SYBASE iAnywhere

Ultra Light - M-Business Anywhere プログラミング

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。

M-Business Anywhere は、Web ベースのモバイル・アプリケーションを開発、配備するための iAnywhere プラットフォームです。以前の名称は、AvantGo M-Business Server でした。

対象読者

このマニュアルは、Ultra Light リレーショナル・データベースのパフォーマンス、リソース効率、堅牢性、セキュリティを利用してデータを格納、同期することを目的とするアプリケーション開発者を対象にしています。

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。

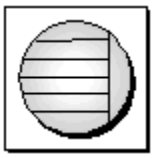
- **環境変数**　マニュアルでは、環境変数設定が引用されます。Windows のシェルでは、環境変数は構文 *%ENVVAR%* を使用して指定されます。UNIX のシェルでは、環境変数は構文 *$ENVVAR* または *${ENVVAR}* を使用して指定されます。

グラフィック・アイコン

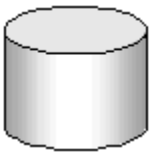
このマニュアルでは、次のアイコンを使用します。

- 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。

Ultra Light for M-Business Anywhere の概要

目次

Ultra Light for M-Business Anywhere の特徴

Ultra Light for M-Business Anywhere は、モバイル・デバイスのためのリレーショナル・データ管理システムです。ビジネス・アプリケーションに必要なパフォーマンス、リソース効率、堅牢性、セキュリティを備えています。Ultra Light では、エンタープライズ・データ・ストアとの同期も提供されます。

システムの稼働条件とサポートされるプラットフォーム

開発プラットフォーム

Ultra Light for M-Business Anywhere を使用してアプリケーションを開発するには、以下が必要です。

- M-Business Anywhere は、AvantGo M-Business Server の新しい名前です。このソフトウェアには、M-Business Server 5.3 以降と、該当する M-Business Anywhere クライアントが必要です。

ターゲット・プラットフォーム

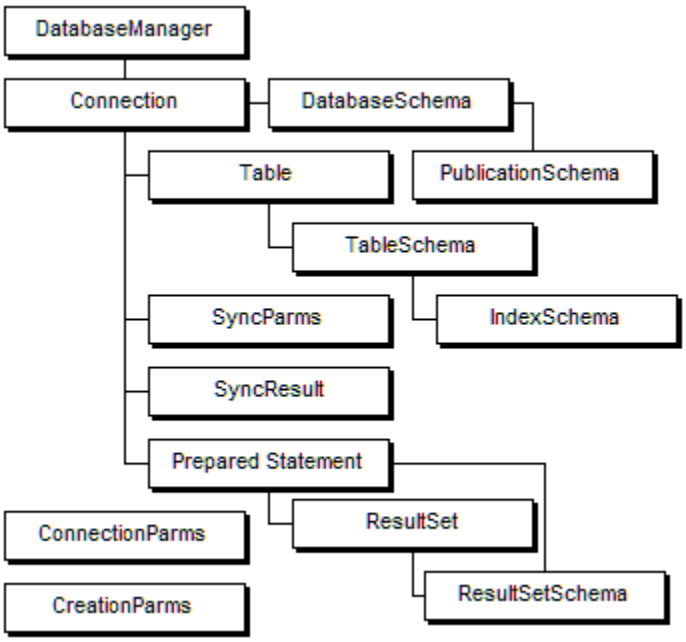
Ultra Light for M-Business Anywhere は、次のターゲット・プラットフォームをサポートしています。

- ARM プロセッサの Pocket PC に搭載した Windows Mobile 3.0 以降 (Windows Mobile 5.0 を含む)
- Palm OS バージョン 5.0 以降
- Windows では、M-Business Anywhere 5.5 以降

配備の詳細については、http://www.ianywhere.jp/developers/technotes/os_components_1101.html を参照してください。

Ultra Light for M-Business Anywhere のアーキテクチャ

Ultra Light プログラミング・インタフェースは、Ultra Light データベースを使用したデータ操作のためのオブジェクト・セットを公開しています。次の図は、オブジェクト階層を示します。



次のリストは、よく使用される高度なオブジェクトの一部を示します。

- **DatabaseManager**　Ultra Light データベースへの接続を管理します。「DatabaseManager クラス」 75 ページを参照してください。
- **ConnectionParms**　接続パラメータのセットを格納します。「ConnectionParms クラス」 71 ページを参照してください。
- **CreationParms**　データベース作成パラメータのセットを格納します。「CreationParms クラス」 73 ページを参照してください。
- **Connection**　データベース接続を表し、トランザクションを管理します。「Connection クラス」 56 ページを参照してください。
- **PreparedStatement、ResultSet、および ResultSetSchema**　SQL を使用してデータベース要求とその結果を管理します。次の項を参照してください。
  - 「PreparedStatement クラス」 87 ページ
  - 「ResultSet クラス」 97 ページ
  - 「ResultSetSchema クラス」 114 ページ
- **Table**　テーブル・ベースの API を使用してデータを管理します。「ULTable クラス」 156 ページを参照してください。

- **SyncParms と SyncResult**　Mobile Link サーバを介して同期を管理します。

  Mobile Link との同期の詳細については、「Ultra Light クライアント」『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。

Ultra Light for M-Business Anywhere 開発の概要

目次

Ultra Light for M-Business Anywhere クイック・スタート

次の手順は、付属するサンプル・アプリケーション CustDB と Simple を実行する方法を示します。

始める前に、M-Business Anywhere 6.0 以降がインストールされて実行されていることと、サーバで管理者の権限を持っていることを確認してください。また、サポートされているハンドヘルド・デバイスが必要です。

♦ **M-Business Anywhere のサンプルをインストールして実行するには、次の手順に従います。**

1. Ultra Light for M-Business Anywhere サンプル・ファイルを配備するためにインストール・ディレクトリにコピーします。
   a. コマンド・プロンプトを開き、SQL Anywhere インストール環境の *samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB* サブディレクトリに移動します。
   b. 次のコマンドを実行します。

      ```
      build.bat deploy-dir
      ```

      ここで *deploy-dir* は、CustDB ファイルと Ultra Light ファイルを配備するディレクトリです。たとえば、*C:¥tutorial¥mba* を選択します。

      このバッチ・ファイルは、必要なファイルを指定したロケーションにコピーします。コピーされるファイルを確認するには、テキスト・エディタで *samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB¥build.bat* ファイルを開きます。
2. Web サーバで仮想ディレクトリを作成します。この仮想ディレクトリは、手順 1 で指定したディレクトリ *deploy-dir* を指すようにします。次に示すのは、Microsoft IIS の場合の手順です。
   a. IIS の管理ツールを開きます。
   b. Web サイトを右クリックし、**[新規作成]** - **[仮想ディレクトリ]** を選択します。この仮想ディレクトリに **CustDB** という名前を付け、コンテンツ・ディレクトリとして配備ディレクトリ *deploy-dir* を指定します。他の設定は、デフォルト値のままにします。
   c. 作成した仮想ディレクトリを右クリックして、**[プロパティ]** を選択します。**[HTTP ヘッダー]** タブで **[ファイルの種類]** をクリックし、次のファイル拡張子をタイプ application/octet-stream として登録します。
      - Windows と Windows Mobile の場合：cab、dll
      - Palm OS の場合：pdb、prc
      - udb
   d. この仮想ディレクトリにあるファイル *main.htm* にアクセスするための URL をメモしておきます。デフォルトのインストール環境では、*http://localhost/CustDB/main.htm* です。
3. M-Business Anywhere にユーザを追加します。

   新しいユーザを M-Business Anywhere に追加するには、新しいユーザ・プロファイルの作成、ユーザによる自己登録の許可、CSV ファイルのインポート、という 3 つの方法があります。

ここでは、新しいユーザ・プロファイルを作成する方法について説明します。詳細については、M-Business Anywhere のマニュアルを参照してください。

a. 管理者として M-Business Anywhere にログインします。

   デフォルトの管理者アカウントの設定は、ユーザ ID が **Admin** で、パスワードは空です。

b. 左ウィンドウ枠で **[Users]** をクリックします。

c. **[Create User]** をクリックします。

d. **[User Name]** フィールドにユニークなユーザ名を入力します。

e. **[Password]** フィールドと **[Confirm Password]** フィールドに、パスワードを入力します。

f. **[保存]** をクリックします。

4. M-Business Anywhere クライアントをハンドヘルド・デバイスまたは PC に配備します。

   a. M-Business Anywhere ログイン・ページの **[Download Client Software Only]** リンクをクリックします。インストール・プログラムを実行して、クライアントをインストールします。

   b. ハンドヘルド・デバイスまたは PC で、M-Business Anywhere Server と同期するように M-Business Connect を設定します。

      作成した新しい M-Business ユーザ・アカウントのユーザ ID とパスワードを入力します。

   c. M-Business Anywhere Server と同期させます。

      このときに接続の問題が発生した場合は、ホスト名ではなく IP アドレスを使用してホストを指定します (ActiveSync の一部バージョンで発生する名前解決の問題を回避するため)。

   詳細については、M-Business Anywhere のマニュアルを参照してください。

5. M-Business Anywhere にグループを追加します。

   このグループは、Ultra Light for M-Business Anywhere をテストするために使用します。

   a. Web ブラウザから M-Business Anywhere に接続します。

      デフォルト URL は、*http://localhost* または *http://localhost:8091* です。

   b. 管理者アカウントを使用してログインします。

   c. 左のナビゲーション・ウィンドウ枠で、**[Groups] - [Create Group]** をクリックします。

   d. グループに **UltraLite Samples** という名前を付けます。

6. M-Business Anywhere チャネルを設定します。

   a. 左ウィンドウ枠の **[Edit Group]** で [Users] オプションを使用して、手順 3 で作成したユーザをグループ UltraLite Samples に追加します。

   b. 左のナビゲーション・ウィンドウ枠でグループの [Channels] オプションを使用し、次のチャネルを作成します。

設定	**値**
[Title]	CustDB

設定	値
[Location]	*http://localhost/CustDB/main.htm* または手順 2 でメモした URL
[Channel Size Limit]	1000
[Link depth]	3
[Allow binary distribution]	チェックする
[Hide From Users]	チェックしない

[Location] の値を設定したら、**[View]** をクリックして、入力した値が正しいことを確認してください。

7. クライアントを同期します。

   最初の同期では、Ultra Light for M-Business Anywhere ファイルがハンドヘルド・デバイスにダウンロードされます。

♦ 設定を確認するには、次の手順に従います。

1. 必要なファイルが存在することをチェックします。
   - Windows Mobile の場合は、デバイスを同期したら、次のファイルが *¥Program Files¥AvantGo ¥Pods* フォルダに存在することをチェックします。
     - ○ *ulpod11.dll*
     - ○ *custdb.udb*

     いずれかのファイルが存在しない場合は、そのファイルを手動でデバイスにコピーする必要がある場合があります。
   - Palm OS の場合は、デバイスを同期したら、次の項目が存在することを Palm OS のアプリケーション情報でチェックします。
     - ○ ulpod
     - ○ custdb

     いずれかの項目が存在しない場合は、Palm のインストール・ユーティリティを使用して、Ultra Light M-Business Anywhere のランタイム *.prc* ファイルとサンプル・スキーマ *.pdb* ファイルをデバイスにインストールする必要がある場合があります。
   - Windows デスクトップの場合は、デバイスを同期したら、次のファイルが *AvantGo Connect* フォルダの *AvantGo¥Pods* サブディレクトリに存在することをチェックします。
     - ○ *ulpod11.dll*
     - ○ *custdb.udb*

     いずれかのファイルが存在しない場合は、そのファイルを手動でデバイスにコピーする必要がある場合があります。
2. M-Business Client を起動します。

ハンドヘルド・デバイスまたは PC で、[About] 画面に Ultra Light for M-Business Anywhere のバージョン番号が表示されることをチェックします。これにより、Ultra Light for M-Business Anywhere が正常にインストールされたことが確認できます。

3. CustDB サンプル・アプリケーションを実行します。

   a. デスクトップ・コンピュータで、Mobile Link サーバを起動します。

      [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。

   b. M-Business Client で CustDB アプリケーションを起動します。

      CustDB アプリケーションは、M-Business ホームページ上のリンクです。

   c. ユーザ ID を入力します。

      デフォルト値は **50** です。

   d. 同期を実行します。

      [Do you have a network connection now?] というプロンプトに対して [はい] と応答するか、CustDB アプリケーションで **[同期]** をクリックします。これにより Mobile Link とデータが同期されますが、この操作は M-Business Anywhere との同期とは独立しています。

      [CustDB] のフィールドにデータが表示されます。これで、CustDB アプリケーションを利用できるようになりました。

「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。

複数データベースとの HotSync

Palm OS デバイス上の各 Ultra Light データベースには、HotSync で適切に処理されるように、固有の作成者 ID が必要です。また、その作成者 ID が設定されたアプリケーションが Palm OS デバイス上に存在する必要があります。

HotSync マネージャは、各アプリケーションの作成者 ID と識別子を使用して同期を処理します。同期を実行するために、適切に設定された各 Ultra Light アプリケーションは、Mobile Link コンジットに渡されます。このコンジットは、アプリケーションと同じ作成者 ID が設定されたデータベースを検索して同期を実行します。

コンジットの設定については、「HotSync 同期の概要」『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。

すべての Ultra Light for M-Business Anywhere アプリケーションは、M-Business Client の作成者 ID である **AvGo** を継承します。この継承により、作成者 ID が **AvGo** である Ultra Light データベース 1 つだけが同期でき、異なる作成者 ID をデータベースに割り当てた場合は、対応する作成者 ID が設定されたアプリケーションが存在しないために、HotSync でそのデータベースを見つけられない、ということになります。

この制限は、2 つのサンプル・アプリケーション (CustDB と Simple) では問題になりません。これは、共通のデータベース・スキーマを共有しているためです。ただし、CustDB データベースの同期時に、Simple サンプルも同期されてしまうという影響はあります。

この問題の解決については、「Palm 作成者 ID の登録」『Ultra Light - C/C++ プログラミング』を参照してください。

Ultra Light データベースへの接続

データベースのデータを操作するには、Ultra Light アプリケーションをデータベースに接続する必要があります。

接続を確立する最も簡単な方法は、次のとおりです。次の項では、この方法の応用について説明します。

```
var DatabaseMgr;
var Connection;
DatabaseMgr = CreateObject("iAnywhere.UltraLite.DatabaseManager.CustDB");
Connection = DatabaseMgr.openConnection("dbf=" + DatabaseMgr.directory + "¥¥mydb.udb");
```

Connection オブジェクトの使用

次に示す Connection オブジェクトのプロパティは、アプリケーションのグローバルな動作を管理します。

Connection オブジェクトの詳細については、「Connection クラス」 56 ページを参照してください。

- **コミット動作** デフォルトでは、Ultra Light アプリケーションは autoCommit モードに設定されています。insert 文、update 文、delete 文はすべて、すぐにデータベースにコミットされます。Connection.autoCommit を false に設定し、アプリケーションにトランザクションを構築します。autoCommit を off に設定しコミットを実行すると、アプリケーションのパフォーマンスを直接向上できます。「commit メソッド」 58 ページを参照してください。
- **ユーザ認証** grantConnectTo メソッドと revokeConnectFrom メソッドを使用すると、アプリケーションのユーザ ID とパスワードをデフォルト値の **DBA** と **sql** から別の値に変更できます。「ユーザの認証」 29 ページを参照してください。
- **同期** 同期を管理するオブジェクトのセットは、Connection オブジェクトからアクセスできます。「データの同期」 30 ページを参照してください。
- **テーブル** Connection.getTable メソッドを使用して、Ultra Light テーブルにアクセスします。「getTable メソッド」 64 ページを参照してください。

ページ間の接続とアプリケーション状態の管理

JavaScript 変数のスコープは、Web ページ 1 つに制限されます。ほとんどの Web アプリケーションでは複数のページが要求されるため、アプリケーションのページ間でオブジェクトを永続させるメカニズムが必要です。

Ultra Light for M-Business Anywhere では、ULTable、ResultSet、PreparedStatement の各オブジェクトについて永続性を用意しています。これらのオブジェクトをページ間で永続させるには、オブジェクトの作成時にパラメータとして「**persistent name**」を指定します。それ以降のページで永続的な名前を使用できます。

接続オブジェクトをページ間で使用するには、各ページでその接続を開き直します。それには reOpen メソッドを使用する方法があります。各 Web ページで JavaScript ファイルをインクルードして設定を初期化することにより、各ページで open メソッドを指定するという方法もあります。この方法の例については、サンプル・ファイル *samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB¥main.htm* と *samples-dir¥UltraLiteForMBusinessAnywhere¥Simple¥main_page.htm* を参照してください。

ページ間で接続を開き直すための要件により、Ultra Light アプリケーションに対してセキュリティ機能が提供されます。セキュリティ機能を使用すると、ページ間を移動するときにユーザにパスワードなどの情報を確認させることを要求できます。

別の Web ページで Ultra Light オブジェクトが不要な場合は、メモリを節約するために、アプリケーションではそのオブジェクトに対して close メソッドを発行する必要があります。

参照

- 「reOpenConnection メソッド」 78 ページ
- 「PreparedStatement クラス」 87 ページ
- 「ResultSet クラス」 97 ページ
- 「ULTable クラス」 156 ページ
- 「PreparedStatement クラス」 87 ページ

M-Business Anywhere アプリケーションにおける永続的な名前

HTML では、制御が新しいページに移ると、古いページで割り付けられていた JavaScript オブジェクトへのすべてのハンドルが失われます。たとえば main.html には、M-Business Anywhere データベース接続オブジェクトがあります。

```
conn = dbMgr.openConnection("...");
```

main.html のリンクをクリックして *insert.html* などの別のページに移動すると、*insert.html* ではオブジェクト "conn" が見つかりません。この接続オブジェクトを取得し直すには、dbMgr.openConnection("...") をもう一度呼び出す必要がある可能性があります。ただし、接続オブジェクトはメモリ内にまだ存在するため、そのようにする必要はありません。そのオブジェクトに対する JavaScript ハンドルが失われたにすぎません。

DataManager、Connection、ULTable、PreparedStatement、ResultSet に対するすべての M-Business Anywhere API 呼び出しに persistName 引数が存在するのはこのためです。たとえば、M-Business Anywhere ランタイムが Ultra Light 接続オブジェクトに対する JavaScript からの呼び出しを受け取ると、M-Business Anywhere は、同じ persistName を持つ接続オブジェクトがメモリ内に存在するかどうかを最初にチェックします。一致するオブジェクトが見つかると、ランタイムはその接続オブジェクトを返します。見つからない場合は、M-Business Anywhere は通常の手順に移り、新しい Ultra Light データベース接続を作成して返します。

永続的な名前の使用

M-Business Anywhere オブジェクト間の階層には 2 種類あります。どちらの場合も、次のように DatabaseManager と Connection から始まります。

- DatabaseManager -> Connection -> Table (テーブル API の場合)
- DatabaseManager -> Connection -> PreparedStatement -> ResultSet (動的 SQL API の場合)

これらの M-Business Anywhere オブジェクトを永続的な名前で取得するには、最上位レベルのオブジェクトを永続的な名前で取得し、次に必要な M-Business Anywhere オブジェクトに至るまで、階層ツリーで上位レベルにあるすべての M-Business Anywhere オブジェクトを取得する必要があります。

たとえば、既存の ULTable オブジェクトを insert.html から取得する場合は、*main.html* で dbMgr、conn、table の各オブジェクトに永続的な名前を付けてから、*insert.html* で永続的な名前を使用してそれらすべてを取得します。

main.html のコード・セグメント：

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );
// "simple" is the persistent name here. A real database manager object is allocated

var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here. A real database connection is made.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// a real table is allocated
```

insert.html のコード・セグメント：

