mluser.exe* ● *bin64¥mlodbc11.dll* ● *bin64¥dbicu11.dll* ● *bin64¥dbicudt11.dll*
mlstop ユーティリティ	● *bin64¥mlstop.exe* ● *bin64¥dbicu11.dll*
Mobile Link モニタ	● *java¥mlmon.jar* ● *java¥JComponents1101.jar* ● *java¥mlstream.jar* ● *java¥jsyblib600.jar* ● *Sun¥JavaHelp-2_0¥jh.jar* ● *Sun¥jaxb1.0¥* ● *bin64¥jsyblib600.dll* ● *bin64¥mlmon.exe* Mobile Link モニタのセキュリティ用[3] ● *bin64¥mlcecc11.dll* ● *bin64¥mlcrsa11.dll* ● *bin64¥mlcrsafips11.dll* ● *bin64¥mlczlib11.dll*
Mobile Link プラグインとモニタのオンライン・ヘルプ	● *¥documentation¥ja¥htmlhelp¥dbadmin_ja11.chm*[1] ● *¥documentation¥ja¥htmlhelp¥dbadmin_ja11.map*[1]
Mobile Link リダイレクタ	● *MobiLink¥redirector*

説明	Windows ファイル
Notifier	● *java¥activation.jar*[2] ● *java¥jodbc.jar* ● *java¥log4j.jar*[4] ● *java¥mailapi.jar*[2] ● *java¥mlnotif.jar* ● *java¥mlscript.jar* ● *java¥smtp.jar*[2] ● *bin64¥mljodbc11.dll* ● *bin64¥mljstrm11.dll*
QAnywhere で必要な Mobile Link サーバ・ファイル	● Notifier ファイル ● *java¥commons-logging.jar* ● *java¥commons-codec-1.3.jar* ● *java¥commons-httpclient-3.0.jar* ● *java¥jsyblib600.jar* ● *java¥log4j.jar*[4] ● *java¥mlscript.jar* ● *java¥mlstream.jar* ● *java¥qaconnector.jar* ● *bin64¥jsyblib600.dll*
Mobile Link リレー・サーバ Outbound Enabler	● *bin64¥rsoe.exe* Outbound Enabler のセキュリティ用 ● *bin64¥mlcrsa11.dll*

[1] ドイツ語、日本語、中国語の版では、en をそれぞれ de、ja、zh と置き換えます。

[2] アプリケーションを再配布する場合は、これらのファイルを Sun から直接入手してください。

[3] ECC および FIPS を使用するには、別途ライセンスが必要な SQL Anywhere セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。RSA セキュリティは、バージョン 10 以降の SQL Anywhere に付属しています。このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

[4] アプリケーションを再配布する場合は、このファイルを Apache から直接入手してください。

[5] *HKEY_LOCAL_MACHINE¥SOFTWARE¥Certicom¥libsb* というレジストリ・キーを作成し、expectedtag という名前の REG_BINARY 値を追加して、データを 5B0F4FA6E24AEF3B4407052EB04902711FD991B6 に設定する必要があります。

UNIX の 32 ビット・アプリケーション (UNIX、Linux、Macintosh)

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。64 ビット UNIX 環境のファイル構造の詳細については、「UNIX の 64 ビット・アプリケーション (UNIX と Linux)」 858 ページを参照してください。