```
var dbMgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.simple" );
// "simple" is the persistent name here.
// The allocated database manager object from main.html is returned

var conn = dbMgr.openConnection( "CON=simple_con;..." );
// "simple_con" is the persistent name here.
// The existing connection object from memory is returned.

var custTable = conn.getTable( "ULCustomer", "simpleCustTable" );
// the existing table object is returned.

var newTable = conn.getTable( "ULOrder", "simpleOrderTable" );
// since there is no order table from main.html,
// it does not exist in memory. A real order table object is allocated.
```

永続的な名前の適切な使用

共通で使用されるコードは、JavaScript ファイルに配置します。M-Business Anywhere アプリケーションのほとんどの HTML ページでは、DatabaseManager オブジェクト、Connection オブジェクト、主要な ULTable オブジェクトを参照する必要があるため、これらのオブジェクトを作成する (または、永続的な名前でこれらのオブジェクトを取得する) コードを共通の JavaScript ファイルに配置し、オブジェクトを使用する HTML ページの最初で、そのファイルをインクルードする方が便利です。M-Business Anywhere サンプル・プログラムの "Simple" と "CustDB" の両方で、その方法が示されています。

別のページからオブジェクトを使用する予定がない場合は、そのオブジェクトを閉じます。M-Business Anywhere アプリケーションに HTML ページが 1 つしかない場合は、永続的な名前を使用する必要はありません。永続的な名前の引数を NULL に設定することもできます。一方、各 HTML ページで PreparedStatement オブジェクトと ResultSet オブジェクトが多数開かれている場合は、開発者は、これらのオブジェクトをメモリ内に保持することで永続名を使用して別の HTML ページから簡単に取得できるという便利さと、これらのオブジェクトが常に存在するために生じるメモリの浪費とのバランスを取る必要があります。たとえば、5 個の PreparedStatement オブジェクトと 10 個の ResultSet オブジェクトが *main.html* で作成されたとします。これらはメモリを大量に占有しています。アプリケーションで *insert.html* に移動する場合、これらのオブジェクトのうち永続的な名前で参照する必要があるのが一部だけであれば、必要ではなくなったオブジェクトによってメモリが浪費されています。*insert.html* で新しく PreparedStatement オブジェクトと ResultSet オブジェクトを作成しようとすると、メモリが不足する可能性があります。これを解決するには、*insert.html* でこれらの PreparedStatement オブジェクトや ResultSet オブジェクトを必要としないことがわかっている場合は、*main.html* の最後に明示的にオブジェクトを閉じるようにします。

各 M-Business Anywhere オブジェクトの状態は、永続的な名前で取得される場合も保持されます。1 ページ目からの永続的な ULTable オブジェクトが存在する状態で、2 ページ目から同じ永続名を使用して openTable メソッドを呼び出すと、1 ページ目での状態と同じ状態でその ULtable オブジェクトを取得します。1 ページ目から移動するときにカーソルがテーブルで n 番目のローにある場合、2 ページ目でこのオブジェクトを取得したときも、カーソルは n 番目のローにあるままです。「最初のローの前」にカーソルが来ることはありません。

ResultSet で永続的な名前を使用する場合は注意してください。PreparedStatement にプレースホルダがある場合は、ResultSet に永続名を付けるかどうかについて慎重に考慮する必要があります。たとえば、*main.html* に次のコードがあるとします。


```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

insert.html で同じ ResultSet オブジェクトが必要な場合は、次のようにする必要があります。

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" );

//OrderStmt.setInt(1, 5000); // no need to do this since both the OrderStmt and
OrderResultSet are retrieve from "cache" without any SQL statement being
actually executed

var OrderResultSet = OrderStmt.executeQuery( "order_query_result" );
```

この OrderResultSet オブジェクトには、"order_id" が 5000 に設定された場合と同じ結果が格納されます。

しかし、別の状況も考えられます。Order テーブルで同じクエリを実行するために、同じ PreparedStatement が必要だとします。ただし、クエリでは 5000 以外の order_id を使用します。この場合、永続的な名前を PreparedStatement に割り当てることはできますが、ResultSet には永続的な名前は必要ありません。この状況では order_id が異なるため、前の例とは違う結果セットになります。*main.html* では、次のような同様のコードになります。

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // with persistent name

OrderStmt.setInt(1, 5000);

var OrderResultSet = OrderStmt.executeQuery( null ); // notice here, no persistent name
```

insert.html では、次のようにして新しい ResultSet を取得します。

```
var OrderStmt = Connection.prepareStatement(
"SELECT order_id, disc, quant FROM ULOrder WHERE order_id = ?",
"order_query_stmt" ); // get the prepared statement from memory with persistent name

OrderStmt.setInt(1, 6000); // set a different place holder value

var OrderResultSet = OrderStmt.executeQuery( null ); // a real query is executed
here!
```

この例では、プレースホルダの値が異なるか、Order テーブルで返される結果セットが異なると予期される他の操作が実行されるため、executeQuery の呼び出し時に ResultSet に永続的な名前を使用する必要がありません。

データベースの暗号化と難読化

Ultra Light for M-Business Anywhere を使用して、Ultra Light データベースを暗号化したり、難読化したりできます。

データベース暗号化の詳細については、「Ultra Light データベースの保護」 『Ultra Light データベース管理とリファレンス』を参照してください。

暗号化

Ultra Light データベースは、暗号化しないことも、暗号化や難読化を適用することもできます。データベースを暗号化や難読化する場合は、データベースの作成時に選択する必要があります。

Ultra Light データベースの暗号化では、強力な業界標準技術を使用して、データベース内のデータを暗号化します。暗号化は、データベースの作成時に指定するキー・フレーズに基づいて行われます。このキー・フレーズは、データベースに接続するときにも指定する必要があります。

Ultra Light データベースを暗号化した場合、そのデータベースに接続するときに正しい暗号化キーを指定しなければ、接続に失敗します。

EncryptionKey プロパティの詳細については、「ConnectionParms クラス」 71 ページと「ChangeEncryptionKey メソッド」 58 ページを参照してください。

難読化

難読化とは、非常に弱い形態の暗号化のことで、ファイルやディスクの閲覧プログラムからデータベースの内容を不用意に閲覧されないように、単にデータベース内のデータをマスクします。データベースを難読化するには、creationParms.obfuscate ブール値を true に設定します。次に例を示します。

```
var create_parms = dbMgr.createCreationParms();
create_parms.obfuscate = true;
```

例

暗号化キーを変更するには、新しい暗号化キーを Connection オブジェクトで指定します。changeEncryptionKey メソッドを呼び出す前に、アプリケーションで既存の暗号化キーを使用して、暗号化されたデータベースに接続する必要があります。次のコード例では "apricot" が新しい暗号化キーです。

```
conn.changeEncryptionKey("apricot")
```


SQL を使用したデータ操作

Ultra Light アプリケーションは、SQL またはテーブル API を使用してテーブル・データにアクセスできます。この項では、SQL を使用したデータ・アクセスについて説明します。

テーブル API の詳細については、「テーブル API を使用したデータ操作」 21 ページを参照してください。

この項では、SQL を使用して次の操作を行う方法を説明します。

- ローの挿入、削除、更新
- クエリの実行
- 結果セットのローのスクロール

この項では、SQL 言語そのものについては説明しません。SQL 機能の詳細については、SQL Anywhere サーバ - SQL リファレンスを参照してください。

データ操作：INSERT、UPDATE、DELETE

Ultra Light では、SQL データ操作言語の操作や DDL 操作を実行できます。これらの操作は、PreparedStatement クラスのメンバである ExecuteStatement メソッドを使用して実行します。

PreparedStatement クラスの詳細については、「PreparedStatement クラス」 87 ページを参照してください。

準備文のパラメータ・マーカ
Ultra Light は、パラメータ・マーカ '?' を使用して変数値を処理します。INSERT、UPDATE、DELETE で、準備文での順序位置に従ってそれぞれの '?' が参照されます。たとえば、最初の '?' は 1、2 番目の '?' は 2 のようになります。

♦ ローを挿入するには、次の手順に従います。

1. PreparedStatement オブジェクトを宣言します。

   ```
   var PrepStmt;
   ```

2. INSERT 文を準備文オブジェクトに割り当てます。次のコードでは、TableName と ColumnName がテーブルとカラムの名前です。

   ```
   PrepStmt = conn.prepareStatement(
      "INSERT into TableName(ColumnName) values (?)", null );
   ```

   NULL パラメータは、文に永続的な名前がないことを示します。

3. その文にパラメータ値を割り当てます。

   ```
   var NewValue;
   NewValue = "Bob";
   PrepStmt.setStringParameter(1, NewValue);
   ```

4. 文を実行します。

```
PrepStmt.executeStatement( null );
```

♦ **ローを更新するには、次の手順に従います。**

1. PreparedStatement オブジェクトを宣言します。

```
var PrepStmt;
```

2. UPDATE 文を準備文オブジェクトに割り当てます。次のコードでは、TableName と ColumnName がテーブルとカラムの名前です。

```
PrepStmt = conn.prepareStatement(
  "UPDATE TableName SET ColumnName = ? WHERE ID = ?", null);
```

   NULL パラメータは、文に永続的な名前がないことを示します。

3. データ型に適切なメソッドを使用して、文にパラメータ値を割り当てます。

```
var NewValue;
NewValue = "Bob";
PrepStmt.setStringParameter(1, NewValue);
PrepStmt.setIntParameter(2, 6);
```

4. 文を実行します。

```
PrepStmt.executeStatement( );
```

♦ **ローを削除するには、次の手順に従います。**

1. PreparedStatement オブジェクトを宣言します。

```
var PrepStmt;
```

2. DELETE 文を準備文オブジェクトに割り当てます。

```
PrepStmt = conn.prepareStatement(
   "DELETE FROM customer WHERE ID = ?", null );
```

   NULL パラメータは、文に永続的な名前がないことを示します。

3. その文にパラメータ値を割り当てます。

```
var IDValue;
IDValue = 6;
PrepStmt.setIntParameter( 1, IDValue );
```

4. 文を実行します。

```
PrepStmt.executeStatement( );
```

データ検索：SELECT

SELECT 文を実行すると、PreparedStatement.executeQuery メソッドは ResultSet オブジェクトを返します。ResultSet クラスには、結果セット内をナビゲーションするためのメソッドや、ResultSet を使用してデータを更新するためのメソッドが含まれています。

ResultSet オブジェクトの詳細については、「ResultSet クラス」 97 ページを参照してください。

例

次のコードでは、クエリの結果に ResultSet としてアクセスします。最初に割り当てられたとき、ResultSet は最初のローの前に配置されます。次に ResultSet.moveFirst メソッドが呼び出され、結果セットの最初のレコードをナビゲーションします。

```
var MyResultSet;
var PrepStmt;
PrepStmt = conn.prepareStatement("SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

例

次のコードは、現在のローのカラム値を取得する方法を説明します。この例では、文字データを使用します。その他のデータ型でも同様のメソッドを利用できます。

getString メソッドは構文 MyResultSetName.getString(*Index*) を使用します。*Index* は SELECT 文でのカラム名の順序位置です。

```
if ( MyResultSet.getRowCount() == 0 ) {
} else {
  alert( MyResultSet.getString(1) );
  alert( MyResultSet.getString(2) );
  MyResultSet.moveRelative(0);
}
```

結果セットのナビゲーションの詳細については、「SQL を使用したナビゲーション」 20 ページを参照してください。

次の手順では、SELECT 文を使用して、データベースから情報を取り出します。クエリの結果は、ResultSet オブジェクトに割り当てられます。

♦ **SELECT 文を実行するには、次の手順に従います。**

1. PreparedStatement オブジェクトを宣言します。

   ```
   var OrderStmt;
   ```

2. 準備文を PreparedStatement オブジェクトに割り当てます。

   ```
   OrderStmt = Connection.prepareStatement(
     "SELECT order_id, disc, quant, notes, status, c.cust_id,
      cust_name, p.prod_id, prod_name, price
     FROM ULOrder o, ULCustomer c, ULProduct p
     WHERE o.cust_id = c.cust_id
     AND o.prod_id = p.prod_id
     ORDER BY der_id", "order_query_stmt" );
   ```

   2 番目のパラメータは、ページ間 JavaScript オブジェクトの永続性を提供する永続的な名前です。

3. クエリを実行します。

   ```
   OrderResultSet = OrderStmt.executeQuery( "order_query" );
   ```

クエリの使用方法の詳細については、*samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB ¥custdb.js* の CustDB サンプル・コードを参照してください。

SQL を使用したナビゲーション

Ultra Light for M-Business Anywhere は、幅広いナビゲーション作業を行うために、結果セットをナビゲーションする多数のメソッドを提供します。

次の ResultSet オブジェクトのメソッドを使うと、結果セット内をナビゲーションできます。

- **moveAfterLast**　最後のローの後に移動します。
- **moveBeforeFirst**　最初のローの前に移動します。
- **moveFirst**　最初のローに移動します。
- **moveLast**　最後のローに移動します。
- **moveNext**　次のローに移動します。
- **movePrevious**　前のローに移動します。
- **moveRelative**　いくつかのローを、現在のローを基準にして相対的に移動します。正のインデックス値は結果セット内を前に移動し、負のインデックス値は結果セット内を後ろに移動し、0 はカーソルを移動しません。ロー・バッファを再移植する場合は、0 が便利です。

例

次のコード・フラグメントは、moveFirst メソッドを使用して、結果セット内をナビゲーションする方法を説明します。

```
PrepStmt = conn.prepareStatement(
   "SELECT ID, Name FROM customer", null );
MyResultSet = PrepStmt.executeQuery( null );
MyResultSet.moveFirst();
```

すべての move メソッドで同じ方法を使用できます。

これらのナビゲーション・メソッドの詳細については、「ResultSet クラス」 97 ページを参照してください。

ResultSetSchema オブジェクト

ResultSet.schema プロパティを使うと、クエリのカラムに関する情報を取り出すことができます。

次の例は、ResultSetSchema を使用して、スキーマ情報を取得する方法を示しています。

```
var I;
var MySchema = rs.schema ;
for ( I = 1; I <= MySchema.columnCount; I++) {
  colname = MySchema.getColumnName(I);
  coltype = MySchema.getColumnSQLType(colname).toString();
  alert ( colname + " " + coltype );
}
```


テーブル API を使用したデータ操作

Ultra Light アプリケーションは、SQL またはテーブル API を使用してテーブル・データにアクセスできます。この項では、テーブル API を使用したデータ・アクセスについて説明します。

SQL の詳細については、「SQL を使用したデータ操作」 17 ページを参照してください。

この項では、テーブル API を使用して次の操作を行う方法について説明します。

- テーブルのローのスクロール
- 現在のローの値へのアクセス
- find メソッドと lookup メソッドを使用したテーブルのローの検索
- ローの挿入、削除、更新

テーブル API を使用したナビゲーション

Ultra Light for M-Business Anywhere は、幅広いナビゲーション作業を行うために、テーブルをナビゲーションする多数のメソッドを提供します。

次の ULTable オブジェクトのメソッドを使うと、結果セット内をナビゲーションできます。

- **moveAfterLast**　最後のローの後に移動します。
- **moveBeforeFirst**　最初のローの前に移動します。
- **moveFirst**　最初のローに移動します。
- **moveLast**　最後のローに移動します。
- **moveNext**　次のローに移動します。
- **movePrevious**　前のローに移動します。
- **moveRelative**　いくつかのローを、現在のローを基準にして相対的に移動します。正のインデックス値はテーブル内を前に移動し、負のインデックス値はテーブル内を後ろに移動し、0 はカーソルを移動しません。ロー・バッファを再移植する場合は、0 が便利です。

例

次のコードは、customer テーブルを開き、そのローをスクロールします。次に、各顧客の姓を示す警告を表示します。

```
var tCustomer;
tCustomer = conn.getTable( "customer", null );
tCustomer.open();
tCustomer.moveBeforeFirst();
While (tCustomer.moveNext()) {
  alert( tCustomer.getString(3) );
}
```


インデックスの指定

テーブル・オブジェクトを開くと、テーブルのローがアプリケーションに公開されます。デフォルトでは、ローはプライマリ・キー値の順に公開されますが、インデックスを指定すると特定の順序でローにアクセスできます。

例

次のコードは、ix_name インデックスで順序付けられた Customer テーブルの最初のローに移動します。

```
tCustomer = conn.getTable("customer", null );
tCustomer.openWithIndex("ix_name");
tCustomer.moveFirst();
```

現在のローの値へのアクセス

ULTable オブジェクトは、次のいずれかの位置に常に置かれています。

- テーブルの最初のローの前
- テーブルのいずれかのローの上
- テーブルの最後のローの後ろ

ULTable オブジェクトがローの上に置かれている場合は、ULTable の get メソッドを使用して、現在のローの各カラムの値を取得できます。

例

次のコード・フラグメントは、tCustomer ULTable オブジェクトから 3 つのカラムの値を取り出して、テキスト・ボックスに表示します。

```
var colID, colFirstName, colLastName;
colID = tCustomer.schema.getColumnID( "id" );
colFirstName = tCustomer.schema.getColumnID( "fname" );
colLastName = tCustomer.schema.getColumnID( "lname" );
alert( tCustomer.getInt( colID ) );
alert( tCustomer.getString( colFirstName ) );
alert( tCustomer.getString( colLastName ) );
```

値を設定するには、ULTable のメソッドを使用することもできます。

```
tCustomer.setString( colLastName, "Kaminski" );
```

これらのプロパティへの値の割り当てによって、データベース内のデータの値が変更されることはありません。

位置がテーブルの最初のローの前または最後のローの後ろにある場合でも、プロパティに値を割り当てることができます。しかし、カラムから値を取得することはできません。たとえば、次のコード・フラグメントはエラーを生成します。

```
tCustomer.moveBeforeFirst();
id = tCustomer.getInt( colID );
```


find と lookup を使用したローの検索

Ultra Light には、データを操作するための操作モードがいくつかあります。これらのモードのうちの 2 つ (検索モードとルックアップ・モード) は、検索に使用されます。ULTable オブジェクトには、テーブル内の特定のローを検索するために、これらのモードに対応するメソッドがあります。

注意
find メソッドや lookup メソッドを使用して検索されるカラムは、テーブルを開くのに使用されたインデックスにあることが必要です。

- **find メソッド**　ULTable オブジェクトを開いたときに指定したソート順に基づいて、指定された検索値と正確に一致する最初のローに移動します。

  find メソッドの詳細については、「ULTable クラス」 156 ページを参照してください。

- **lookup メソッド**　ULTable オブジェクトを開いたときに指定したソート順に基づいて、指定された検索値と一致するか、それより大きい値の最初のローに移動します。

  lookup メソッドの詳細については、「ULTable クラス」 156 ページを参照してください。

♦ ローを検索するには、次の手順に従います。

1. 検索モードまたはルックアップ・モードを開始します。

   FindBegin メソッドまたは LookupBegin メソッドを呼び出します。たとえば、次のコード・フラグメントは ULTable.findBegin を呼び出します。

   ```
   tCustomer.findBegin();
   ```

2. 検索値を設定します。

   検索値は、現在のローの値を設定することで設定します。これらの値を設定すると、バッファに影響しますが、データベースには影響しません。たとえば、次のコード・フラグメントは、バッファの姓のカラムを Kaminski に設定します。

   ```
   tCustomer.setString(3, "Kaminski" );
   ```

   マルチカラム・インデックスの場合は、最初のカラムの値が必要ですが、ほかのカラムは省略できます。

3. ローを検索します。

   適切なメソッドを使用して検索を実行します。たとえば、次の指示は、現在のインデックスで指定された値と正確に一致する最初のローを検索します。

   ```
   tCustomer.findFirst();
   ```

ローの挿入、更新、削除

Ultra Light は、テーブルのローを一度に 1 つずつアプリケーションに公開します。ULTable オブジェクトにはカレント・ポジションがあります。カレント・ポジションは、テーブルのローの上、最初のローの前、または最後のローの後ろになります。

アプリケーションがロケーションを変更すると、Ultra Light はバッファにそのローのコピーを作成します。値を取得または設定する操作はすべて、このバッファにあるデータのコピーにのみ影響します。データベースのデータには影響しません。

例

次の文は、バッファの最初のカラムの値を 3 に変更します。

```
tCustomer.setInt( 1 , 3 );
```

Ultra Light のモードの使用

Ultra Light モードによって、バッファ内の値を使用する目的が決まります。Ultra Light には、デフォルト・モードに加えて、次の 4 つの操作モードがあります。

- **挿入モード**　ULTable.insert メソッドを呼び出すと、バッファ内のデータが新しいローとしてテーブルに追加されます。
- **更新モード**　ULTable.update メソッドを呼び出すと、現在のローがバッファ内のデータに置き換えられます。
- **検索モード**　ULTable.find メソッドの 1 つが呼び出されたときに、値がバッファ内のデータに正確に一致するローの検索に使用されます。
- **ルックアップ・モード**　いずれかの ULTable.lookup メソッドが呼び出されたときに、バッファ内のデータと一致するか、それより大きい値のローを検索します。

♦ ローを更新するには、次の手順に従います。

1. 更新するローに移動します。

   テーブルをスクロールするか、find メソッドや lookup メソッドを使用して検索し、ローに移動できます。

2. 更新モードを開始します。

   たとえば、次の指示は、tCustomer テーブル上で更新モードを開始します。

   ```
   tCustomer.updateBegin();
   ```

3. 更新するローの新しい値を設定します。

   たとえば、次の指示は新しい値を Elizabeth に設定します。

   ```
   tCustomer.setString( 2, "Elizabeth" );
   ```

4. Update を実行します。

   ```
   tCustomer.update();
   ```

更新操作が終了すると、直前に更新したローが現在のローになります。ULTable オブジェクトを開いたときに指定したインデックスのカラム値を変更した場合は、現在の位置は不確定です。

デフォルトでは、Ultra Light は autoCommit モードで動作するため、更新は永続的な記憶領域のローに即時適用されます。autoCommit モードを無効にした場合は、コミット操作を実行するまで、更新は適用されません。autoCommit モードの詳細については、「トランザクションの管理」 26 ページを参照してください。

警告
ローのプライマリ・キーを更新しないでください。代わりに、ローを削除して新しいローを追加してください。

ローの挿入

ローの挿入手順は、ローの更新手順とほぼ同じです。ただし、挿入操作の場合は、テーブル内の特定のローにあらかじめ指定する必要はありません。ローは、テーブルを開くときに使用したインデックスで自動的にソートされます。

♦ ローを挿入するには、次の手順に従います。

1. 挿入モードを開始します。

   たとえば、次の指示は、CustomerTable テーブル上で更新モードを開始します。

   ```
   tCustomer.insertBegin();
   ```

2. 新しいローの値を設定します。

   カラムの値を設定しない場合、そのカラムにデフォルト値があるときはデフォルト値が使用されます。カラムにデフォルト値がない場合は、NULL が使用されます。カラムが NULL を許可しない場合は、次のデフォルトが使用されます。

   - 数値カラムの場合は 0
   - 文字カラムの場合は空の文字列

   明示的に値を NULL に設定するには、setNull メソッドを使用します。

   ```
   colID = tCustomer.schema.getColumnID( "id" );
   colFirstName = tCustomer.schema.getColumnID( "fname" );
   colLastName = tCustomer.schema.getColumnID( "lname" );
   tCustomer.setInt( colID, 42 );
   tCustomer.setString( colFirstName, "Mitch" );
   tCustomer.setString( colLastName, "McLeod" );
   ```

3. 挿入を実行します。

   挿入されたローは、Commit を実行したときに永続的にデータベースに保存されます。autoCommit モードでは、Insert メソッドの一部として Commit が実行されます。

   ```
   tCustomer.insert();
   ```

ローの削除

挿入モードや更新モードに対応する削除モードはありません。

次のプロシージャは、ローを削除します。

♦ **ローを削除するには、次の手順に従います。**

1. 削除するローに移動します。
2. 削除を実行します。

```
tCustomer.deleteRow();
```

BLOB データの操作

GetBytes メソッドまたは GetBytesSection メソッドを使用して、BINARY または LONG BINARY と宣言された、カラムの BLOB データをフェッチできます。

「getBytes メソッド」 163 ページと「getBytesSection メソッド」 100 ページを参照してください。

トランザクションの管理

Ultra Light のトランザクション処理は、データベース内のデータの整合性を保証します。トランザクションは、作業の論理単位です。トランザクション全体が実行されるか、トランザクション内の文がどれも実行されないかのいずれかです。

デフォルトでは、Ultra Light for M-Business Anywhere は autoCommit モードで動作します。autoCommit モードでは、挿入、更新、削除はそれぞれ独立したトランザクションとして実行されます。操作が完了すると、データベースに変更が加えられます。

Connection.autoCommit プロパティを false に設定すると、複数文のトランザクションを使用できます。たとえば、2 つの口座間で資金を移動するアプリケーションでは、振り込み元の口座からの引き落としと振り込み先口座への振り込みが、1 つのトランザクションを構成します。

autoCommit が false に設定されている場合は、Connection.commit() 文を実行してトランザクションを完了し、データベースへの変更を永続的なものにするか、Connection.rollback() 文を実行してトランザクションのすべての処理をキャンセルしてください。autoCommit をオフにすると、パフォーマンスが向上します。

> **注意**
> autoCommit を false に設定していても、同期はコミットを実行します。

スキーマ情報へのアクセス

Connection、ULTable、ResultSet オブジェクトにはそれぞれ、スキーマ・プロパティが含まれます。これらのスキーマ・オブジェクトは、データベース内のテーブル、カラム、インデックス、パブリケーションに関する情報を提供します。

- **DatabaseSchema**　データベース内のテーブルの数と名前、日付と時刻のフォーマットなどのグローバル・プロパティ。

  DatabaseSchema オブジェクトを取得するには、Connection.databaseSchema プロパティにアクセスします。

- **TableSchema**　テーブル内のカラムの数と名前、テーブルの Indexes コレクション。

  TableSchema オブジェクトを取得するには、ULTable.schema プロパティにアクセスします。

- **IndexSchema**　インデックス内のカラムに関する情報。インデックスには直接対応するデータがないので、個別の Index オブジェクトはなく、IndexSchema オブジェクトだけが存在します。

  IndexSchema オブジェクトは、TableSchema.getIndex メソッドを使用してアクセスできます。

- **PublicationSchema**　パブリケーションに含まれるテーブルとカラムの数と名前。パブリケーションには PublicationSchema オブジェクトは含まれますが、Publication オブジェクトは含まれません。

  PublicationSchema オブジェクトは、DatabaseSchema.getPublicationSchema メソッドを使用してアクセスできます。

- **ResultSetSchema**　結果セットのカラムの数と名前。

  ResultSetSchema オブジェクトは、PreparedStatement.getResultSetSchema メソッドまたは ResultSet.schema プロパティを使用してアクセスできます。

エラー処理

通常の操作では、Ultra Light for M-Business Anywhere はスクリプト環境でキャッチされて処理されることを意図したエラーをスローすることがあります。「SQLError クラス」 118 ページを参照してください。

エラーは SQLCODE 値として表現され、負の数字は特定の種類のエラーを示します。

Ultra Light for M-Business Anywhere は、DatabaseManager オブジェクトと Connection オブジェクトからのみエラーをスローします。DatabaseManager の次のメソッドは、エラーをスローできます。

- createDatabase
- dropDatabase
- openConnection

Ultra Light for M-Business Anywhere 内での他のエラーや例外はすべて、Connection オブジェクトを経由します。

エラー番号には DatabaseManager オブジェクトと Connection オブジェクトからアクセスできます。次の項を参照してください。

- 「Connection クラス」 56 ページ
- 「DatabaseManager クラス」 75 ページ

ユーザの認証

新しいユーザは既存の接続から追加します。Ultra Light のすべてのデータベースは、デフォルトのユーザ ID **DBA** とパスワード **sql** を使用して作成されるため、まずは初期ユーザとして接続します。

ユーザ ID の変更はできません。ユーザを 1 人追加して既存のユーザを削除します。Ultra Light ではデータベースごとにユーザ ID が 4 つまで許可されます。

接続権限の付与または取り消しの詳細については、「grantConnectTo メソッド」 64 ページと「revokeConnectFrom メソッド」 66 ページを参照してください。

♦ **ユーザを追加する、または既存のユーザのパスワードを変更するには、次の手順に従います。**

1. 既存のユーザとしてデータベースに接続します。
2. 希望するパスワードでユーザに接続権限を付与します。

```
conn.grantConnectTo("Robert", "newPassword");
```

♦ **既存のユーザを削除するには、次の手順に従います。**

1. 既存のユーザとしてデータベースに接続します。
2. 次のように、ユーザの接続権限を取り消します。

```
conn.revokeConnectFrom("Robert");
```

データの同期

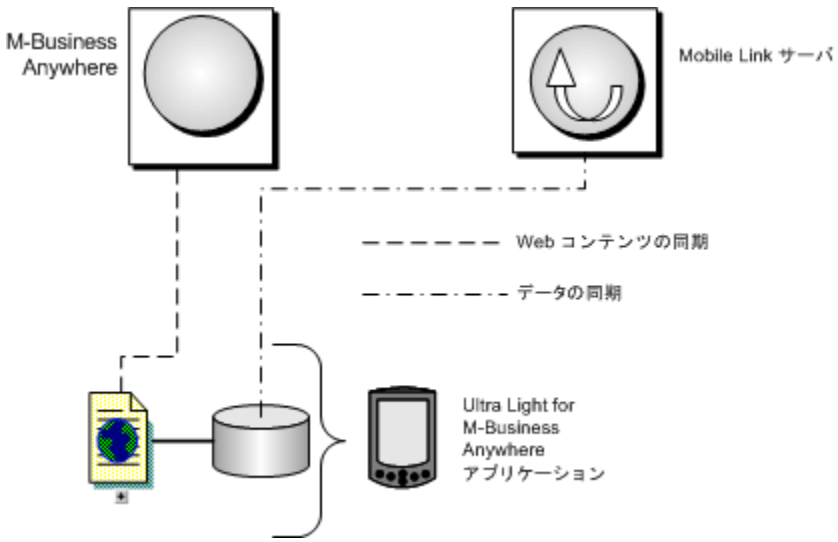
Ultra Light for M-Business Anywhere アプリケーションには、一般に 2 種類の同期があります。

- **Web コンテンツの同期**　Web コンテンツ (アプリケーション自体を定義する HTML ページを含む) が M-Business Anywhere で同期されます。
- **データの同期**　Ultra Light データベースが Mobile Link サーバと同期されます。

これら 2 種類の同期は異なりますが、「**one-button synchronization**」という手法を使用して一緒に開始することができます。ワンタッチ同期は、ほとんどのアプリケーションで推奨されるモデルです。ただし、データと Web コンテンツの同期を完全に分離しておく必要のある場合もあり、その場合の手法については後述します。

ワンタッチ同期

ワンタッチ同期は、Web コンテンツの同期 (M-Business Anywhere を使用) と Ultra Light データの同期 (Mobile Link を使用) を 1 回の操作で開始するための手法です。この手法は、Windows Mobile と Windows でのみ利用できます。ワンタッチ同期のアーキテクチャは、次のとおりです。



ワンタッチ同期では、次のような一連のイベントが発生します。

1. ユーザは、クレードルに置くなどして Web アプリケーションを同期します。
2. M-Business Client は、Web コンテンツを同期します。
3. M-Business Client の MBConnect コンポーネントは、*ulconnect.exe* アプリケーションを呼び出します。
4. *ulconnect.exe* は、Ultra Light データベースの同期を開始します。

5. データが Mobile Link と同期されます。

ワンタッチ同期を実装するには、次の手順を実行します。

1. アプリケーションで、Mobile Link 同期用の同期パラメータを設定します。

   M-Business Anywhere を使用して同期を行う場合は、SyncParms.setMBAServer メソッドを使用してホストとポートの同期パラメータを設定できます。「setMBAServer メソッド」 135 ページを参照してください。

   そうでない場合は、標準の方法で同期パラメータを設定してください。「SyncParms クラス」 130 ページを参照してください。

2. *ulconnect.exe* で読み込めるように、同期パラメータを保存します。

   Connection.saveSyncParms メソッドを呼び出して、同期パラメータを保存します。「saveSyncParms メソッド」 67 ページを参照してください。

データの同期

ほとんどのユーザは、データの同期と Web コンテンツの同期の両方を開始するワンタッチ同期を使用する方が便利です。詳細については、「ワンタッチ同期」 30 ページを参照してください。

この項は、Web コンテンツの同期と独立してデータを同期したいユーザを対象としています。

同期には、Mobile Link サーバと適切なライセンスが必要です。CustDB サンプル・アプリケーションには、同期の実例もあります。

Ultra Light for M-Business Anywhere は、TCP/IP、HTTP、HTTPS、HotSync 同期をサポートしています。同期は、Ultra Light アプリケーションによって開始されます。いずれの場合でも、Connection オブジェクトのメソッドとプロパティを使用して同期を制御します。

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。

♦ TCP/IP または HTTP で同期するには、次の手順に従います。

1. 同期情報を準備します。

   Connection.syncParms オブジェクトの必須プロパティに値を割り当てます。

   設定するプロパティと値の詳細については、「Ultra Light クライアント」 『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。

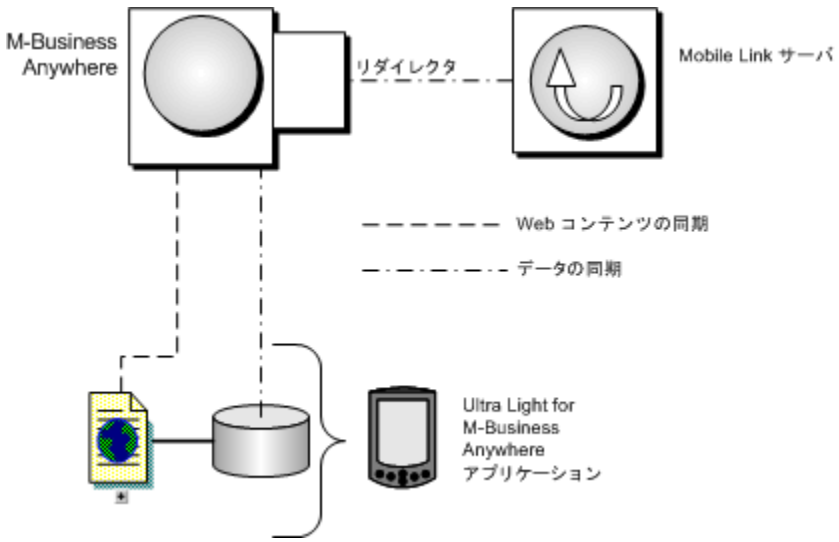
2. 同期を実行します。

   Connection.synchronize メソッドを呼び出します。

M-Business Anywhere を使用したデータの同期

ワンタッチ同期と個別のデータ同期のどちらを使用するかにかかわらず、M-Business Anywhere Server が Mobile Link サーバとの間でデータを送受信するように設定するために、Mobile Link リダイレクタを使用することができます。ファイアウォールの外部からの同期の場合、これによって、外部からアクセス可能であることが必要なポート数が少なくて済みます。

次の図は、ワンタッチ同期の場合のアーキテクチャを示しています。



♦ **M-Business Anywhere を使用してデータを同期するには、次の手順に従います。**

1. サーバ側で、M-Business Anywhere と Mobile Link サーバとの間でデータを送受信するように Mobile Link リダイレクタを設定します。「M-Business Anywhere リダイレクタ (旧式)」『Mobile Link - サーバ管理』を参照してください。
2. クライアントで同期パラメータを設定して、Ultra Light 同期が M-Business Anywhere のホストとポート番号に送信されるようにします。SyncParms.setMBAServer メソッド使用してこの作業を行うことができます。「setMBAServer メソッド」 135 ページを参照してください。
3. クライアント・アプリケーションから、ワンタッチ同期または個別のデータ同期を使用して、同期を開始します。次の項を参照してください。
   - 「ワンタッチ同期」 30 ページ
   - 「データの同期」 31 ページ

Ultra Light for M-Business Anywhere アプリケーションの配備

アプリケーションが完了したら、またはアプリケーションをテストしたい場合、アプリケーションをデバイスに配備する必要があります。この項では、デバイスに Ultra Light アプリケーションを配備するための手順について説明します。

Windows Mobile と Windows デスクトップへのアプリケーションの配備

Ultra Light アプリケーションを Windows Mobile デバイスに配備するには、次の手順を実行する必要があります。

- アプリケーションと Ultra Light コンポーネントを配備します。「Ultra Light for M-Business Anywhere クイック・スタート」 6 ページを参照してください。
- Ultra Light データベースの初期コピーを配備します。「Ultra Light for M-Business Anywhere クイック・スタート」 6 ページを参照してください。

  多くの場合、Ultra Light データベースを配備すれば十分です。このため、同期を使用してデータの初期コピーをロードできます。

  データベースは、アプリケーションが特定できるロケーションに配備する必要があります。Database On CE 接続パラメータは、Windows Mobile 用のロケーションを定義します。Database on Desktop 接続パラメータは、Windows 用のロケーションを定義します。次の項を参照してください。

  ○ 「Ultra Light CE_FILE 接続パラメータ」 『Ultra Light データベース管理とリファレンス』
  ○ 「Ultra Light DBF 接続パラメータ」 『Ultra Light データベース管理とリファレンス』

ワンタッチ同期を使用するアプリケーションの配備

ワンタッチ同期では、*ulconnect.exe*、*ulconnect.udb*、*ulpod11.dll*、*ulrt11.dll* を含む一連のファイルが必要です。Windows Mobile の場合、これらのファイルはディレクトリ *install-dir¥ultralite¥UltraLiteForMBusinessAnywhere¥ce¥arm¥* のファイル *ulpod.cab* 内にあります。この cab ファイルを Windows Mobile デバイスに配備すると、cab ファイルの内容が適切なロケーションに自動的にインストールされます。Windows の場合、必要なファイルは、ディレクトリ *install-dir¥ultralite¥UltraLiteForMBusinessAnywhere¥win32¥386¥* から手動で配備する必要があります。

Palm OS へのアプリケーションの配備

Ultra Light アプリケーションを Palm OS デバイスに配備するには、次の手順を実行する必要があります。

- アプリケーションと Ultra Light コンポーネントを配備します。「Ultra Light for M-Business Anywhere クイック・スタート」 6 ページを参照してください。
- Ultra Light データベースの初期コピーを配備します。「Ultra Light for M-Business Anywhere クイック・スタート」 6 ページを参照してください。

  多くの場合、適切に初期化された Ultra Light データベース・ファイルを配備すれば十分です。このため、同期を使用してデータの初期コピーをロードできます。

  Palm OS に配備するための *.pdb* ファイルは、ulxml と ulinit を含む任意の Ultra Light ユーティリティを使用して作成できます。

  データベースは、アプリケーションがロケーションを特定できるように正しい作成者 ID を使用して指定する必要があります。Database On Palm 接続パラメータは、作成者 ID を使用してデータベースを検索します。「Ultra Light PALM_FILE 接続パラメータ」 『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。
- HotSync 用 Mobile Link 同期コンジットを配備します。

  この手順が必要なのは、HotSync を使用してアプリケーションを同期する場合のみです。「Palm OS の HotSync」 『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。

チュートリアル：M-Business Anywhere 用のサンプル・アプリケーション

目次

M-Business Anywhere の開発チュートリアルの概要

このチュートリアルでは、プラットフォームを問わない Ultra Light for M-Business Anywhere アプリケーションを構築する方法について説明します。アプリケーションも、統合データベースと同期できます。

所要時間

このチュートリアルは、コードをコピーして貼り付ける場合、約 60 分で終了します。*main.htm* と *tutorial.js* の完全なコード・サンプルは、このチュートリアルの最後の「main.htm と tutorial.js のリスト」 49 ページにあります。

前提条件

このチュートリアルでは、JavaScript によるプログラミング、M-Business Anywhere 環境でのモバイル・アプリケーション開発、Sybase Central を使用したデータベース管理についての基本的な知識を前提としています。

参照

- 「データベース作成ウィザードを使用したデータベースの作成」 『Ultra Light データベース管理とリファレンス』
- 「Ultra Light データベースの作成と設定」 『Ultra Light データベース管理とリファレンス』

レッスン 1: データベースの設定

このレッスンでは、チュートリアルで使用するリモート・データベースを作成する方法、および同期モデルを統合データベースに展開する方法について説明します。

♦ **データベースを設定するには、次の手順に従います。**

1. このチュートリアル用のディレクトリを作成します。このチュートリアルでは、保存先ディレクトリを *c:¥tutorial* とします。別の名前のディレクトリを作成した場合は、チュートリアルを通じてそのディレクトリを使用してください。
2. 次の情報に従い、Sybase Central を使用して Ultra Light データベースを作成します (リモート・データベースの作成については、「データベース作成ウィザードを使用したデータベースの作成」『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)。
   - **テーブル名** Customer
   - **カラム**

カラム名	データ型 (サイズ)	カラムの NULL 値の許可	デフォルト値
ID	integer	いいえ	オートインクリメント
GivenName	char (15)	いいえ	なし
Surname	char (20)	いいえ	なし
City	char (20)	はい	なし
Phone	char (12)	はい	なし
Street	char(50)	いいえ	なし

3. リモート・データベース・ファイルを次のプラットフォーム用に保存します。
   - **Windows** *c:¥tutorial¥WIN32_OS¥tutorial.udb*
   - **Windows Mobile** *c:¥tutorial¥WIN32_CE¥tutorial.udb*
   - **Palm** *c:¥tutorial¥PALM_OS¥tutorial.udb*

♦ **同期モデルを展開するには、次の手順に従います。**

1. Sybase Central から同期モデルを作成します。同期モデルの作成の詳細については、「モデルの作成」『Mobile Link - クイック・スタート』を参照してください。

   c:¥tutorial を同期モデル・ファイルの作業フォルダとして使用します。**[統合データベース・スキーマ]** ページと **[リモート・データベース・スキーマ]** ページでは、次の設定を使用します。

a. **[統合データベース・スキーマ]** では、SQL Anywhere サンプル・データベースの *demo.db* を使用して、スキーマを取得します。

b. **[リモート・データベース・スキーマ]** では、Ultra Light サンプル・データベースの *tutorial.udb* (または *.pdb*) を使用して、スキーマを取得します。

c. ウィザードのそれ以外のページでは、デフォルトを使用します。

2. Sybase Central から同期モデルを展開します。同期モデルの展開の詳細については、「モデルの配備」『Mobile Link - クイック・スタート』を参照してください。

a. **[統合データベースの展開先]** ページでは、SQL Anywhere サンプル・データベースを統合データベースとして使用します。

b. **[リモート・データベースの展開]** ページでは、**[既存の SQL Anywhere または Ultra Light データベース]** オプションを選択して、同期モデルを Ultra Light データベース *tutorial.udb* (または *.pdb*) に展開します。

c. **[既存のリモート・データベース]** ページでは、**[リモート・データベースに接続して変更を直接適用する]** チェックボックスをオフにします。

d. **[Mobile Link ユーザ]** ページでは、Mobile Link サーバに接続するための次の設定を指定します。

- **ユーザ名** tutorial
- **パスワード** tutorial

e. ウィザードのそれ以外のページでは、デフォルトを使用します。

同期モデル展開ウィザードを完了すると、コマンド・ファイル *tutorial_mlsrv.bat* が生成されます。このコマンド・ファイルは、チュートリアルの後の方で使用します。

レッスン 2：アプリケーション・ファイルの作成

このレッスンでは、アプリケーション・ファイルを設定する方法について説明します。

♦ アプリケーション・ファイルの作成

1. ファイル *c:¥tutorial¥main.htm* を作成します。

   このチュートリアルの後の方で、*main.htm* に論理を追加します。ここでは、プラットフォーム固有のファイル *ul_deps.html* をインクルードするように設定します。

   次の内容を *main.htm* に追加します。

   ```
   <html>
   <body>
   <a href="AG_DEVICEOS/ul_deps.html"></a>
   </body>
   </html>
   ```

2. プラットフォーム固有のファイル *ul_deps.html* を作成します。

   このファイルは、次に示すように、さまざまなオペレーティング・システムに特化されたバイナリ・ファイルを参照します。

   - **Windows** *c:¥tutorial¥WIN32_OS¥ul-deps.htm*

     ```
     <!-- WIN32_OS¥ul_deps.html -->
     <html>
       <a href="ulpod11.dll"></a>
       <a href="tutCustomer.udb"></a>
     </html>
     ```

   - **Windows Mobile** *c:¥tutorial¥WINCE_OS¥ul-deps.htm*

     ```
     <!-- WINCE_OS¥ul_deps.html -->
     <html>
       <a href="AG_DEVICEPROCESSOR/ulpod11.dll"></a>
       <a href="tutCustomer.udb"></a>
     </html>
     ```

   - **Palm** *c:¥tutorial¥PALM_OS¥ul-deps.htm*

     ```
     <!-- PALM_OS¥ul_deps.html -->
     <html>
       <a href="ulpod11.prc"></a>
       <a href="tutCustomer.pdb"></a>
     </html>
     ```

3. Ultra Light Pod ファイル (Windows および Windows Mobile の場合は *ulpod11.dll*、Palm の場合は *.prc*) を *tutorial* ディレクトリにコピーします。

   - Windows デスクトップでは、*ulpod11.dll* を *install-dir¥UltraLite¥UltraLiteForMBusinessAnywhere¥win32¥386* から *c:¥tutorial¥WIN32_OS¥* にコピーします。
   - Windows Mobile では、*ulpod11.dll* を *install-dir¥UltraLite¥UltraLiteForMBusinessAnywhere¥CE¥Arm* から *c:¥tutorial¥WINCE_OS¥arm¥* にコピーします。
   - Palm OS では、*ulpod11.prc* を *install-dir¥UltraLite¥UltraLiteForMBusinessAnywhere¥Palm¥68k* から *c:¥tutorial¥PALM_OS¥* にコピーします。

すべてのアプリケーション・ファイルが配置されました。

レッスン 3：M-Business Anywhere の設定

このレッスンでは、M-Business Anywhere を設定して Ultra Light チュートリアル・チャネルを同期する方法について説明します。

♦ M-Business Anywhere の設定

1. M-Business Anywhere 管理コンソールを開き、**Admin** として (パスワードなしで) ログインします。
2. *tutorial* という名前の新しいユーザを作成します。
3. 次のようにして、このユーザのチャネルを作成します。
   a. チャネルに次の設定を使用します。お使いの Web ブラウザに応じて適切な URL に置き換えてください。
      - **[Channel Title]** Ultra Light チュートリアル
      - **[Channel URL]** *http://<yourwebserver>/tutorial/main.htm*

        このロケーションは、Web サーバがサービスを行う、チュートリアルの *main.htm* ページの URL です。
      - **[Channel Size]** 1000 KB
      - **[Channel Link Depth]** 3
      - **[Allow Binary Distribution]** true (オン)
      - **[Hide From Users]** false (オフ)

ユーザとチャネルが M-Business Anywhere に設定されました。次の手順では、このチャネルの内容を M-Business Client に同期します。これは、使用するどのプラットフォームからでも実行できます。

次の手順では、M-Business Client がインストール済みであることを前提としています。**[ツール] - [オプション]** をクリックし、**[JavaScript エラーを表示する]** クライアント・オプションを設定することをおすすめします。このように設定することで、アプリケーションで発生するあらゆるエラーのデバッグが簡単になります。

♦ デバイス用のチャネルの同期

- Ultra Light チャネルをデバイスに同期します。

  この時点では、アプリケーションのコンテンツはないため、ブランクのページが表示されます。

M-Business Anywhere 環境の詳細については、『**M-Business Anywhere アプリケーション・デベロッパ・ガイド**』を参照してください。

レッスン 4：アプリケーションへの起動コードの追加

このレッスンでは、Ultra Light データベースに接続するための起動コードをアプリケーションに追加します。

♦ アプリケーションへのコンテンツの追加

1. 次の内容を *main.htm* の、<a> タグの直前に追加します。