説明	UNIX ファイル
Mobile Link サーバ	● *bin32/mlsrv11* ● *bin32/mlsrv11.lic* ● *lib32/libdbodm11_r.so*[3] ● *lib32/libmlodbc11_r.so*[3] ● *lib32/libmlsql11_r.so*[3] ● *lib32/libdbtasks11_r.so*[3] ● *lib32/libdbicu11_r.so*[3] ● *lib32/libdbicudt11_r.so*[3] ● *lib32/libdbodbcinst11_r.so*[3]
言語ライブラリ	● *res/dblgen11.res*[1]
バージョン 8 と 9 クライアント用の同期ストリーム・ライブラリ (使用するライブラリを配備)	● *lib32/libmlhttp11_r.so*[3] ● *lib32/libmlsock11_r.so*[3]
Java 同期論理	● *java/activation.jar*[2] ● *java/imap.jar*[2] ● *java/jodbc.jar* ● *java/log4j.jar*[5] ● *java/mailapi.jar*[2] ● *java/mlscript.jar* ● *java/mlsupport.jar* ● *java/pop3.jar*[2] ● *java/smtp.jar*[2] ● *lib32/libmljava11_r.so*[3] ● *lib32/libmljodbc11.so*[3]
.NET 同期論理	● 該当なし
バージョン 10 と 11 クライアントのセキュリティ・オプション (mlsrv11 -x)[4]	● *lib32/libmlecc_tls11_r.so*[3] ● *lib32/libmlrsa_tls11_r.so*[3]
バージョン 8 と 9 クライアントのセキュリティ・オプション (mlsrv11 -xo)[4]	● *lib32/libmlhttps11_r.so*[3] ● *lib32/libmljrsa11_r.so*[3] ● *lib32/libmljtls11_r.so*[3] ● *lib32/libmlrsa11_r.so*[3] ● *lib32/libmltls11_r.so*[3]
設定スクリプト (統合データベース用のファイルを配備)	● *MobiLink/setup* ● *MobiLink/upgrade*

説明	UNIX ファイル
mluser ユーティリティ	● *bin32/mluser* ● *lib32/libmlodbc11_r.so*[3] ● *lib32/libdbicu11.so*[3] ● *lib32/libdbicudt11.so*[3]
mlstop ユーティリティ	● *bin32/mlstop* ● *lib32/libdbicu11.so*[3]
Mobile Link モニタ	● *bin32/mlmon* ● *java/mlmon.jar* ● *java/mlstream.jar* ● *lib32/libjsyblib600_r.so*[3] ● *sun/JavaHelp-2_0/jh.jar* ● *sun/jaxb1.0/* ● *java/JComponents1101.jar* ● *java/jsyblib600.jar*
Mobile Link リダイレクタ	● *Mobilink/redirector/redirector.config* ● *MobiLink/redirector/apache/* ● *MobiLink/redirector/java/* ● *MobiLink/redirector/MBusinessAnywhere/* ● *MobiLink/redirector/nsapi/*
Mobile Link プラグインと Mobile Link モニタのオンライン・ヘルプ	● *java/sqlanywhere_ja11.jar*[1]
Notifier	● *java/activation.jar*[2] ● *java/jodbc.jar* ● *java/log4j.jar*[2] ● *java/mailapi.jar*[2] ● *java/mlnotif.jar* ● *java/mlscript.jar* ● *java/smtp.jar*[2] ● *lib32/libmljstrm11_r.so*[3]
QAnywhere で必要な Mobile Link サーバ・ファイル	● Notifier ファイル ● *java/commons-codec-1.3.jar* ● *java/commons-httpclient-3.0.jar* ● *java/commons-logging.jar* ● *java/jsyblib600.jar* ● *java/log4j.jar*[5] ● *java/mlscript.jar* ● *java/mlstream.jar* ● *java/qaconnector.jar* ● *lib32/libjsyblib600_r.so*[3]

説明	UNIX ファイル
Mobile Link リレー・サーバ Outbound Enabler	● *bin32/rsoe* Outbound Enabler のセキュリティ用 ● *lib32/libmlcrsa11_r.so*[3]

[1] ドイツ語、日本語、中国語の版では、en をそれぞれ de、ja、zh と置き換えます。

[2] アプリケーションを再配布する場合は、これらのファイルを Sun から直接入手してください。

[3] Linux のファイル拡張子は *.so* です。Macintosh のファイル拡張子は *.dylib* です。

[4] トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスが必要な SQL Anywhere セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」 『SQL Anywhere 11 - 紹介』を参照してください。

[5] アプリケーションを再配布する場合は、これらのファイルを Apache から直接入手してください。

UNIX の 64 ビット・アプリケーション (UNIX と Linux)

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。32 ビット UNIX 環境のファイル構造の詳細については、「UNIX の 32 ビット・アプリケーション (UNIX、Linux、Macintosh)」 855 ページを参照してください。

説明	UNIX ファイル
Mobile Link サーバ	● *bin64/mlsrv11* ● *bin64/mlsrv11.lic* ● *lib64/libdbodm11_r.so*[3] ● *lib64/libmlodbc11_r.so*[3] ● *lib64/libmlsql11_r.so*[3] ● *lib64/libdbtasks11_r.so*[3] ● *lib64/libdbicu11_r.so*[3] ● *lib64/libdbicudt11_r.so*[3] ● *lib64/libdbodbcinst11_r.so*[3]
言語ライブラリ	● *res/dblgen11.res*[1]
バージョン 8 と 9 クライアント用の同期ストリーム・ライブラリ (使用するライブラリを配備)	● *lib64/libmlhttp11_r.so*[3] ● *lib64/libmlsock11_r.so*[3]

説明	UNIX ファイル
Java 同期論理	● *java/activation.jar*[2] ● *java/imap.jar*[2] ● *java/jodbc.jar* ● *java/log4j.jar*[5] ● *java/mailapi.jar*[2] ● *java/mlscript.jar* ● *java/mlsupport.jar* ● *java/pop3.jar*[2] ● *java/smtp.jar*[2] ● *lib64/libmljava11_r.so*[3] ● *lib64/libmljodbc11.so*[3]
.NET 同期論理	● 該当なし
バージョン 10 と 11 クライアントのセキュリティ・オプション (mlsrv11 -x)[4]	● *lib64/libmlecc_tls11_r.so*[3] ● *lib64/libmlrsa_tls11_r.so*[3]
バージョン 8 と 9 クライアントのセキュリティ・オプション (mlsrv11 -xo)[4]	● *lib64/libmlhttps11_r.so*[3] ● *lib64/libmljrsa11_r.so*[3] ● *lib64/libmljtls11_r.so*[3] ● *lib64/libmlrsa11_r.so*[3] ● *lib64/libmltls11_r.so*[3]
設定スクリプト (統合データベース用のファイルを配備)	● *MobiLink/setup* ● *MobiLink/upgrade*
mluser ユーティリティ	● *bin64/mluser* ● *lib64/libmlodbc11_r.so*[3] ● *lib64/libdbicu11.so*[3] ● *lib64/libdbicudt11.so*[3]
mlstop ユーティリティ	● *bin64/mlstop* ● *lib64/libdbicu11.so*[3]
Mobile Link モニタ	● *bin64/mlmon* ● *java/mlmon.jar* ● *java/mlstream.jar* ● *lib64/libjsyblib600_r.so*[3] ● *sun/JavaHelp-2_0/jh.jar* ● *sun/jaxb1.0* ● *java/JComponents1101.jar* ● *java/jsyblib600.jar*

説明	UNIX ファイル
Mobile Link リダイレクタ	● *Mobilink/redirector/redirector.config* ● *MobiLink/redirector/apache/* ● *MobiLink/redirector/java/* ● *MobiLink/redirector/MBusinessAnywhere/* ● *MobiLink/redirector/nsapi/*
Mobile Link プラグインと Mobile Link モニタのオンライン・ヘルプ	● *java/sqlanywhere_en11.jar*[1]
Notifier	● *java/activation.jar*[2] ● *java/jodbc.jar* ● *java/log4j.jar*[2] ● *java/mailapi.jar*[2] ● *java/mlnotif.jar* ● *java/mlscript.jar* ● *java/smtp.jar*[2]
QAnywhere で必要な Mobile Link サーバ・ファイル	● Notifier ファイル ● *java/commons-codec-1.3.jar* ● *java/commons-httpclient-3.0.jar* ● *java/commons-logging.jar* ● *java/jsyblib600.jar* ● *java/log4j.jar*[5] ● *java/mlscript.jar* ● *java/mlstream.jar* ● *java/qaconnector.jar* ● *lib64/libjsyblib600_r.so*[3]
Mobile Link リレー・サーバ Outbound Enabler	● *bin64/rsoe* Outbound Enabler のセキュリティ用 ● *lib64/libmlcrsa11_r.so*[3]

[1] ドイツ語、日本語、中国語の版では、en をそれぞれ de、ja、zh と置き換えます。

[2] アプリケーションを再配布する場合は、これらのファイルを Sun から直接入手してください。

[3] Solaris SPARC および Linux のファイル拡張子は *.so* です。AIX のファイル拡張子は *.a* です。

[4] トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスが必要な SQL Anywhere セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」『SQL Anywhere 11 - 紹介』を参照してください。