```
<form name="form">
<br><td> ID: </td>
   <td> <input type="text" name="ID" size="10"> </td>
<br><td> Given Name: </td>
   <td> <input type="text" name="GivenName" size="15">
   </td>
<br><td> Surname: </td>
   <td> <input type="text" name="Surname" size="50"> </td>
<br><td> Street: </td>
   <td> <input type="text" name="Street" size="20"> </td>
<br><td> City: </td>
   <td> <input type="text" name="City" size="20"> </td>
<br><td> Phone: </td>
   <td> <input type="text" name="Phone" size="12"> </td>
<br>
<br>

<table>
<tr>
   <td> <input type="button" value="Insert"
      onclick="ClickInsert();"> </td>
   <td> <input type="button" value="Next"
      onclick="ClickNext();"> </td>
   <td> <input type="button" value="Prev"
      onclick="ClickPrev();"> </td>
</tr>
<tr>
   <td colspan=3>
   <input type="button" value="Synchronize"
   onclick="ClickSync();">
   </td>
</tr>
</table>
</form>
```

2. アプリケーション論理を提供する JavaScript ファイル *c:¥tutorial¥tutorial.js* を作成します。
3. Ultra Light Pod オブジェクト用の次の変数宣言を *tutorial.js* に追加します。

```
var DB_mgr;
var Connection;
var Table;
```

4. チュートリアル・データベースに接続するための次の関数を *tutorial.js* に追加します。

```
function Connect()
{
   var    dir;
   var    open_parms;
   var    browser = navigator.platform;
```



```
    DB_mgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.Tutorial" );
    if( DB_mgr == null ) {
     alert( "Error: cannot create database manager: " + DB_mgr.sqlCode );
  return;
    }
    dir = DB_mgr.directory;
    if( browser == "Palm OS" ) {
  open_parms = "con=tutorial;palm_file=tutorial"
    } else {
     open_parms = "con=tutorial;" + "file_name=" + dir + "¥¥tutorial.udb";
    }
    try {
  Connection = DB_mgr.reOpenConnection( "tutorial" );
     if( Connection == null ) {
     Connection = DB_mgr.openConnection( open_parms );
  }
    } catch( ex ) {
     if( DB_mgr.sqlCode !=
         DB_mgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
         alert( "Error: cannot connect to database: " + ex.getMessage() );
     return;
     }
  }
```

5. onload イベント・ハンドラを使用して、アプリケーションの起動時にデータベースに接続します。*main.htm* を次のように変更します。

   a. 次の行を <body> タグの直前に追加することで、*tutorial.js* をロードします。

      ```
      <script src="tutorial.js"></script>
      ```

   b. <body> タグを次のように変更します。

      ```
      <body onload="Connect();">
      ```

6. アプリケーションをテストします。

   Ultra Light チュートリアル・チャネルを同期します。同期アプリケーションがチュートリアル・データベースに接続します。

レッスン 5：データ操作とナビゲーションの追加

このレッスンでは、アプリケーションにデータ操作論理とナビゲーション論理を追加する方法について説明します。

♦ テーブルの初期化

1. 次のコードを *tutorial.js* の **Connect** 関数の末尾に追加することで、データベースの **Customer** テーブルを表す CustomerTable を初期化します。

```
try {
  CustomerTable = Connection.getTable( "customer", null );
  CustomerTable.open();
} catch( ex3 ) {
  alert("Error: " + ex3.getMessage() );
}
```

2. データベースと Web フォームの間でデータを移動するための変数を追加します。

   顧客データ用の次の変数宣言を *tutorial.js* の先頭に追加します。

```
var GivenName = "";
var Surname = "";
var Street = "";
var City = "";
var Phone = "";
var ID = "";
```

3. 顧客のデータをフェッチして表示するための関数を作成します。

   次の関数を *tutorial.js* に追加します。この関数は、顧客データの現在のローをフェッチして、NULL カラムが空の文字列として表示されるようにします。

```
function Fetch()
{
    if( Table.getRowCount() == 0 ) {
 GivenName = "";
 Surname = "";
 Street = "";
 City = "";
 Phone = "";
 ID = "";
 return;
    }
    ID = Table.getString( Table.schema.getColumnID( "ID" ) );
    GivenName = Table.getString( Table.schema.getColumnID( "GivenName" ) );
    Surname = Table.getString( Table.schema.getColumnID( "Surname" ) );
    Street = Table.getString( Table.schema.getColumnID( "Street" ) );
    if( Table.isNull( Table.schema.getColumnID( "City" ) ) ) {
 City = "";
    } else {
 City = Table.getString( Table.schema.getColumnID( "City" ) );
    }
    if( Table.isNull( Table.schema.getColumnID( "Phone" ) ) ) {
 Phone = "";
    } else {
 Phone = Table.getString( Table.schema.getColumnID( "Phone" ) );
    }
}
```

次のコードを *main.htm* の、<script> タグの直後に追加します。**DisplayRow** は、データベースから値を取得して、Web フォームに表示します。**FetchForm** は、Web フォームの現在の値を取得して、データベース・コードで利用できるようにします。

```
<script>
function DisplayRow() {
    Fetch();
    document.form.ID.value = ID;
    document.form.GivenName.value = GivenName;
    document.form.Surname.value = Surname;
    document.form.Street.value = Street;
    document.form.City.value = City;
    document.form.Phone.value = Phone;
}

function FetchForm() {
    GivenName = document.form.GivenName.value;
    Surname = document.form.Surname.value;
    Street = document.form.Street.value;
    City = document.form.City.value;
    Phone = document.form.Phone.value;
}
</script>
```

4. アプリケーションのロード時に、現在のローを表示する **DisplayCurrentRow** を呼び出します。*main.htm* の先頭の <body> タグを次のように変更します。

```
<body onload="Connect(); DisplayCurrentRow();">
```

この時点ではまだチュートリアル・データベースにデータがありませんが、チャネルを同期して、アプリケーションが正しく稼働していることを確認するにはよいタイミングです。

♦ ローを挿入するコードの追加

● 顧客データを挿入するための関数を作成します。

次のプロシージャでは、**InsertBegin** を呼び出すと、アプリケーションが挿入モードになり、現在のローのすべての値がデフォルトに設定されます。たとえば、ID カラムは、次のオートインクリメント値を受け取ります。カラム値が設定されると、新しいローが挿入されます。

次の関数を *tutorial.js* に追加します。

```
function Insert()
{
    try {
 Table.insertBegin();
 Table.setString( Table.schema.getColumnID( "GivenName" ), GivenName );
 Table.setString( Table.schema.getColumnID( "Surname" ), Surname );
 Table.setString( Table.schema.getColumnID( "Street" ), Street );
 if( City.length > 0 ) {
     Table.setString( Table.schema.getColumnID( "City" ), City );
 }
 if( Phone.length > 0 ) {
     Table.setString( Table.schema.getColumnID( "Phone" ), Phone );
 }
 Table.insert();
 Table.moveLast();
    } catch( ex ) {
 alert( "Error: cannot insert row: " + ex.getMessage() );
```

```
    }
}
```

次の関数を *main.htm* に追加します。

```
function ClickInsert()
{
    FetchForm();
    Insert();
    DisplayRow();
}
```

レッスン 6：アプリケーションへのナビゲーションの追加

このレッスンでは、**Customer** テーブルのローを前後に移動するためのコードについて説明します。

♦ アプリケーションへのナビゲーション・コードの追加

1. **Next** 関数を *tutorial.js* に追加します。

```
function Next()
{
    if( ! Table.moveNext() ) {
 Table.moveLast();
    }
}
```

2. **Prev** 関数を *tutorial.js* に追加します。

```
function Prev()
{
    if( ! Table.movePrevious() ) {
 Table.moveFirst();
    }
}
```

3. 次の関数を *main.htm* に追加します。

```
function ClickNext()
{
    Next();
    DisplayRow();
}

function ClickPrev()
{
    Prev();
    DisplayRow();
}
```

4. 最初にフォームが表示されると、現在位置が最初のローの前にあるため、コントロールは空です。フォームが表示されたら、**[Next]** と **[Prev]** をクリックして、テーブルのローの間を移動します。

レッスン 7：アプリケーションへの同期の追加

次の手順は、同期を実装します。

♦ アプリケーションへの同期関数の追加

1. **Synchronize** 関数を *tutorial.js* に追加します。

```
function Synchronize()
{
    var sync_parms;

    sync_parms = Connection.syncParms;
    sync_parms.setUserName( "tutorial" );
    sync_parms.setPassword( "tutorial" );
    sync_parms.setVersion( "tutorial" );
    sync_parms.setStream( sync_parms.STREAM_TYPE_TCPIP );
    try {
 Connection.synchronize();
    } catch( ex ) {
     alert( "Error: cannot synchronize: " + ex.getMessage() );
    }
}
```

   同期パラメータは、SyncParms オブジェクトに格納されます。たとえば SyncParms.userName プロパティは、Mobile Link が検索するユーザ名を指定します。SyncParms.sendColumnNames プロパティは、カラム名が Mobile Link に送信されることを指定し、アップロード・スクリプトとダウンロード・スクリプトを生成できるようにします。

   この関数は、TCP/IP 同期ストリームと、デフォルトのネットワーク通信オプション (ストリーム・パラメータ) を使用します。これらのデフォルトのオプションは、Mobile Link サーバを実行しているコンピュータに ActiveSync を使用して接続した Windows Mobile クライアントから、または Mobile Link と同じコンピュータで動作している 32 ビット Windows デスクトップ・クライアントから同期を実行していることを想定しています。これ以外の場合は、同期ストリーム・タイプを変更し、ネットワーク通信オプションを適切な値に設定してください。

   参照：

   - 「setStream メソッド」 138 ページ
   - 「setStreamParms メソッド」 138 ページ

2. 次の関数を *main.htm* に追加します。

```
function ClickSync()
{
    Synchronize();
    DisplayRow();
}
```

3. *tutorial_mlsrv.bat* ファイルを使用することで、Mobile Link サーバを起動します。

Customer テーブル内のデータがダウンロードされます。リモート・データベースの **Customer** テーブル内のローの間を移動できるようになりました。

これでチュートリアルが完了しました。*main.htm* と *tutorial.js* の完全なコード・サンプルについては、「main.htm と tutorial.js のリスト」 49 ページを参照してください。

main.htm と tutorial.js のリスト

次に示すのは、使用する *main.htm* の完全なリストです。

```
<html>
<script src="tutorial.js"></script>

<script>
function DisplayRow() {
    Fetch();
    document.form.ID.value = ID;
    document.form.GivenName.value = GivenName;
    document.form.Surname.value = Surname;
    document.form.Street.value = Street;
    document.form.City.value = City;
    document.form.Phone.value = Phone;
}

function FetchForm() {
    GivenName = document.form.GivenName.value;
    Surname = document.form.Surname.value;
    Street = document.form.Street.value;
    City = document.form.City.value;
    Phone = document.form.Phone.value;
}

function ClickInsert()
{
    FetchForm();
    Insert();
    DisplayRow();
}

function ClickNext()
{
    Next();
    DisplayRow();
}

function ClickPrev()
{
    Prev();
    DisplayRow();
}

function ClickSync()
{
    Synchronize();
    DisplayRow();
}
</script>

<body onload="Connect(); DisplayRow();" >

<form name="form">
<br><td> ID: </td>
    <td> <input type="text" name="ID" size="10"> </td>
<br><td> Given Name: </td>
    <td> <input type="text" name="GivenName" size="15"> </td>
<br><td> Surname: </td>
    <td> <input type="text" name="Surname" size="50"> </td>
<br><td> Street: </td>
```

```
    <td> <input type="text" name="Street" size="20"> </td>
<br><td> City: </td>
    <td> <input type="text" name="City" size="20"> </td>
<br><td> Phone: </td>
    <td> <input type="text" name="Phone" size="12"> </td>
<br>
<br>

<table>
<tr>
    <td> <input type="button" value="Insert" onclick="ClickInsert();"> </td>
    <td> <input type="button" value="Next" onclick="ClickNext();"> </td>
    <td> <input type="button" value="Prev" onclick="ClickPrev();"> </td>
</tr>
<tr>
    <td colspan=3>
    <input type="button" value="Synchronize" onclick="ClickSync();">
    </td>
</tr>
</table>
</form>

<a href="AG_DEVICEOS/ul_deps.htm"></a>
</body>
</html>
```

次に示すのは、参照および使用する *tutorial.js* の完全なリストです。

```
// UltraLite Tutorial

var DB_mgr;
var Connection;
var Table;

var GivenName = "";
var Surname = "";
var Street = "";
var City = "";
var Phone = "";
var ID = "";

function Connect()
{
    var     dir;
    var     open_parms;
    var     browser = navigator.platform;

    DB_mgr = CreateObject( "iAnywhere.UltraLite.DatabaseManager.Tutorial" );
    if( DB_mgr == null ) {
     alert( "Error: cannot create database manager: " + DB_mgr.sqlCode );
 return;
    }
    dir = DB_mgr.directory;
    if( browser == "Palm OS" ) {
 open_parms = "con=tutorial;palm_file=tutorial"
    } else {
     open_parms = "con=tutorial;" + "file_name=" + dir + "¥¥tutorial.udb";
    }
    try {
 Connection = DB_mgr.reOpenConnection( "tutorial" );
     if( Connection == null ) {
     Connection = DB_mgr.openConnection( open_parms );
 }
    } catch( ex ) {
```



```
    if( DB_mgr.sqlCode != DB_mgr.SQLError.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
        alert( "Error: cannot connect to database: " + ex.getMessage() );
    return;
    }
   }
   try {
 Table = Connection.getTable( "Customer", null );
 if( Table != null ) {
    Table.open();
 }
   } catch( ex ) {
 alert( "Error: cannot open table: " + ex.getMessage() );
   }
}

function Fetch()
{
   if( Table.getRowCount() == 0 ) {
 GivenName = "";
 Surname = "";
 Street = "";
 City = "";
 Phone = "";
 ID = "";
 return;
   }
   ID = Table.getString( Table.schema.getColumnID( "ID" ) );
   GivenName = Table.getString( Table.schema.getColumnID( "GivenName" ) );
   Surname = Table.getString( Table.schema.getColumnID( "Surname" ) );
   Street = Table.getString( Table.schema.getColumnID( "Street" ) );
   if( Table.isNull( Table.schema.getColumnID( "City" ) ) ) {
 City = "";
   } else {
 City = Table.getString( Table.schema.getColumnID( "City" ) );
   }
   if( Table.isNull( Table.schema.getColumnID( "Phone" ) ) ) {
 Phone = "";
   } else {
 Phone = Table.getString( Table.schema.getColumnID( "Phone" ) );
   }
}

function Insert()
{
   try {
 Table.insertBegin();
 Table.setString( Table.schema.getColumnID( "GivenName" ), GivenName );
 Table.setString( Table.schema.getColumnID( "Surname" ), Surname );
 Table.setString( Table.schema.getColumnID( "Street" ), Street );
 if( City.length > 0 ) {
    Table.setString( Table.schema.getColumnID( "City" ), City );
 }
 if( Phone.length > 0 ) {
    Table.setString( Table.schema.getColumnID( "Phone" ), Phone );
 }
 Table.insert();
 Table.moveLast();
   } catch( ex ) {
 alert( "Error: cannot insert row: " + ex.getMessage() );
   }
}

function Next()
{
```

```
    if( ! Table.moveNext() ) {
 Table.moveLast();
    }
}

function Prev()
{
    if( ! Table.movePrevious() ) {
 Table.moveFirst();
    }
}

function Synchronize()
{
    var sync_parms;

    sync_parms = Connection.syncParms;
    sync_parms.setUserName( "tutorial" );
    sync_parms.setPassword( "tutorial" );
    sync_parms.setVersion( "tutorial" );
    sync_parms.setStream( sync_parms.STREAM_TYPE_TCPIP );
    try {
 Connection.synchronize();
    } catch( ex ) {
     alert( "Error: cannot synchronize: " + ex.getMessage() );
    }
}
```


Ultra Light for M-Business Anywhere API リファレンス

目次

Ultra Light for M-Business Anywhere のデータ型

JavaScript には、数値データ型と DATE データ型がそれぞれ 1 つだけあります。

この API リファレンスのプロトタイプでは、メソッドやプロパティの説明にその他のデータ型が含まれています。これらの型は、M-Business Anywhere の内部データ型です。UInt32 (32 ビット符号なし整数) などの個別の数値データ型に対しては、渡される可能性のあるデータのサイズと精度について例を記載しています。日付と時間に関連する個別のデータ型 (DATE、TIME、TIMESTAMP) については、渡されるデータから必要な情報を抽出するコードを必要に応じて記述できるような説明を記載しています。

AuthStatusCode クラス

Mobile Link ユーザ認証の実行中にレポートされる可能性のあるステータス・コードを列挙します。

このオブジェクトを DatabaseManager から取得するには、次のように記述します。

```
var authStatus = dbMgr.AuthStatusCode;
```

プロパティ

AuthStatusCode のプロパティは次の定数です。

定数	値	説明
UNKNOWN	0	認証ステータスが不明です。接続がまだ同期を実行していない可能性があります。
VALID	1	ユーザ ID とパスワードは、同期時には有効でした。
VALID_BUT_EXPIRES_SOON	2	ユーザ ID とパスワードは、同期時には有効でしたが、まもなく有効期限が切れます。
EXPIRED	3	ユーザ ID またはパスワードの有効期限が切れているため、認証に失敗しました。
INVALID	4	ユーザ ID またはパスワードが正しくないため、認証に失敗しました。
IN_USE	5	ユーザ ID がすでに使用されているため、認証に失敗しました。

toString メソッド

認証ステータス・コード定数の文字列名を生成します。

構文

String **toString()**;

戻り値

コードの名前、または認識されたコードでない場合は **unknown**。

Connection クラス

Ultra Light データベースへの接続を表します。

次のいずれかのメソッドを使用して、接続がインスタンス化されます。

- DatabaseManager.openConnection
- DatabaseManager.createDatabase

接続は、他の操作の実行前に開きます。また、その接続での操作がすべて終了したら、接続を閉じてからアプリケーションを終了します。

接続で開いたテーブルをすべて閉じてから、その接続を閉じます。

Ultra Light データベース操作の失敗が原因で JavaScript エラーがスローされた場合、SQL エラー・コードが Connection オブジェクトの sqlCode フィールドに設定されます。

プロパティ

プロトタイプ	説明
Boolean autoCommit	各文 (insert、update、delete) の後でコミットを行うかどうかを指定します。 **autoCommit** が false の場合、コミットまたはロールバックが実行されるのは、ユーザが **commit()** メソッドまたは **rollback()** メソッドを呼び出したときだけです。 デフォルトでは、各文の処理が成功した後でデータベースのコミットが実行されます。コミットに失敗する場合は、追加の SQL 文を実行してからコミットを再度実行するか、ロールバック文を実行できます。
String openParms (読み込み専用)	name=value のペアをセミコロンで区切ったリストで、接続パラメータの文字列を取得します。 「Ultra Light 接続パラメータ」『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。
DatabaseSchema databaseSchema (読み込み専用)	データベース・スキーマを取得します。このプロパティが有効なのは、接続が開かれている間だけです。
Boolean skipMBASync (読み込み／書き込み)	ワンタッチ同期中にデータベースを同期するか (false)、スキップするか (true) を指定します。 デフォルトは false です。 「ワンタッチ同期」 30 ページを参照してください。

プロトタイプ	説明
Int32 sqlCode (読み込み専用)	この接続で行った最後の操作の SQL コードを取得します。 SQL コードは、SQL Anywhere の標準のコードであり、この接続の以後の Ultra Light データベース操作によってリセットされます。
SyncParms syncParms (読み込み専用)	この接続の同期設定を取得します。 「Ultra Light の同期パラメータ」『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。
SyncResult syncResult (読み込み専用)	この接続の最後の同期の結果を取得します。 「Ultra Light の同期パラメータ」『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。
INVALID_DATABASE_ID (読み込み専用)	無効なデータベースを示す定数。

cancelGetNotification メソッド

指定された名前に一致する、すべてのキューに登録されている保留中の取得通知コールをキャンセルします。キャンセルしたイベント通知の数を返します。

構文

UInt32 **cancelGetNotification**(String *queue_name*)

パラメータ

- **queue_name**　イベント通知キューの名前。

参照

- 「イベント通知の操作」『Ultra Light データベース管理とリファレンス』
- 「createNotificationQueue メソッド」 59 ページ
- 「declareEvent メソッド」 59 ページ
- 「destroyNotificationQueue メソッド」 60 ページ
- 「getNotification メソッド」 62 ページ
- 「registerForEvent メソッド」 65 ページ
- 「sendNotification メソッド」 67 ページ
- 「triggerEvent メソッド」 69 ページ

ChangeEncryptionKey メソッド

データベースの暗号化キーを、指定された新しいキーに変更します。

構文

changeEncryptionKey(String *newKey*)

パラメータ

- **newkey** データベースの新しい暗号化キー。

備考

暗号化キーが失われた場合は、データベースを開くことができません。

close メソッド

この接続を閉じます。

構文

close()

備考

接続を閉じると、その接続をもう一度開くことはできません。接続をもう一度開くには、新しい接続オブジェクトを作成して開く必要があります。

閉じた接続に関連付けられたオブジェクト (テーブル、スキーマなど) を使用するとエラーになります。

JavaScript では、閉じた接続オブジェクトは接続が閉じた後で自動的に NULL に設定されません。接続を閉じた後で、明示的に接続オブジェクトを NULL に設定することをおすすめします。

commit メソッド

データベースへの未処理の変更をコミットします。

構文

commit()

CountUploadRow メソッド

次回の同期でアップロードするロー数を返します。

構文

UInt32 **countUploadRow(** String *pub-list*, UInt32 *threshold***)**

パラメータ

- **pub-list**　チェックするパブリケーションのカンマ区切りのリスト。

  PublicationSchema クラスを参照してください。

- **threshold**　カウントするローの最大数を決定する値。呼び出しにかかる時間を制限します。値 0 は制限が最大であることを示します。値 1 は、同期の必要なローがあるかどうかを判別する場合に使用します。**threshold** は、**[0,0x0fffffff]** の範囲内であることが必要です。

createNotificationQueue メソッド

この接続のイベント通知キューを作成します。

構文

void **createNotificationQueue(** String *queue_name*, String *parms***)**

パラメータ

- **queue_name**　イベント通知キューの名前。
- **parms**　作成パラメータ。現在は使われず、NULL に設定されています。

備考

指定された名前のイベント通知キューが作成され、以後のイベント通知に使用できます。

参照

- 「イベント通知の操作」『Ultra Light データベース管理とリファレンス』
- 「cancelGetNotification メソッド」 57 ページ
- 「declareEvent メソッド」 59 ページ
- 「destroyNotificationQueue メソッド」 60 ページ
- 「getNotification メソッド」 62 ページ
- 「registerForEvent メソッド」 65 ページ
- 「sendNotification メソッド」 67 ページ
- 「triggerEvent メソッド」 69 ページ

declareEvent メソッド

登録およびトリガされるイベントを宣言します。

構文

void **declareEvent(**String *event_name***)**

パラメータ

- **event_name**　イベントの名前。

参照

- 「createNotificationQueue メソッド」 59 ページ
- 「cancelGetNotification メソッド」 57 ページ
- 「destroyNotificationQueue メソッド」 60 ページ
- 「getNotification メソッド」 62 ページ
- 「registerForEvent メソッド」 65 ページ
- 「sendNotification メソッド」 67 ページ
- 「triggerEvent メソッド」 69 ページ

destroyNotificationQueue メソッド

指定されたイベント通知キューを破棄します。

構文

void **destroyNotificationQueue(**String *queue_name***)**

パラメータ

- **queue_name**　既存のイベント通知キューの名前。

参照

- 「イベント通知の操作」 『Ultra Light データベース管理とリファレンス』
- 「createNotificationQueue メソッド」 59 ページ
- 「cancelGetNotification メソッド」 57 ページ
- 「declareEvent メソッド」 59 ページ
- 「getNotification メソッド」 62 ページ
- 「registerForEvent メソッド」 65 ページ
- 「sendNotification メソッド」 67 ページ
- 「triggerEvent メソッド」 69 ページ

executeNextSQLPassthroughScript メソッド

「次」の SQL パススルー・スクリプトを実行します。

構文

void **executeNextSQLPassthroughScript()**

備考

スクリプトの実行中にエラーが発生した場合、例外がスローされます。

参照

- 「executeSQLPassthroughScripts メソッド」 61 ページ
- 「getSQLPassthroughScriptCount メソッド」 64 ページ

executeSQLPassthroughScripts メソッド

使用可能な「すべて」の SQL パススルー・スクリプトを実行します。

構文

void **executeSQLPassthroughScripts()**

備考

スクリプトの実行中にエラーが発生した場合、例外がスローされます。

参照

- 「executeNextSQLPassthroughScript メソッド」 60 ページ
- 「getSQLPassthroughScriptCount メソッド」 64 ページ

getDatabaseID メソッド

setDatabaseID() で設定された現在のデータベース ID 値を取得します。

構文

UInt32 **getDatabaseID( )**

備考

値が設定されていない場合は、定数 Connection.INVALID_DATABASE_ID が返されます。

getGlobalAutoIncrementUsage メソッド

利用可能なグローバル・オートインクリメントの値の使用済み比率 (%) を返します。

構文

UInt16 **getGlobalAutoIncrementUsage( )**

備考

比率が 100% に近づいたら、**setDatabaseID** を使用して、使用中のアプリケーションに新しいグローバル・データベース ID を設定してください。

getLastDownloadTime メソッド

最後のダウンロードのタイムスタンプを返します。

構文

Date **getLastDownloadTime(** String *pub-list* **)**

パラメータ

- **pub-list**　チェックするパブリケーション名のカンマ区切りのリスト。

備考

パラメータ pub-list は、単一のパブリケーション名を参照することが必要です。データベース全体を最後にダウンロードした時刻を表す場合は SYNC_ALL_DB。

参照

- 「PublicationSchema クラス」 96 ページ

getLastIdentity メソッド

直前に使用した identity の値を返します。

構文

UInt64 **getLastIdentity()**

備考

この関数は、次の SQL 文と同義です。