[5] アプリケーションを再配布する場合は、これらのファイルを Apache から直接入手してください。

SQL Anywhere Mobile Link クライアントの配備

注意

- SQL Anywhere クライアントの場合、SQL Anywhere データベース・サーバと Mobile Link クライアントを配備する必要があります。

  「データベースとアプリケーションの配備」 『SQL Anywhere サーバ - プログラミング』を参照してください。
- Mobile Link 同期クライアントを再配布する場合は、SQL Anywhere データベースに必要なファイルのほかに、以下のファイルをインストール環境に含める必要があります。
- 以下のファイルを配備するときは、特に指定がないかぎり、ファイルを同じディレクトリ構造に配置してください。
- Sybase Central を配備する方法については、「管理ツールの配備」 『SQL Anywhere サーバ - プログラミング』を参照してください。
- Windows 用の配備ウィザードがあります。「Deployment ウィザードの使用」 『SQL Anywhere サーバ - プログラミング』を参照してください。
- Windows Mobile の配備環境の場合、以下で *bin32* ディレクトリにあると示されているファイルは、*ce¥arm.50* ディレクトリにあります。.NET アセンブリは、*ce¥Assembly¥v2* ディレクトリに置かれます。

Windows アプリケーション

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。

説明	Windows ファイル
Mobile Link 同期クライアント (dbmlsync)	● *bin32¥dbcon11.dll*[2] ● *bin32¥dbicu11.dll*[3] ● *bin32¥dblgen11.dll*[1] ● *bin32¥dblib11.dll* ● *bin32¥dbmlsync.exe* ● *bin32¥dbtool11.dll*[2]
dbmlsync 統合コンポーネント (旧式)	● Mobile Link 同期クライアント・ファイル ● ビジュアル・コンポーネント： *bin32¥dbmlsynccomg.dll* ● 非ビジュアル・コンポーネント： *bin32¥dbmlsynccom.dll*
セキュリティ・オプション [2]	● *bin32¥mlcecc11.dll* ● *bin32¥mlcrsa11.dll* ● *bin32¥mlcrsafips11.dll* ● *bin32¥sbgse2.dll*

説明	Windows ファイル
ActiveSync ユーティリティと HotSync ユーティリティ	● *bin32¥mlasinst.exe* ● *bin32¥mlasdesk.dll* ● *bin32¥dbcon11.exe* ● *ce¥chip¥mlasdev.dll* (*chip* は、arm.50 などのサポートされる任意のチップ)
Listener	● *bin32¥dblgen11.dll*[1] ● *bin32¥dblsn.exe* ● *bin32¥lsn_udp.dll* ● *bin32¥lsn_swi510.dll* ● *bin32¥maac555.dll* ● *bin32¥maac750.dll* ● *bin32¥maac750r3.dll* ● *bin32¥mabridge.dll*

[1] ドイツ語、日本語、中国語の版では、en をそれぞれ de、ja、zh と置き換えます。

[2] Windows Mobile では dbtools インタフェースを使用しないかぎり不要です。

[3] dbinit -zn UTF8BIN を使用してデータベースを初期化する場合は不要です。「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。

UNIX のアプリケーション (UNIX、Linux、Macintosh)

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。

説明	UNIX ファイル
Mobile Link 同期クライアント	● *bin32/dbmlsync* ● *res/dblgen11.res* ● *lib32/libdbcon11_r.so*[1] ● *lib32/libdbicu11_r.so*[1] ● *lib32/libdblib11_r.so*[1] ● *lib32/libdbtool11_r.so*[1]
セキュリティ・オプション [2]	● *lib32/libmlcecc11_r.so*[1] ● *lib32/libmlcrsa11_r.so*[1]

[1] Linux のファイル拡張子は *.so* です。Macintosh のファイル拡張子は *.dylib* です。

[2] トランスポート・レイヤ・セキュリティを使用するには、別途ライセンスが必要な SQL Anywhere セキュリティ・オプションを入手する必要があります。このセキュリティ・オプションは、輸出規制対象品目です。このコンポーネントの注文方法については、「別途ライセンスが必要なコンポーネント」『SQL Anywhere 11 - 紹介』を参照してください。