```
SELECT @@identity
```

この関数は、グローバル・オートインクリメント・カラムで使うと特に便利です。戻り値は、符号なし 64 ビット整数であるデータベース・データ型 UNSIGNED BIGINT です。この文では最後に割り当てられたデフォルト値がわかるだけなので、間違った結果を取らないために INSERT 文を実行した直後にこの値を取り出してください。

ときには、1 つの INSERT 文にグローバル・オートインクリメント型のカラムが複数含まれていることがあります。この場合、戻り値は生成されたデフォルト値のいずれか 1 つですが、そのうちのどの値であるかを判別する信頼できる方法はありません。このような状況を回避するようにデータベースを設計し、INSERT 文を記述することをおすすめします。

getNewUUID メソッド

新しい UUID 値を返します。

構文

UUID **getNewUUID()**

getNotification メソッド

イベント通知を読み込んで、イベント名を返します。

構文

String **getNotification(** String *queue_name*, UInt32 *wait_ms***)**

パラメータ

- **queue_name**　イベント通知キューの名前。デフォルトの接続キューの場合は NULL。
- **wait_ms**　最大待機時間 (ミリ秒)。無期限に待機する場合は、**UL_READ_WAIT_INFINITE** を渡します。

参照

- 「イベント通知の操作」『Ultra Light データベース管理とリファレンス』
- 「createNotificationQueue メソッド」 59 ページ
- 「cancelGetNotification メソッド」 57 ページ
- 「declareEvent メソッド」 59 ページ
- 「destroyNotificationQueue メソッド」 60 ページ
- 「registerForEvent メソッド」 65 ページ
- 「sendNotification メソッド」 67 ページ
- 「triggerEvent メソッド」 69 ページ

getNotificationParameter メソッド

getNotification によって読み込まれたイベント通知のパラメータを取得します。

構文

String **getNotificationParameter(** String *queue_name*, String *param_name***)**

パラメータ

- **queue_name**　イベント通知キューの名前。
- **param_name**　読み込むパラメータの名前。すべてを読み込む場合は "*"。

参照

- 「イベント通知の操作」『Ultra Light データベース管理とリファレンス』
- 「createNotificationQueue メソッド」 59 ページ
- 「cancelGetNotification メソッド」 57 ページ
- 「declareEvent メソッド」 59 ページ
- 「destroyNotificationQueue メソッド」 60 ページ
- 「getNotification メソッド」 62 ページ
- 「registerForEvent メソッド」 65 ページ
- 「sendNotification メソッド」 67 ページ
- 「triggerEvent メソッド」 69 ページ

getSQLPassthroughScriptCount メソッド

実行できる SQL パススルー・スクリプトの数を取得します。実行できる SQL パススルー・スクリプトの数を返します。

構文

UInt32 **getSQLPassthroughScriptCount()**

参照

- 「executeNextSQLPassthroughScript メソッド」 60 ページ
- 「executeSQLPassthroughScripts メソッド」 61 ページ

getTable メソッド

データベース内の要求されたテーブルへの参照を作成して返します。

構文

Table **getTable(**String *name*, String *persistName* **)**

パラメータ

- **name**　フェッチするテーブルの名前。
- **persistName**　ページ間 JavaScript オブジェクト持続性の名前。持続性が必要ない場合 (たとえばアプリケーションに単一の HTML ページしかない場合) は NULL を設定します。

grantConnectTo メソッド

パスワードを指定して、特定のユーザ ID に Ultra Light データベースへのアクセスを許可します。

構文

grantConnectTo(String *uid*, String *pwd***)**

パラメータ

- **uid**　アクセスを許可するユーザ ID。最大長は 16 文字です。
- **pwd**　ユーザ ID のパスワード。

備考

既存のユーザ ID が指定されていれば、この関数を使用してそのユーザのパスワードを更新します。Ultra Light では、最大で 4 人のユーザがサポートされます。

isOpen メソッド

接続が開いている場合は true、閉じている場合は false を返します。

構文

Boolean **isOpen()**;

prepareStatement メソッド

IN パラメータあり、または IN パラメータなしで、SQL 文を事前にコンパイルして PreparedStatement オブジェクトに格納します。

構文

PreparedStatement **prepareStatement(**String *sql*, String *persistName***)**

パラメータ

- **sql**　IN パラメータのプレースホルダ '?' が 1 つまたは複数含まれている可能性のある SQL 文。
- **persistName**　ページ間 JavaScript オブジェクト持続性の名前。持続性が必要ない場合 (たとえばアプリケーションに単一の HTML ページしかない場合) は NULL を設定します。

備考

このオブジェクトを使用すると、SQL 文を効率的に何回も実行できます。

registerForEvent メソッド

イベントの通知を受け取るためのキューを登録します。

構文

void **registerForEvent(**String *event_name*, String *object_name*, String *queue_name*, Boolean *register***)**

パラメータ

- **event_name**　イベントの名前。
- **object_name**　イベントが適用されるオブジェクトの名前 (テーブル名など)。
- **queue_name**　イベント通知キューの名前。
- **register**　登録する場合は TRUE、登録解除する場合は FALSE。

参照

- 「イベント通知の操作」 『Ultra Light データベース管理とリファレンス』
- 「createNotificationQueue メソッド」 59 ページ
- 「cancelGetNotification メソッド」 57 ページ
- 「declareEvent メソッド」 59 ページ
- 「destroyNotificationQueue メソッド」 60 ページ
- 「getNotification メソッド」 62 ページ
- 「sendNotification メソッド」 67 ページ
- 「triggerEvent メソッド」 69 ページ

resetLastDownloadTime メソッド

最後のダウンロードの時刻をリセットします。

構文

void **resetLastDownloadTime(**String *pub-list***)**

パラメータ

- **pub-list**　リセットするパブリケーション名のカンマ区切りのリスト。

revokeConnectFrom メソッド

指定されたユーザ ID に対して、Ultra Light データベースへのアクセス権を取り消します。

構文

revokeConnectFrom(String *uid* **)**

パラメータ

- **uid**　データベース・アクセスから除外されるユーザ ID。最大長は 16 文字です。

rollback メソッド

データベースへの未処理の変更をロールバックします。

構文

rollback( )

rollbackPartialDownload メソッド

失敗した同期からの変更をロールバックします。

構文

rollbackPartialDownload( )

備考

同期のダウンロード時に通信エラーが発生した場合、Ultra Light では、ダウンロードした変更を適用し、同期が中断した時点から同期を再開することができます。ダウンロードした変更が不要な場合 (ダウンロードが中断した時点での再開を望まない場合)、RollbackPartialDownload を使用することで、失敗したダウンロード・トランザクションをロールバックします。

saveSyncParms メソッド

HotSync または ワンタッチ同期で使用される同期パラメータを保存します。

構文

saveSyncParms( )

備考

saveSyncParms メソッドと Connection.SyncParms プロパティを混同しないでください。SyncParms プロパティは、この接続の同期パラメータを定義するために使用されます。saveSyncParms メソッドは、HotSync で使用できるようにそれらのパラメータを保存するだけです。

参照

- 「ワンタッチ同期」 30 ページ

sendNotification メソッド

指定された名前に一致するすべてのキュー (現在の接続におけるキューを含む) に通知を送信します。送信した通知の数 (一致するキューの数) を返します。

構文

UInt32 **sendNotification(** String *queue_name*, String *event_name*, String *parms***)**

パラメータ

- **queue_name**　イベント通知キューの名前。
- **event_name**　イベントの名前。
- **parms**　通知のパラメータ文字列 (通知がある場合)。name=value というフォーマットに従います。

参照

- 「イベント通知の操作」 『Ultra Light データベース管理とリファレンス』
- 「createNotificationQueue メソッド」 59 ページ
- 「cancelGetNotification メソッド」 57 ページ
- 「declareEvent メソッド」 59 ページ
- 「destroyNotificationQueue メソッド」 60 ページ
- 「getNotification メソッド」 62 ページ
- 「registerForEvent メソッド」 65 ページ
- 「triggerEvent メソッド」 69 ページ

setDatabaseID メソッド

グローバル・オートインクリメント・カラムに使用するデータベース ID の値を設定します。

構文

setDatabaseID(UInt32 *value* **)**

パラメータ

- **value**　データベース ID の値。**value** は、**[0,0x0fffffff]** の範囲内であることが必要です。

startSynchronizationDelete メソッド

同期用に、この接続によって行われる以後のすべての削除にマークを付けます。

構文

startSynchronizationDelete( )

備考

この関数が呼び出されると、すべての削除操作がもう一度同期されます。

stopSynchronizationDelete メソッド

この関数が呼び出されると、削除操作が同期されなくなります。

構文

stopSynchronizationDelete( )

備考

領域を節約するために、Ultra Light データベースから古い情報を削除して、統合データベースにはそれを残しておく場合に使用すると便利です。

synchronize メソッド

現在の SyncParms オブジェクトを使用してデータベースの同期をとります。

構文

synchronize( )

備考

結果の詳細なステータスは、この接続の SyncResult オブジェクトでレポートされます。この接続の Connection.SyncParms オブジェクトで定義される同期プロパティを使用して、同期が実行されます。

synchronizeWithParm メソッド

指定された SyncParms オブジェクトを使用してデータベースの同期をとります。

構文

synchronizeWithParm(SyncParms *parms***)**

パラメータ

- **parms**　この同期で使用される SyncParms オブジェクト。

備考

このメソッドでは、接続間で同期パラメータを共有できます。

結果の詳細なステータスは、この接続の SyncResult オブジェクトでレポートされます。

triggerEvent メソッド

イベントをトリガして、登録されたすべてのキューに通知を送信します。送信したイベント通知の数を返します。

構文

UInt32 **triggerEvent(** String *event_name*, String *parms***)**

パラメータ

- **event_name**　イベント通知キューの名前。
- **parms**　追加パラメータ。

参照

- 「イベント通知の操作」 『Ultra Light データベース管理とリファレンス』
- 「createNotificationQueue メソッド」 59 ページ
- 「cancelGetNotification メソッド」 57 ページ
- 「declareEvent メソッド」 59 ページ
- 「destroyNotificationQueue メソッド」 60 ページ
- 「getNotification メソッド」 62 ページ
- 「registerForEvent メソッド」 65 ページ
- 「sendNotification メソッド」 67 ページ

validateDatabase メソッド

この接続でのデータベースを検証します。

構文

void **validateDatabase(** UInt16 *type*, String *tablename***)**

パラメータ

- **type**　実行する検証のタイプ。「プロパティ」 75 ページを参照してください。
- **tablename**　検証する特定のテーブルの名前。NULL の場合はデータベース全体を検証します。

備考

このメソッドを使用して、特定のテーブルまたはデータベース全体を検証できます。

ConnectionParms クラス

Ultra Light データベースへの接続を開くためのパラメータを指定します。

データベースは、1 人の認証済みユーザ **DBA** で作成されます。このユーザの最初のパスワードは **sql** です。デフォルトでは、ユーザ ID **DBA** とパスワード **sql** を使用して、接続が開かれます。デフォルトのユーザを無効にするには、Connection.revokeConnectFrom を使用します。ユーザを追加したりユーザのパスワードを変更するには、Connection.grantConnectTo を使用します。

現在のところ、一度に開くことができる接続は 1 つのみです。一度にアクティブにできるデータベースは 1 つのみです。他の接続が開いているときに別のデータベースへの接続を開こうとすると、エラーが発生します。

プロパティ

このクラスのプロパティは、次のとおりです。

プロトタイプ	説明
String additionalParms (読み込み／書き込み)	セミコロンで区切られた **name**=*value* のペアとして指定される追加のパラメータ。
String cacheSize (読み込み／書き込み)	キャッシュのサイズ。CacheSize の値はバイト単位で指定します。キロバイトの単位を示すにはサフィックス k または K を使用し、メガバイトの単位を示すにはサフィックス M または m を使用します。 「Ultra Light CACHE_SIZE 接続パラメータ」『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。
String connectionName (読み込み／書き込み)	接続の名前。接続名は、複数の Web ページ間で 1 つの接続を共有するために使用されます。 「Ultra Light CON 接続パラメータ」『Ultra Light データベース管理とリファレンス』と「ページ間の接続とアプリケーション状態の管理」 12 ページを参照してください。
String creatorIdOnPalm	Palm デバイス上の Ultra Light データベースの作成者 ID。
String databaseOnCE (読み込み／書き込み)	Windows Mobile 上のデータベースのファイル名。 「Ultra Light CE_FILE 接続パラメータ」『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。
String databaseOnDesktop (読み込み／書き込み)	Windows に配備されるデータベースのファイル名。 「Ultra Light DBF 接続パラメータ」『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。

プロトタイプ	説明
String databaseOnPalm (読み込み／書き込み)	Palm 上の Ultra Light データベースのファイル名。 「Ultra Light PALM_FILE 接続パラメータ」『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。
String encryptionKey (読み込み／書き込み)	データベースを暗号化するためのキー。OpenConnection は、データベース作成中に指定されたキーと同じキーを使用する必要があります。キーは以下の条件を満たしていることが推奨されます。 1. 任意の長い文字列を選択します。 2. キーを見破られる可能性を減らすため、多様な数字、文字、特殊文字を使用した文字列を選択します。 「Ultra Light DBKEY 接続パラメータ」『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。
String password (読み込み／書き込み)	認証ユーザのパスワード。データベースは最初、1 つの認証されたユーザ (**DBA**) とパスワード **sql** を使用して、作成されます。データベースで大文字と小文字が区別されない場合は、パスワードの大文字と小文字は区別されません。データベースで大文字と小文字が区別される場合は、パスワードの大文字と小文字も区別されます。デフォルト値は *sql* です。 「Ultra Light PWD 接続パラメータ」『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。
String userID (読み込み／書き込み)	データベースで認証されたユーザ。データベースは最初、1 つの認証されたユーザ DBA を使用して、作成されます。UserID では大文字と小文字は区別されません。デフォルト値は *DBA* です。 「Ultra Light UID 接続パラメータ」『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。

toString メソッド

認証ステータス・コード定数の文字列名を生成します。

構文

String **toString()**;

戻り値

コードの名前、または認識されたコードでない場合は **unknown**。

CreationParms クラス

Ultra Light データベースの作成時に指定できるパラメータを定義します。

一部の Ultra Light データベース・オプションは、データベースの作成時に設定する必要があります。createDatabase メソッドを使用してデータベースを作成するときは、以下のパラメータを指定できます。「createDatabase メソッド」 76 ページを参照してください。

プロパティ

このクラスのプロパティは、次のとおりです。対応する説明の詳細については、「Ultra Light で使用するデータベース作成パラメータの選択」 『Ultra Light データベース管理とリファレンス』を参照してください。

プロトタイプ	説明
Boolean caseSensitive	Ultra Light データベースの文字列を比較するときに大文字と小文字を区別するかどうかを設定します。
UInt32 checksumLevel	データベースのチェックサム検証のレベルを設定します。 デフォルトは 0 です。
String dateFormat	日付がデータベースから取り出されるときのデフォルトの文字列フォーマットを設定します。
String dateOrder	年、月、日の日付の順序を解釈する方法を制御します。
UInt32 maxHashSize	Ultra Light のインデックスのハッシュに使用する最大バイト数を設定します。 デフォルトは 0 です。
UInt32 nearestCentury	文字列から日付への変換で、2 桁の年の解釈を制御します。
Boolean obfuscate	データベースのデータを難読化するかどうか制御します。難読化は単純暗号化です。
UInt32 pageSize	データベース・ページ・サイズを定義します。有効な値は、1024、2048、4096、8192、16384 です。 デフォルトは 4096 です。
UInt32 precision	10 進法計算での結果の最大桁数を指定します。
UInt32 scale	計算結果が最大 precision にトランケートされる場合の、小数点以下の最小桁数を指定します。

プロトタイプ	説明
String timeFormat	データベースから取り出した時刻のフォーマットを設定します。
String timestampFormat	Ultra Light でのタイムスタンプのフォーマットを指定します。
String timestampIncrement	Ultra Light でのタイムスタンプのトランケート方法を指定します。
Boolean utf8Encoding	Unicode 用の 8 ビット・マルチバイト・エンコードである UTF-8 フォーマットでデータをエンコードします。

DatabaseManager クラス

Ultra Light データベースへの接続を管理します。

接続は、他の操作の実行前に開きます。また、その接続での操作がすべて終了したら、接続を閉じてからアプリケーションを終了します。接続で開いたテーブルをすべて閉じてから、その接続を閉じます。

プロパティ

このクラスのプロパティは、次のとおりです。

プロパティ	説明
AuthStatusCode **AuthStatusCode** (読み込み専用)	最後の同期に関連付けられた AuthStatusCode オブジェクトを取得します。
String **directory** (読み込み専用)	M-Business Anywhere が実行しているディレクトリ。 Palm OS の場合は NULL です。
UInt32 **runtimeType** (読み込み専用)	ランタイム・タイプ。Ultra Light ランタイム (スタンドアロン) ライブラリまたは Ultra Light データベース・エンジンのいずれかです。値は一覧で、次のいずれかです。 ● DatabaseManager.UL_STANDALONE ● DatabaseManager.UL_ENGINE_CLIENT
Int32 **sqlCode** (読み込み専用)	最後の操作に関連付けられた SQL コード値を取得します。
SQLError **SQLError** (読み込み専用)	SQLError オブジェクトを取得します。
SQLType **SQLType** (読み込み専用)	SQLType オブジェクトを取得します。
PODSUInt32 **UL_STANDALONE** (読み込み専用)	ランタイム・タイプが Ultra Light ランタイム・ライブラリであることを示す定数。
PODSUInt32 **UL_ENGINE_CLIENT** (読み込み専用)	ランタイム・タイプが Ultra Light データベース・エンジンであることを示す定数。

プロパティ	説明
UInt16 **VALIDATE_EXPRESS** (読み込み専用)	validateDatabase メソッドでエクスプレス検証 (**VALIDATE_FULL** 検証ほど完全ではないが、高速) を指定するために使用する定数。
UInt16 **VALIDATE_FULL** (読み込み専用)	validateDatabase メソッドで完全な検証 (テーブル、インデックス、データベース・ページの検証) を指定するために使用する定数。

createDatabase メソッド

データベースを作成し、**access_parms** によって指定されたデータベースへの接続を開きます。

構文