Ultra Light Mobile Link クライアントの配備

Ultra Light クライアントの場合、Ultra Light ランタイム・ライブラリまたは Ultra Light コンポーネントには、必要な同期ストリーム機能が含まれています。Ultra Light ランタイム・ライブラリは、アプリケーションにコンパイルされます。配備はライセンス契約に応じて決まります。

参照

- 「Ultra Light のデバイスへの配備」『Ultra Light データベース管理とリファレンス』
- C/C++：「Palm アプリケーションの配備」『Ultra Light - C/C++ プログラミング』と「Windows Mobile アプリケーションの配備」『Ultra Light - C/C++ プログラミング』
- M-Business Anywhere：「Ultra Light for M-Business Anywhere アプリケーションの配備」『Ultra Light - M-Business Anywhere プログラミング』
- .NET：「レッスン 5：アプリケーションのビルドと配置」『Ultra Light - .NET プログラミング』

QAnywhere アプリケーションの配備

QAnywhere は、SQL Anywhere メッセージ・ストアをサポートしている C++ 、Java、.NET API を提供します。Java と .NET API は、Ultra Light メッセージ・ストアもサポートしています。QAnywhere アプリケーションを配備するのに必要なファイルは、Windows 環境、メッセージ・ストアのタイプ、API の選択によって異なります。モバイル Web サービス・アプリケーションを開発している場合、追加のファイルが必要になります。

以下に示すファイルのほかに、QAnywhere アプリケーションでは次のファイルも必要です。

- 「SQL Anywhere Mobile Link クライアントの配備」 861 ページの Mobile Link 同期クライアント、リスナ、セキュリティ (任意) に関する各項に記載されたすべてのファイル。リスナ・ファイルは、Push 通知を使用している場合 (デフォルト設定) のみ必要です。
- 「データベース・サーバの配備」 『SQL Anywhere サーバ - プログラミング』に記載された dbeng11 ファイルまたは dbsrv11 ファイル。

Sybase Central を配備する方法については、「管理ツールの配備」 『SQL Anywhere サーバ - プログラミング』を参照してください。

Windows アプリケーション

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。

Windows Mobile 環境のファイル構造の詳細については、「Windows Mobile アプリケーション」 866 ページを参照してください。

次に、SQL Anywhere メッセージ・ストアをセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows ファイル
C++	● *bin32¥qany11.dll* ● *bin32¥qaagent.exe* ● *bin32¥qastop.exe*
Java	● *bin32¥qaagent.exe* ● *bin32¥qastop.exe* ● *java¥qaclient.jar* ● *java¥jodbc.jar* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*

クライアント API	Windows ファイル
.NET	● *bin32¥qazlib.dll* ● *bin32¥qaagent.exe* ● *bin32¥qastop.exe* ● *assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *assembly¥v2¥iAnywhere.Data.SQLAnywhere.dll* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

次に、QAnywhere Agent を使用した配備環境で Ultra Light メッセージ・ストアをセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows ファイル
Java	● *bin32¥qauagent.exe* ● *bin32¥qastop.exe* ● *bin32¥qadbiuljni.dll* ● *java¥qaclient.jar* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*
.NET	● *bin32¥qazlib.dll* ● *bin32¥qauagent.exe* ● *bin32¥qastop.exe* ● *assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ultralite¥ultralite.NET¥assembly¥v2¥iAnywhere.Data.UltraLite.dll* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

Ultra Light メッセージ・ストアを作成する場合は、Ultra Light データベース作成ユーティリティを使用して udb データベース・ファイルを作成し、次に QAnywhere Ultra Light Agent の -si オプションを使用してデータベースを初期化してください。「Ultra Light データベース作成ユーティリティ (ulcreate)」『Ultra Light データベース管理とリファレンス』と「qauagent ユーティリティ」『QAnywhere』を参照してください。

次に、QAnywhere スタンドアロン・クライアントによる配備環境をセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows ファイル
Java	● *java¥qastandaloneclient.jar* ● *bin32¥qadbiulsjni.dll*
.NET	● *assembly¥v2¥iAnywhere.QAnywhere.StandAloneClient.dll* ● *assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ultralite¥ultralite.NET¥assembly¥v2¥iAnywhere.Data.UltraLite.dll*

Windows Mobile アプリケーション

すべてのディレクトリは、*install-dir* を基準とした相対ディレクトリです。

Windows 環境のファイル構造の詳細については、「Windows アプリケーション」 864 ページを参照してください。

次に、SQL Anywhere メッセージ・ストアをセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows Mobile ファイル
C++	● *ce¥arm.50¥qany11.dll* ● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe*
Java	● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe* ● *java¥qaclient.jar* ● *java¥jodbc.jar* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*

クライアント API	Windows Mobile ファイル
.NET	● *ce¥arm.50¥qazlib.dll* ● *ce¥arm.50¥qaagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ce¥assembly¥v2¥iAnywhere.Data.SQLAnywhere.dll* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *ce¥Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

次に、QAnywhere Agent を使用した配備環境で Ultra Light メッセージ・ストアをセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows Mobile ファイル
Java	● *ce¥arm.50¥qauagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥arm.50¥qadbiuljni.dll* ● *java¥qaclient.jar* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *java¥iawsrt.jar* ● *java¥jaxrpc.jar*
.NET	● *ce¥arm.50¥qazlib.dll* ● *ce¥arm.50¥qauagent.exe* ● *ce¥arm.50¥qastop.exe* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Client.dll* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ultralite¥ultralite.NET¥ce¥assembly¥v2¥iAnywhere.Data.UltraLite.dll* モバイル Web サービス・アプリケーションの場合、次のファイルも必要です。 ● *ce¥Assembly¥v2¥iAnywhere.QAnywhere.WS.dll*

Ultra Light メッセージ・ストアを作成する場合は、Ultra Light データベース作成ユーティリティを使用して udb データベース・ファイルを作成し、次に QAnywhere Ultra Light Agent の -si オプションを使用してデータベースを初期化してください。「Ultra Light データベース作成ユーティリティ (ulcreate)」『Ultra Light データベース管理とリファレンス』と「qauagent ユーティリティ」『QAnywhere』を参照してください。

次に、QAnywhere スタンドアロン・クライアントによる配備環境をセットアップする際に必要なファイルのリストを示します。

クライアント API	Windows Mobile ファイル
Java	● *java¥qastandaloneclient.jar* ● *ce¥arm.50¥qadbiulsjni.dll*
.NET	● *ce¥assembly¥v2¥iAnywhere.QAnywhere.StandAloneClient.dll* ● *ce¥assembly¥v2¥iAnywhere.QAnywhere.Resources.dll* ● *ultralite¥ultralite.NET¥ce¥assembly¥v2¥iAnywhere.Data.UltraLite.dll*

QAnywhere .NET API DLL の登録

QAnywhere .NET API DLL (*Assembly¥v2¥iAnywhere.QAnywhere.Client.dll*) は、Windows (Windows Mobile を除く) のグローバル・アセンブリ・キャッシュに登録する必要があります 。グローバル・アセンブリ・キャッシュには、コンピュータに登録されているすべてのプログラムがリストされています。SQL Anywhere をインストールすると、インストール・プログラムによって登録されます。Windows Mobile の場合、この DLL を登録する必要はありません。

QAnywhere を配備する場合は、.NET Framework に含まれている gacutil ユーティリティを使用して、QAnywhere .NET API DLL (*Assembly¥v2¥iAnywhere.QAnywhere.Client.dll*) を登録してください。

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」 876 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」 881 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」 885 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)」 885 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」 893 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」 873 ページ
- 「jConnect」 873 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」 873 ページ
- 「iAnywhere JDBC ドライバ」 872 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」 873 ページ
- 「iAnywhere JDBC ドライバ」 872 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」 881 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」 876 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」 900 ページ
- 「同期」 900 ページ
- 「Ultra Light」 877 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」 881 ページ
- 「Listener」 873 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」 875 ページ
- 「サーバ起動同期」 881 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」 881 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」 871 ページ。