```
Connection createDatabase(
String access_parms ,
PODSArray *coll_bytes,
String create_parms
)
```

パラメータ

- **access_parms**　データベースに接続するためのパラメータ。access_parms は、接続パラメータ (データベースのファイル名やロケーションなど) を指定したり、接続を開いたりするために使用されます。

  「Ultra Light データベースへの接続」『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。

- **coll_bytes**　作成されるデータベースで使用するデータベース照合を定義するバイト配列。Ultra Light にはいくつかのソース・ファイルが付属しています。これらは *install-dir¥UltraLite¥Collations¥js¥* にある JavaScript ソース・ファイル (*.js*) で、ファイル名の形式は **coll_XXXXX.js** です。XXXXX は照合名を表します。例：*coll_1250LATIN2.js*。

  照合ファイルは、メインの html ファイルでデータベース論理の前にインクルードする必要があります。バイト配列変数は coll_XXXXX.js ファイルで定義されます。

- **create_parms**　データベースを作成するためのパラメータ。パラメータ・キーワードは大文字と小文字を区別しませんが、ほとんどの値は大文字と小文字を区別します。create_parms は、データベースの作成時のみ指定できるパラメータを指定するために使用されます。

  「Ultra Light で使用するデータベース作成パラメータの選択」『Ultra Light データベース管理とリファレンス』を参照してください。

戻り値

戻り値なし。

備考

すでにデータベースが存在する場合は、SQLE_DATABASE_NOT_CREATED 例外がスローされます。

一度にアクティブにできるデータベースは 1 つのみです。データベースへの接続が開いているときに、別のデータベースへの接続を開こうとすると、エラーが発生します。

dropDatabase メソッド

指定されたデータベースを削除します。

構文

void **dropDatabase(**String *parms***)**

パラメータ

- **parms**　データベースを識別するためのパラメータ。

備考

parms は、keyword=value のペアがセミコロンで区切られたリスト (**"param1=value1;param2=value2"**) です。パラメータ・キーワードは大文字と小文字を区別しませんが、ほとんどの値は大文字と小文字を区別します。

接続が開かれているデータベースを削除することはできません。

getDatabaseOptions メソッド

構文

Connection **openConnection(** String *parms***)**

openConnection メソッド

parms によって指定されたデータベースへの接続を開きます。

構文

Connection **openConnection(** String *parms***)**

パラメータ

- **parms**　接続を開くためのパラメータ (keyword=value の組み合わせのセット) を保持する文字列。パラメータ・キーワードは大文字と小文字を区別しませんが、ほとんどの値は大文字と小文字を区別します。

戻り値

開かれた接続。

備考

データベースが存在しない場合は、エラーがスローされます。エラーをキャッチするコード内で Connection.sqlCode をチェックすると、エラーの原因を特定できます。

一度にアクティブにできるデータベースは 1 つのみです。他の接続が開いているときに別のデータベースを作成しようとすると、エラーが発生します。

reOpenConnection メソッド

開かれた Connection オブジェクトを返します。

構文

Connection **reOpenConnection(** String *connectionName* **)**

パラメータ

- **connectionName**　ConnectionParms.connectionName プロパティで指定される、もう一度開く接続の名前。

戻り値

このメソッドを使用して、複数の Web ページ間の接続を管理します。

validateDatabase メソッド

データベースに現在接続されていない Ultra Light データベースを検証します。

構文

void **validateDatabase(**
String *start_parms*;
UInt16 *type*)

パラメータ

- **start_parms**　データベースに接続するためのパラメータ。
- **type**　実行する検証のタイプ。DatabaseManager クラス 「プロパティ」 75 ページ の **VALIDATE_EXPRESS** プロパティと **VALIDATE_FULL** プロパティを参照してください。

DatabaseSchema クラス

Ultra Light データベースのスキーマを表します。**DatabaseSchema** オブジェクトは、接続にアタッチされ、接続が開いている間のみ有効です。

定数

定数	説明
SYNC_ALL_DB	データベース内のすべてのテーブルを同期します (非同期として定義されているテーブルを除く)。
SYNC_ALL_PUBS	データベース内のすべてのパブリケーションを同期します。

このクラスのメンバは、次のとおりです。

getCollationName メソッド

このデータベースで使用される文字セットとソート順を示す文字列を返します。

構文

String **getCollationName()**

getDatabaseProperty メソッド

指定されたデータベースのプロパティの値を返します。

構文

String **getDatabaseProperty(**String *name***)**

パラメータ

- **name**　データベースのプロパティの名前。

備考

識別されるプロパティは次のとおりです。

- **"date_format"**　データベースによる文字列変換に使用される日付フォーマット。
- **"date_order"**　データベースによる文字列変換に使用される日付順。
- **"nearest_century"**　データベースによる文字列変換に使用される最も近い世紀。
- **"precision"**　データベースによる文字列変換に使用される浮動小数点の精度。

- **"scale"**　データベースによる文字列変換中に、計算結果が最大 precision にトランケートされるときの小数点以下の最小桁数。
- **"time_format"**　データベースによる文字列変換に使用される時刻フォーマット。
- **"timestamp_format"**　データベースによる文字列変換に使用されるタイムスタンプのフォーマット。
- **"timestamp_increment"**　2 つのユニークなタイムスタンプ間のマイクロ秒 (1,000,000 分の 1 秒) 単位の最小差。

getDateFormat メソッド

文字列変換に使用される日付フォーマットを返します。

構文

String **getDateFormat()**

getDateOrder メソッド

文字列変換に使用される日付順を返します。

構文

String **getDateOrder()**

getNearestCentury メソッド

文字列変換に使用される最も近い世紀を返します。

構文

String **getNearestCentury()**

getPrecision メソッド

文字列変換に使用される浮動小数点の精度を返します。

構文

String **getPrecision( )**

getPublicationCount メソッド

データベース内のパブリケーションの数を返します。

構文

UInt16 **getPublicationCount( )**

備考

パブリケーション ID の範囲は、1 ～ **getPublicationCount()** です。

注意：パブリケーションの ID とカウントは、スキーマのアップグレード中に変更されることがあります。パブリケーションを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getPublicationName メソッド

指定されたパブリケーション ID で識別されたパブリケーションの名前を返します。

構文

String **getPublicationName(** UInt16 *pubID* **)**

パラメータ

- **pubID**　パブリケーションの ID。**pubID** は、**[1,getPublicationCount()]** の範囲内であることが必要です。

備考

注意：パブリケーションの ID とカウントは、スキーマのアップグレード中に変更されることがあります。パブリケーションを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getPublicationSchema メソッド

指定したパブリケーションに対応するパブリケーション・スキーマを返します。

構文

PublicationSchema **getPublicationSchema(** String *name* **)**

パラメータ

- **name**　パブリケーションの名前。

getSignature メソッド

このデータベースのシグネチャを返します。

構文

String **getSignature( )**

getTableAGDBSet メソッド

指定されたテーブル名を使用して、AGDBSet オブジェクトにバインドします。TableAGDBSet オブジェクトのインスタンスを返します。

構文

TableAGDBSet **getTableAGDBSet(** String *tablename***)**

パラメータ

- **tablename**　AGDBSet にバインドするテーブルの名前。

備考

AGDB オブジェクトのマニュアルについては、「M-Business Anywhere API リファレンス」を参照してください。

getTableCount メソッド

データベース内のテーブルの数を返します。

構文

UInt16 **getTableCount( )**

戻り値

テーブル数。接続が開いていない場合は 0。

備考

テーブル ID の範囲は、1 ～ **getTableCount()** です。

getTableName メソッド

指定されたテーブル ID で識別されたテーブルの名前を返します。

構文

String **getTableName(** UInt16 *tableID***)**

パラメータ

- **tableID**　テーブルの ID。**tableID** は、**[1,getTableCount()]** の範囲内であることが必要です。

備考

注意：テーブル ID は、スキーマのアップグレード中に変更されることがあります。テーブルを正しく識別するには、名前でアクセスするか、キャッシュされている ID をスキーマのアップグレード後にリフレッシュします。

getTimeFormat メソッド

文字列変換に使用される時刻フォーマットを返します。

構文

String **getTimeFormat()**

getTimestampFormat メソッド

文字列変換に使用されるタイムスタンプのフォーマットを返します。

構文

String **getTimestampFormat()**

isCaseSensitive メソッド

データベースで大文字と小文字が区別される場合は true、区別されない場合は false を返します。

構文

Boolean **isCaseSensitive()**

isOpen メソッド

データベース・スキーマが開いている場合は true、閉じている場合は false を返します。

構文

Boolean **isOpen()**

IndexSchema クラス

Ultra Light テーブルのインデックスのスキーマを表します。

このオブジェクトを直接インスタンス化することはできません。インデックス・スキーマは、TableSchema.getPrimaryKey、TableSchema.getIndex、TableSchema.getOptimalIndex の各メソッドを使用して作成されます。

getColumnCount メソッド

インデックス内のカラム数を返します。

構文

UInt16 **getColumnCount( )**

備考

インデックス内のカラム ID の範囲は、1 ～ getColumnCount() です。

getColumnName メソッド

このインデックス内の *colIDInIndex* カラムの名前を返します。

構文

String **getColumnName(**UInt16 *colIDInIndex* **)**

パラメータ

- **colIDInIndex**　カラムのこのインデックス内の ID。colIDInIndex は、[1, getColumnCount()] の範囲内であることが必要です。

getName メソッド

このインデックスの名前を返します。

構文

String **getName()**

getReferencedIndexName メソッド

このインデックスが外部キーである場合、参照されるプライマリ・インデックスの名前を返します。

構文

String **getReferencedIndexName()**

getReferencedTableName メソッド

インデックスが外部キーである場合、参照されるプライマリ・テーブルの名前を返します。

構文

String **getReferencedTableName()**

isColumnDescending メソッド

カラムが降順で使用される場合は true、昇順で使用される場合は false を返します。

構文

Boolean **isColumnDescending(**String *name***)**

パラメータ

- **name**　カラムの名前。

isForeignKey メソッド

インデックスが外部キーである場合は true、外部キーでない場合は false を返します。

構文

Boolean **isForeignKey()**

備考

外部キー内のカラムは、別のテーブルの NULL 以外のユニーク・インデックスを参照することができます。

isForeignKeyCheckOnCommit メソッド

参照整合性がコミット時にチェックされる場合は true、挿入時および更新時にチェックされる場合は false を返します。

構文

Boolean **isForeignKeyCheckOnCommit()**

isForeignKeyNullable メソッド

この外部キーが NULL 入力可であれば true、NULL 入力不可であれば false を返します。

構文

Boolean **isForeignKeyNullable()**

isPrimaryKey メソッド

インデックスがプライマリ・キーである場合は true、プライマリ・キーでない場合は false を返します。

構文

Boolean **isPrimaryKey()**

備考

プライマリ・キー内のカラムでは NULL は許可されません。

isUniqueIndex メソッド

インデックスがユニークである場合は true、ユニークでない場合は false を返します。

構文

Boolean **isUniqueIndex()**

備考

ユニークなインデックス内のカラムは NULL であることがあります。

isUniqueKey メソッド

インデックスがユニーク・キーである場合は true、ユニーク・キーでない場合は false を返します。

構文

Boolean **isUniqueKey()**

備考

ユニーク・キー内のカラムでは NULL は許可されません。

PreparedStatement クラス

IN パラメータあり、または IN パラメータなしで事前にコンパイルされた SQL 文を表します。実行時に Connection.prepareStatement を使用して作成されます。

このオブジェクトを使用すると、この文を効率的に何回も実行できます。

準備文が閉じられると、それに関連する ResultSet オブジェクトと ResultSetSchema オブジェクトもすべて閉じられます。リソース管理の理由により、準備文を使用し終わったら、その準備文を明示的に閉じることをおすすめします。

appendBytesParameter メソッド

指定されたバイト配列の指定されたサブセットを、指定された SQLType.LONGBINARY カラムの新しい値に追加します。

構文

appendBytesParameter(
  UInt16 *parameterID*,
  Array *value*,
  UInt32 *srcOffset*,
  UInt32 *count*
)

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの現在の新しい値に追加する値。
- **srcOffset**　ソース配列の開始位置。
- **count**　コピーされるバイト数。

備考

配列 **value** の srcOffset (0 から始まります) から srcOffset+count-1 までの位置のバイトが、指定されたパラメータの値に追加されます。挿入時には、**insertBegin** は新しい値をパラメータのデフォルト値に初期化します。

次のいずれかに該当する場合、コード SQLError.SQLE_INVALID_PARAMETER とともにエラーがスローされ、追加先は修正されません。

- **value** 引数が NULL である
- **srcOffset** 引数が負の値である
- **count** 引数が負の値である
- **srcOffset+count** がソース配列の長さ **value.length** よりも大きい

appendStringChunkParameter メソッド

文字列を指定された SQLType.LONGVARCHAR の新しい値に追加します。

構文

appendStringChunkParameter(
UInt16 *parameterID*,
String *value*,
)

パラメータ

- **parameterID** パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value** パラメータの現在の新しい値に追加する値。

例

次の文は、文字列 **XYZ** のインスタンス 100 個を最初のパラメータに追加します。

```
for ( I = 0; I < 100; I++ ){
  stmt.appendStringChunkParameter( 1, "XYZ" );
}
```

close メソッド

準備文を閉じます。

構文

close( )

備考

準備文が閉じられると、それに関連する ResultSet オブジェクトと ResultSetSchema オブジェクトもすべて閉じられます。

preparedStatement オブジェクトを閉じたら、ただちに NULL に設定することをおすすめします。

executeQuery メソッド

SQL SELECT 文を実行し、結果セットを返します。

構文

ResultSet **executeQuery(** String *persistName* **)**

パラメータ

- **persistName** ページ間 JavaScript オブジェクト持続性の名前。持続性が必要ない場合 (たとえばアプリケーションに単一の HTML ページしかない場合) は NULL を設定します。

戻り値

クエリの結果セット (ローのセット)。

executeStatement メソッド

SQL INSERT 文、DELETE 文、UPDATE 文のように、結果セットを返さない文を実行します。

構文

Int32 **executeStatement( )**

戻り値

文の影響を受けるローの数。

備考

Connection.autoCommit が true の場合は、文が 1 つまたは複数のローに影響するときだけ、文がコミットされます。

getPlan メソッド

クエリを実行するのに Ultra Light が使用するアクセス・プランを記述する文字列を返します。

構文

String **getPlan( )**

備考

このメソッドは、主に開発中の使用を目的とします。

参照

- 「Ultra Light の実行プラン」 『Ultra Light データベース管理とリファレンス』

getResultSetSchema メソッド

このクエリ文の結果セットを記述するスキーマを返します。

構文

ResultSetSchema **getResultSetSchema()**

hasResultSet メソッド

この文が実行されたときに結果セットが生成される場合は true、生成されない場合は false を返します。

構文

Boolean **hasResultSet( )**

isOpen メソッド

準備文が開いている場合は true、閉じている場合は false を返します。

構文

Boolean **isOpen( )**

setBooleanParameter メソッド

指定されたパラメータの値を、Boolean を使用して設定します。

構文

setBooleanParameter(UInt16 *parameterID*, Boolean *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を設定します。

```
stmt.setBooleanParameter(1, false);
```

setBytesParameter メソッド

指定されたパラメータの値を、バイト配列を使用して設定します。

構文

setBytesParameter(UInt16 *parameterID*, Array *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

備考

SQLType.BINARY 型、SQLType.LONGBINARY 型のカラムにのみ適しています。

例

次の文は、最初のパラメータの値を設定します。

```
var blob = new Array( 3 );
blob[ 0 ] = 78;
blob[ 1 ] = 0;
blob[ 2 ] = 68;
stmt.setBytesParameter( 1, blob );
```

setDateParameter メソッド

指定した SQLType.DATE 型のパラメータの値を、日付を使用して設定します。

構文

setDateParameter(UInt16 *parameterID*, Date *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

備考

Date オブジェクトの年、月、日のフィールドだけが関係します。

例

次の文は、最初のパラメータの値を 2004/09/27 に設定します。

```
stmt.setDateParameter(
    1, new Date( 2004,9,27,0,0,0,0 )
);
```

setDoubleParameter メソッド

指定されたパラメータの値を、**double** を使用して設定します。

構文

setDoubleParameter(UInt16 *parameterID*, Double *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を設定します。

```
stmt.setDoubleParameter( 1, Number.MAX_VALUE );
```

setFloatParameter メソッド

指定した SQLType.REAL パラメータの値を設定します。

構文

setFloatParameter(UInt16 *parameterID*, Float *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの浮動小数点値を設定します。

```
stmt.setFloatParameter( 1,
    (2 - Math.pow(2,-23)) * Math.pow(2,127)
);
```

setIntParameter メソッド

指定されたパラメータの値を、UInt16 を使用して設定します。

構文

setIntParameter(UInt16 *parameterID*, UInt16 *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を **2147483647** に設定します。

```
stmt.setIntParameter( 1, 2147483647 );
```

setLongParameter メソッド

指定されたパラメータの値を設定します。

構文

setLongParameter(UInt16 *parameterID*, Int64 *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を **9223372036854770000** に設定します。

```
stmt.setLongParameter( 1, 9223372036854770000 );
```

setNullParameter メソッド

指定されたパラメータの値を SQL NULL に設定します。

構文

setNullParameter(UInt16 *parameterID* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。

setShortParameter メソッド

指定されたパラメータの値を設定します。

構文

setUInt16Parameter(UInt16 *parameterID*, UInt16 *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を **32767** に設定します。

```
stmt.setShortParameter( 1, 32767 );
```

setStringParameter メソッド

指定されたパラメータの値を設定します。

構文

setStringParameter(UInt16 *parameterID*, String *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を **ABC** に設定します。

```
stmt.setStringParameter( 1, "ABC" );
```

setTimeParameter メソッド

指定された SQLType.TIME 型のパラメータの値を、日付を使用して設定します。

構文

setTimeParameter(UInt16 *parameterID*, Date *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

備考

Date オブジェクトの時、分、秒のフィールドだけが関係します。

例

次の文は、最初のパラメータの値を 18:02:13:0000 に設定します。

```
stmt.setTimeParameter(
    1, new Date( 1966,4,1,18,2,13,0 )
);
```

setTimestampParameter メソッド

指定したパラメータの値を、**Timestamp** を使用して設定します。

構文

setTimestampParameter(UInt16 *parameterID*, Date *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

例

次の文は、最初のパラメータの値を 1966/04/01 18:02:13:0000 に設定します。

```
stmt.setTimestampParameter(
    1, new Date( 1966,4,1,18,2,13,0 )
);
```

setULongParameter メソッド

指定されたパラメータの値を、符号なし値として扱われる Double を使用して設定します。

構文

setULongParameter(UInt16 *parameterID*, UInt64 *value* **)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。64 ビット符号なし整数値を表す Double を使用します。

備考

Unsigned64 クラスを参照してください。

例

次の文は、最初のパラメータの値を設定します。

```
stmt.setLongParameter( 1, 9223372036854770000 * 4096 );
```

setUUIDParameter メソッド

指定されたパラメータの値を、**UUID** を使用して設定します。

構文

setUUIDParameter(UInt16 *parameterID*, UUID *value***)**

パラメータ

- **parameterID**　パラメータの ID 番号。結果セットの最初のパラメータの ID 値は 1 です。
- **value**　パラメータの新しい値。

PublicationSchema クラス

Ultra Light パブリケーションのスキーマを表します。

このクラスを直接インスタンス化することはできません。パブリケーションのスキーマは DatabaseSchema.getPublicationSchema メソッドを使用して作成されます。

パブリケーションは、名前で識別されます。一部のメソッドでは、パブリケーション名のリストをカンマ区切りのリストで指定する必要があります。

特別な 2 つの定数値は、DatabaseSchema オブジェクトによって提供されます。**SYNC_ALL_DB** は、データベース全体に対応します。**SYNC_ALL_PUBS** は、すべてのパブリケーションに対応します。

getName メソッド

このパブリケーションの名前を返します。

構文

String **getName()**

ResultSet クラス

Ultra Light データベースの結果セットを表します。実行時に PreparedStatement.executeQuery を使用して作成されます。

プロパティ

このクラスのプロパティは、次のとおりです。

プロパティ	説明
ResultSetSchema schema (読み込み専用)	この結果セットのスキーマ。このプロパティが有効なのは、その準備文が開かれている間だけです。
NULL_TIMESTAMP_VAL	タイムスタンプ値が NULL であることを示す定数。

appendBytes メソッド

指定されたバイト配列の指定されたサブセットを、指定された SQLType.LONGBINARY カラムの新しい値に追加します。

構文

```
appendBytes(
  UInt16 columnID,
  Array value,
  UInt32 srcOffset,
  UInt32 count
)
```

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。
- **srcOffset**　カラムの現在の新しい値に追加する値。
- **count**　コピーされるバイト数。

備考

配列 **value** の srcOffset (0 から始まります) から **srcOffset+count-1** までの位置のバイトが、指定されたカラムの値に追加されます。挿入時には、insertBegin は新しい値をカラムのデフォルト値に初期化します。ローのデータは、**insert** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

次のいずれかに該当する場合、コード SQLCode.SQLE_INVALID_PARAMETER とともにエラーがスローされ、追加先は修正されません。

- **value** 引数が NULL である
- **srcOffset** 引数が負の値である
- **count** 引数が負の値である
- **srcOffset+count** がソース配列の長さ **value.length** よりも大きい

その他のエラーの場合は、それに応じたエラー・コードとともに **SQLException** がスローされます。

appendStringChunk メソッド

指定された文字列を指定された SQLType.LONGVARCHAR カラムの新しい値に追加します。

構文

```
appendStringChunk(
  UInt16 columnID,
  String value
)
```

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

例

次の文は、文字列 **XYZ** のインスタンス 100 個を最初のカラムの値に追加します。

```
for ( I = 0; I < 100; I++ ){
  t.AppendStringChunk( 1, "XYZ" );
}
```

close メソッド

このオブジェクトに関連付けられているリソースを解放します。

構文

close()

deleteRow メソッド

現在の行を削除します。

構文

deleteRow( )

備考

deleteRow を行う前に updateBegin を呼び出してください。

getAGDBSet メソッド

結果セットから AGDBSet オブジェクトにバインドします。TableAGDBSet オブジェクトのインスタンスを返します。

構文

AGDBSet **getAGDBSet()**

備考

AGDB オブジェクトのマニュアルについては、「M-Business Anywhere API リファレンス」を参照してください。

getBoolean メソッド

指定されたカラムの値を Boolean として返します。

構文

Boolean **getBoolean(** UInt16 *index* **)**

パラメータ

- **index**　カラムの ID 番号。結果セットの最初のカラムの ID は 1 です。

getBytes メソッド

指定されたカラムの値のバイト配列を返します。

構文

Array **getBytes(** UInt16 *index* **)**

パラメータ

- **index**　カラムの ID 番号。結果セットの最初のカラムの ID は 1 です。

備考

SQLType.BINARY 型、SQLType.LONGBINARY 型のカラムの場合にのみ有効です。

getBytesSection メソッド

指定されたソース・オフセットで始まる、指定された SQLType.LONGBINARY または SQLType.BINARY カラムの内容のサブセットを、コピー先のバイト配列の指定されたオフセットにコピーします。

構文