Replication Agent

参照：「LTM」 873 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」 873 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」 883 ページ
- 「SQL」 876 ページ
- 「データベース管理システム (DBMS)」 885 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」 877 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」 893 ページ
- 「パブリケーション」 888 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」 887 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」 880 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」 901 ページ
- 「コード・ページ」 881 ページ
- 「照合」 898 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」 877 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」 880 ページ
- 「カーソル位置」 879 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」 879 ページ
- 「カーソル結果セット」 880 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」 879 ページ
- 「カーソル位置」 879 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」 876 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」 886 ページ
- 「ローカル・テンポラリ・テーブル」 894 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」 881 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」 901 ページ
- 「エンコード」 879 ページ
- 「照合」 898 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」 888 ページ
- 「Mobile Link ユーザ」 874 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」 882 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」 882 ページ
- 「生成されたジョイン条件」 899 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」 878 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」 901 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」 900 ページ
- 「SQL ベースの同期」 877 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」 895 ページ
- 「プライマリ・キー」 890 ページ
- 「データベース管理システム (DBMS)」 885 ページ
- 「リレーショナル・データベース管理システム (RDBMS)」 893 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」 871 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)」 893 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)」 885 ページ
- 「SYS」 877 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」 885 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」 886 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」 881 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」 894 ページ
- 「グローバル・テンポラリ・テーブル」 880 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」 886 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」 887 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」 894 ページ
- 「文レベルのトリガ」 901 ページ
- 「整合性」 898 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」 888 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」 893 ページ
- 「アーティクル」 878 ページ
- 「パブリケーションの更新」 888 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」 893 ページ
- 「パブリケーション」 888 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」 893 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」 898 ページ
- 「ユーザ定義データ型」 892 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」 889 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」 895 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」 898 ページ
- 「検査制約」 897 ページ
- 「外部キー制約」 896 ページ
- 「一意性制約」 895 ページ
- 「整合性」 898 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」 877 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」 878 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」 891 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」 886 ページ
- 「ビュー」 889 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」 881 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」 891 ページ
- 「ビュー」 889 ページ

ミラー・ログ

参照：「トランザクション・ログ・ミラー」 887 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」 883 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」 893 ページ
- 「FILE」 872 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」 880 ページ
- 「サーバ・メッセージ・ストア」 881 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」 893 ページ
- 「統合データベース」 900 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」 886 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」 881 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」 900 ページ
- 「統合データベース」 900 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)」 885 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」 893 ページ
- 「パブリケーションの更新」 888 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」 893 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」 887 ページ
- 「文レベルのトリガ」 901 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」 886 ページ
- 「グローバル・テンポラリ・テーブル」 880 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」 887 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」 895 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」 887 ページ
- 「トランザクション・ログ・ミラー」 887 ページ
- 「フル・バックアップ」 890 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」 901 ページ
- 「同時性 (同時実行性)」 901 ページ
- 「整合性」 898 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」 896 ページ
- 「プライマリ・キー制約」 890 ページ
- 「制約」 898 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」 890 ページ
- 「外部テーブル」 896 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」 898 ページ
- 「検査制約」 897 ページ
- 「プライマリ・キー制約」 890 ページ
- 「一意性制約」 895 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」 882 ページ
- 「内部ジョイン」 901 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」 895 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」 898 ページ
- 「外部キー制約」 896 ページ
- 「プライマリ・キー制約」 890 ページ
- 「一意性制約」 895 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」 895 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」 890 ページ
- 「外部キー」 895 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」 890 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」 901 ページ
- 「コード・ページ」 881 ページ
- 「エンコード」 879 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」 889 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」 897 ページ
- 「外部キー制約」 896 ページ
- 「プライマリ・キー制約」 890 ページ
- 「一意性制約」 895 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」 897 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」 882 ページ
- 「ジョイン条件」 883 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」 881 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」 893 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」 900 ページ
- 「レプリケーション」 893 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」 873 ページ
- 「SQL Remote」 876 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」 887 ページ
- 「独立性レベル」 901 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」 883 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」 882 ページ
- 「外部ジョイン」 896 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」 887 ページ
- 「ロー・レベルのトリガ」 894 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」 881 ページ
- 「エンコード」 879 ページ
- 「照合」 898 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

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

ふ

へ

ゆ

よ

り

れ

ろ

わ