```
UInt32 getBytesSection(
  UInt16 index,
  UInt32 srcOffset,
  Array  dst,
  UInt32 dstOffset,
  UInt32 count
)
```

パラメータ

index　バイナリ・データを含むカラムの 1 から始まる順序。

srcOffset　ソース・バイト配列への、0 から始まる相対オフセット。ソース・オフセットは、0 以上であることが必要です。それ以外の場合は、SQLE_INVALID_PARAMETER エラーが発生します。64K を超えるバッファも許されます。

dst　コピー先バイト配列。

dstOffset　コピー先バイト配列への、0 から始まる相対オフセット。コピー先オフセットは、0 以上であることが必要です。それ以外の場合は、SQLE_INVALID_PARAMETER エラーが発生します。64K を超えるバッファも許されます。

count　移動するバイト数。count は 0 以上であることが必要です。

戻り値

読み込まれたバイト数。

備考

コピー元の配列の srcOffset (0 から始まります) から srcOffset+count-1 までの位置のバイトが、コピー先の配列の dstOffset から dstOffset+count-1 までの位置に、それぞれコピーされます。指定されたバイト数がコピーされる前に、ソース値の末尾が検出された場合は、コピー先の配列の残りは変更されないままになります。

次のいずれかに該当する場合、エラーがスローされ、SQLError コードが SQLE_INVALID_PARAMETER に設定され、コピー先は修正されません。

- dst 引数が NULL である
- srcOffset 引数が負の値である
- dstOffset 引数が負の値である
- count 引数が負の値である
- dstOffset + count がコピー先の配列の長さ dst.length よりも長い

エラー・セット

SQLE_CONVERSION_ERROR　カラムのデータ型が BINARY でも LONG BINARY でもない場合、エラーが発生します。

SQLE_INVALID_PARAMETER　カラムのデータ型が BINARY でオフセットが 0 でも 1 でもない、またはデータ長が 0 より小さい場合、エラーが発生します。

カラムのデータ型が LONG BINARY で、オフセットが 1 より小さい場合も、エラーが発生します。

getDate メソッド

Date として値を返します。

構文

Date **getDate(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getDouble メソッド

Double として値を返します。

構文

Double **getDouble(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getFloat メソッド

指定されたカラムの値を返します。

構文

Float **getFloat(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getInt メソッド

指定されたカラムの値を返します。

構文

UInt32 **getInt(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getLong メソッド

指定されたカラムの値を返します。

構文

Int64 **getLong(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getRowCount メソッド

結果セット内のロー数を返します。

構文

UInt32 **getRowCount( )**

getRowCountWithThreshold メソッド

指定されたスレッショルドのロー数以内で、結果セット内のロー数を返します。

構文

UInt32 **getRowCount(** UInt32 *threshold***)**

パラメータ

threshold　この値は、ロー・カウント操作に上限値を指定します。スレッショルド値を超える個数のローがある場合、結果はスレッショルド値になります。ローが多い場合、ローのカウントは負荷の高い操作になることがあります。一部のアプリケーションでは、特定のロー数を超えているかどうかを確認するだけで済み (たとえば、ユーザがより多くのローを要求するためのオプションを提供するかどうかを決定する場合など)、正確なロー・カウントは必要としません。この値が 0 の場合、すべてのローがカウントされます。

getShort メソッド

Int16 として値を返します。

構文

Int16 **getShort(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getString メソッド

String として値を返します。

構文

String **getString(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getStringChunk メソッド

指定されたオフセットで始まる、指定された SQLType.LONGVARCHAR カラムの値のサブセットを String オブジェクトにコピーします。

構文

String **getStringChunk(**
  UInt16 *index*,
  UInt32 *srcOffset*,
  UInt32 *count*
)

パラメータ

- **index**　結果セットで取得する 1 から始まる順序。
- **srcOffset**　文字列値で 0 から始まる開始位置。
- **count**　コピーされる文字数。

戻り値

指定された文字数がコピーされた文字列。

getTime メソッド

Date として値を返します。

構文

Date **getTime(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getTimestamp メソッド

Date として値を返します。

構文

Date **getTimestamp(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getULong メソッド

64 ビット符号なし整数として値を返します。

構文

UInt64 **getULong(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getUUID メソッド

UUID としてカラムの値を返します。

構文

UUID **getUUID(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

備考

カラムは長さが 16 の SQLType.BINARY 型である必要があります。

isBOF メソッド

現在のローの位置が最初のローよりも前である場合は **true**、それ以外の場合は **false** を返します。

構文

Boolean **isBOF( )**

isEOF メソッド

現在のローの位置が最後のローよりも後である場合は **true**、それ以外の場合は **false** を返します。

構文

Boolean **isEOF( )**

isNull メソッド

値が NULL の場合は **true**、NULL 以外の場合は **false** を返します。

構文

Boolean **isNull(** UInt16 *index* **)**

パラメータ

index　カラムのインデックス値。

isOpen メソッド

ResultSet が開いている場合は **true**、それ以外の場合は **false** を返します。

構文

Boolean **isOpen( )**

moveAfterLast メソッド

ULResultSet の最後のローの後に移動します。

構文

moveAfterLast( )

moveBeforeFirst メソッド

最初のローの前に移動します。

構文

moveBeforeFirst( )

moveFirst メソッド

最初のローに移動します。

構文

Boolean **moveFirst( )**

戻り値

成功の場合は **True**。

失敗の場合は **False**。たとえば、ローがない場合、メソッドは失敗します。

moveLast メソッド

最後のローに移動します。

構文

Boolean **moveLast( )**

戻り値

成功の場合は **True**。

失敗の場合は **False**。たとえば、ローがない場合、メソッドは失敗します。

moveNext メソッド

次のローに移動します。

構文

Boolean **moveNext( )**

戻り値

成功の場合は **True**。

失敗の場合は **False**。たとえば、ローがない場合、メソッドは失敗します。

movePrevious メソッド

前のローに移動します。

構文

Boolean **movePrevious( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveRelative メソッド

いくつかのローを、現在のローを基準にして相対的に移動します。

構文

Boolean **moveRelative(** Int32 *index* **)**

パラメータ

index　移動するローの数。値は、正、負、または 0 です。

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

備考

結果セットでのカーソルの現在位置を基準にして、正のインデックス値は結果セット内を前に移動し、負のインデックス値は結果セット内を後ろに移動し、0 はカーソルを移動しません。

setBoolean メソッド

指定されたカラムの値を、**boolean** を使用して設定します。

構文

setBoolean(short *columnID*, boolean *value*)

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、**update** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setBytes メソッド

指定されたカラムの値を、**byte** 配列を使用して設定します。

構文

setBytes(UInt16 *columnID*, Array *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

SQLType.BINARY 型、**SQLType.LONGBINARY** 型のカラムにのみ適しています。ローのデータは、**update** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setDate メソッド

指定されたカラムの値を、**Date** を使用して設定します。

構文

setDate(UInt16 *columnID*, Date *value***)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setDateTime メソッド

指定されたカラムの値を、**Date** を使用して設定します。

構文

setDateTime(UInt16 *columnID*, Date *value***)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setDouble メソッド

指定されたカラムの値を、**double** を使用して設定します。

構文

setDouble(UInt16 *columnID*, Double *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setFloat メソッド

指定されたカラムの値を、**float** を使用して設定します。

構文

setFloat(UInt16 *columnID*, Float *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setInt メソッド

指定されたカラムの値を、Integer を使用して設定します。

構文

setInt(UInt16 *columnID*, Int32 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setLong メソッド

指定されたカラムの値を、Int64 を使用して設定します。

構文

setLong(UInt16 *columnID*, Int64 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setNull メソッド

カラムに SQL NULL を設定します。

構文

setNull(UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

データは、update を実行するまでは、実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setShort メソッド

指定されたカラムの値を、UInt16 を使用して設定します。

構文

setShort(UInt16 *columnID*, Int16 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setString メソッド

指定されたカラムの値を、String を使用して設定します。

構文

setString(UInt16 *columnID*, String *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setTime メソッド

指定されたカラムの値を、Date を使用して設定します。

構文

setTime(UInt16 *columnID*, Date *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setTimestamp メソッド

指定されたカラムの値を、Date を使用して設定します。

構文

setTimestamp(UInt16 *columnID*, Date *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setULong メソッド

指定されたカラムの値を、符号なし値として扱われる 64 ビット整数を使用して設定します。

構文

setULong(UInt16 *columnID*, UInt64 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setUUID メソッド

指定されたカラムの値を、UUID を使用して設定します。

構文

setUUID(UInt16 *columnID*, UUID *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。長さが 16 の SQLType.BINARY 型のカラムの場合にのみ有効です。

参照

- 「UUID の使用」 『Mobile Link - サーバ管理』

update メソッド

現在のカラム値 (set メソッドを使用して指定されます) で新しいローを更新します。

構文

update()

備考

update を行う前に updateBegin を呼び出してください。

updateBegin メソッド

この結果セットの現在のローを更新する準備を行います。

構文

updateBegin()

備考

カラム値は、適切な set*Type* メソッドを呼び出すことによって修正します。

データは、update を実行するまでは、実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

ResultSetSchema クラス

Ultra Light の結果セットのスキーマを表します。

getColumnCount メソッド

このカーソル内のカラム数を返します。

構文

UInt16 **getColumnCount()**;

備考

カラム ID の範囲は、1 ～ getColumnCount です。

カラムの ID とカウントは、スキーマのアップグレード中に変更されることがあります。カラムを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getColumnID メソッド

指定されたカラムのカラム ID を返します。

構文

UInt16 **getColumnID(**String *name***)**

パラメータ

- **name**　カラムの名前。

備考

カラム ID の範囲は、1 ～ getColumnCount() です。

カラムの ID とカウントは、スキーマのアップグレード中に変更されることがあります。カラムを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getColumnName メソッド

指定されたカラム ID で識別されたカラムの名前を返します。

構文

String **getColumnName(**UInt16 *columnID***)**

パラメータ

- **columnID**　カラムの ID。**columnID** は、**[1,getColumnCount()]** の範囲内であることが必要です。

備考

カラムの ID とカウントは、スキーマのアップグレード中に変更されることがあります。カラムを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getColumnPrecision メソッド

指定されたカラムの精度を返します。

構文

Int32 **getColumnPrecision(**String *name***)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnPrecisionByColID メソッド

カラムの精度を返します。

構文

Int32 **getColumnPrecisionByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。結果セットの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnScale メソッド

カラムの位取りを返します。

構文

Int32 **getColumnScale(**String *name***)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnScaleByColID メソッド

カラムの位取りを返します。

構文

Int32 **getColumnScaleByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。結果セットの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnSize メソッド

指定されたカラムのサイズを返します。

構文

UInt32 **getColumnSize(**String *name***)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnSizeByColID メソッド

カラムのサイズを返します。

構文

UInt32 **getColumnSizeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。結果セットの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnSQLType メソッド

指定されたカラムの SQL データ型を返します。

構文

UInt16 **getColumnSQLType(**String *name***)**

パラメータ

- **name**　カラムの名前。

getColumnSQLTypeByColID メソッド

カラムの SQLType を示す SQLType 列挙整数を返します。

構文

UInt16 **getColumnSQLTypeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。結果セットの最初のカラムの ID 値は 1 です。

isOpen メソッド

結果セットが開いている場合は **true**、それ以外の場合は **false** を返します。

構文

Boolean **isOpen()**;

SQLError クラス

Ultra Light for M-Business Anywhere でレポートされる可能性のある SQL コードを列挙します。このクラスは静的定数を提供し、直接インスタンス化することはできません。

メンバ	説明
SQLE_AGGREGATES_NOT_ALLOWED	「集合関数の使用が無効です。」『エラー・メッセージ』を参照。
SQLE_ALIAS_NOT_UNIQUE	「エイリアス '%1' がユニークではありません。」『エラー・メッセージ』を参照。
SQLE_ALIAS_NOT_YET_DEFINED	「エイリアス '%1' の定義は、最初の参照前に記述する必要があります。」『エラー・メッセージ』を参照。
SQLE_AMBIGUOUS_INDEX_NAME	「インデックス名 '%1' があいまいです。」『エラー・メッセージ』を参照。
SQLE_BAD_ENCRYPTION_KEY	「暗号化キーが不正であるか、見つかりません。」『エラー・メッセージ』を参照。
SQLE_BAD_PARAM_INDEX	「入力パラメータ・インデックスが範囲外です。」『エラー・メッセージ』を参照。
SQLE_CANNOT_ACCESS_FILESYSTEM	「デバイス上のファイルシステムにアクセスできません。」『エラー・メッセージ』を参照。
SQLE_CANNOT_CHANGE_USER_NAME	「前回のアップロードのステータスが不明の場合、同期の user_name は変更できません。」『エラー・メッセージ』を参照。
SQLE_CANNOT_CONVERT	「不正なデータ変換」『エラー・メッセージ』を参照。
SQLE_CANNOT_EXECUTE_STMT	「文を実行することができませんでした。」『エラー・メッセージ』を参照。
SQLE_CANNOT_MODIFY	「テーブル '%2' のカラム '%1' を変更できません。」『エラー・メッセージ』を参照。
SQLE_CLIENT_OUT_OF_MEMORY	「クライアントでメモリが不足しています。」『エラー・メッセージ』を参照。
SQLE_COLUMN_AMBIGUOUS	「カラム '%1' が複数のテーブルで見つかりました。相関名が必要です。」『エラー・メッセージ』を参照。
SQLE_COLUMN_CANNOT_BE_NULL	「テーブル '%2' のカラム '%1' を NULL にすることはできません。」『エラー・メッセージ』を参照。

メンバ	説明
SQLE_COLUMN_IN_INDEX	「インデックスのカラムを変更することはできません。」 『エラー・メッセージ』を参照。
SQLE_COLUMN_NOT_FOUND	「カラム '%1' が見つかりません。」 『エラー・メッセージ』を参照。
SQLE_COLUMN_NOT_INDEXED	「カラム '%1' は、それを含んでいるテーブルのどのインデックスにも属していません。」 『エラー・メッセージ』を参照。
SQLE_COLUMN_NOT_STREAMABLE	「操作が失敗しました。カラム '%1' のタイプがストリーミングをサポートしていません。」 『エラー・メッセージ』を参照。
SQLE_COMMUNICATIONS_ERROR	「通信エラーが発生しました。」 『エラー・メッセージ』を参照。
SQLE_CONNECTION_ALREADY_EXISTS	「この接続はすでに存在します。」 『エラー・メッセージ』を参照。
SQLE_CONNECTION_NOT_FOUND	「接続が見つかりません。」 『エラー・メッセージ』を参照。
SQLE_CONNECTION_RESTORED	「Ultra Light の接続がリストアされました。」 『エラー・メッセージ』を参照。
SQLE_CONSTRAINT_NOT_FOUND	「制約 '%1' が見つかりません。」 『エラー・メッセージ』を参照。
SQLE_CONVERSION_ERROR	「値 %1 をデータ型 %2 に変換できません。」 『エラー・メッセージ』を参照。
SQLE_COULD_NOT_FIND_FUNCTION	「ダイナミック・ライブラリ '%2' に '%1' が見つかりませんでした。」 『エラー・メッセージ』を参照。
SQLE_COULD_NOT_LOAD_LIBRARY	「ダイナミック・ライブラリ '%1' をロードできませんでした。」 『エラー・メッセージ』を参照。
SQLE_CURSOR_ALREADY_OPEN	「カーソルはすでに開いています。」 『エラー・メッセージ』を参照。
SQLE_CURSOR_NOT_OPEN	「カーソルが開きません。」 『エラー・メッセージ』を参照。
SQLE_CURSOR_RESTORED	「Ultra Light のカーソル (あるいは結果セットまたはテーブル) がリストアされました。」 『エラー・メッセージ』を参照。
SQLE_CURSOROP_NOT_ALLOWED	「不正なカーソル処理をしようとしました。」 『エラー・メッセージ』を参照。

メンバ	説明
SQLE_DATABASE_ERROR	「データベースの内部エラー %1 -- トランザクションはロールバックされました。」『エラー・メッセージ』を参照。
SQLE_DATABASE_NAME_REQUIRED	「サーバを起動するには、データベース名が必要です。」『エラー・メッセージ』を参照。
SQLE_DATABASE_NOT_CREATED	「データベースの作成に失敗しました : %1」『エラー・メッセージ』を参照。
SQLE_DATATYPE_NOT_ALLOWED	「式にサポートされていないデータ型があります。」『エラー・メッセージ』を参照。
SQLE_DBSPACE_FULL	「DB 領域が最大ファイル・サイズに達しています。」『エラー・メッセージ』を参照。
SQLE_DESCRIBE_NONSELECT	「SELECT 文以外は記述できません。」『エラー・メッセージ』を参照。
SQLE_DIV_ZERO_ERROR	「ゼロで除算しようとしました。」『エラー・メッセージ』を参照。
SQLE_DOWNLOAD_CONFLICT	「既存のローと競合しているため、ダウンロードに失敗しました。」『エラー・メッセージ』を参照。
SQLE_DOWNLOAD_RESTART_FAILED	「ダウンロードをリトライできません。アップロードが完了していません。」『エラー・メッセージ』を参照。
SQLE_DROP_DATABASE_FAILED	「データベース '%1' の削除に失敗しました。」『エラー・メッセージ』を参照。
SQLE_DUPLICATE_CURSOR_NAME	「カーソル名 '%1' はすでに存在します。」『エラー・メッセージ』を参照。
SQLE_DUPLICATE_FOREIGN_KEY	「テーブル '%2' の外部キー '%1' は、既存の外部キーと重複しています。」『エラー・メッセージ』を参照。
SQLE_DUPLICATE_OPTION	「オプション '%1' が複数回指定されています。」『エラー・メッセージ』を参照。
SQLE_DYNAMIC_MEMORY_EXHAUSTED	「動的メモリが足りません。」『エラー・メッセージ』を参照。
SQLE_ENCRYPTION_INITIALIZATION_FAILED	「暗号化 DLL を初期化できませんでした : '%1'」『エラー・メッセージ』を参照。

メンバ	説明
SQLE_ENGINE_ALREADY_RUNNING	「データベース・サーバはすでに起動しています。」『エラー・メッセージ』を参照。
SQLE_ENGINE_NOT_MULTI USER	「データベース・サーバはマルチユーザ・モードで実行していません。」『エラー・メッセージ』を参照。
SQLE_ERROR	「実行時 SQL エラーです -- %1」『エラー・メッセージ』を参照。
SQLE_ERROR_CALLING_FUNCTION	「外部関数の呼び出しのためのリソースを割り付けられませんでした。」『エラー・メッセージ』を参照。
SQLE_ERROR_IN_ASSIGNMENT	「割り当てのエラー」『エラー・メッセージ』を参照。
SQLE_EXPRESSION_ERROR	「'%1' 付近に無効な式があります。」『エラー・メッセージ』を参照。
SQLE_FEATURE_NOT_ENABLED	「呼び出そうとしたメソッドは、お使いのアプリケーションでは使用できません。」『エラー・メッセージ』を参照。
SQLE_FILE_BAD_DB	「指定されたデータベースを開始できません。'%1' は有効なデータベース・ファイルではありません。」『エラー・メッセージ』を参照。
SQLE_FILE_IN_USE	「指定されたデータベース・ファイルはすでに使用されています。」『エラー・メッセージ』を参照。
SQLE_FILE_NOT_DB	「指定されたデータベースを開始できません。'%1' はデータベースではありません。」『エラー・メッセージ』を参照。
SQLE_FILE_VOLUME_NOT_FOUND	「データベース '%1' に対して指定したファイルシステム・ボリュームが見つかりません。」『エラー・メッセージ』を参照。
SQLE_FILE_WRONG_VERSION	「指定されたデータベースを開始できません。'%1' は異なるバージョンのソフトウェアで作成されています。」『エラー・メッセージ』を参照。
SQLE_FOREIGN_KEY_NAME_NOT_FOUND	「外部キー '%1' は見つかりません。」『エラー・メッセージ』を参照。
SQLE_IDENTIFIER_TOO_LONG	「識別子 '%1' が長すぎます。」『エラー・メッセージ』を参照。

メンバ	説明
SQLE_INCORRECT_VOLUME_ID	「'%1' のボリューム ID が不正です。」 『エラー・メッセージ』を参照。
SQLE_INDEX_NAME_NOT_UNIQUE	「インデックス名 '%1' はユニークではありません。」 『エラー・メッセージ』を参照。
SQLE_INDEX_NOT_FOUND	「インデックス '%1' が見つかりません。」 『エラー・メッセージ』を参照。
SQLE_INDEX_NOT_UNIQUE	「テーブル '%2' のインデックス '%1' はユニークでなければなりません。」 『エラー・メッセージ』を参照。
SQLE_INTERRUPTED	「文の実行がユーザによって中断させられました。」 『エラー・メッセージ』を参照。
SQLE_INVALID_CONSTRAINT_REF	「制約 '%1' への参照または操作が無効です。」 『エラー・メッセージ』を参照。
SQLE_INVALID_DESCRIPTOR_INDEX	「記述子のインデックスが正しくありません。」 『エラー・メッセージ』を参照。
SQLE_INVALID_DESCRIPTOR_NAME	「SQL 記述子名が正しくありません。」 『エラー・メッセージ』を参照。
SQLE_INVALID_DISTINCT_AGGREGATE	「グループ化されたクエリに、複数の異なる集合関数が含まれています。」 『エラー・メッセージ』を参照。
SQLE_INVALID_FOREIGN_KEY	「テーブル '%2' の外部キー '%1' に対応するプライマリ・キーの値がありません。」 『エラー・メッセージ』を参照。
SQLE_INVALID_FOREIGN_KEY_DEF	「外部キーのカラム '%1' にプライマリ・キーと異なる定義があります。」 『エラー・メッセージ』を参照。
SQLE_INVALID_GROUP_SELECT	「'%1' に対する関数またはカラムの参照も GROUP BY 句に記述する必要があります。」 『エラー・メッセージ』を参照。
SQLE_INVALID_LOGON	「ユーザ ID またはパスワードが無効です。」 『エラー・メッセージ』を参照。
SQLE_INVALID_OPTION_SETTING	「オプション '%1' の設定が無効です。」 『エラー・メッセージ』を参照。
SQLE_INVALID_OPTION_VALUE	「'%1' は '%2' に対して無効な値です。」 『エラー・メッセージ』を参照。

メンバ	説明
SQLE_INVALID_ORDER	「ORDER BY 句の指定が不正です。」 『エラー・メッセージ』を参照。
SQLE_INVALID_PARAMETER	「不正なパラメータです。」 『エラー・メッセージ』を参照。
SQLE_INVALID_PARSE_PARAMETER	「解析エラー : %1」 『エラー・メッセージ』を参照。
SQLE_INVALID_SQL_IDENTIFIER	「SQL の識別子が無効です。」 『エラー・メッセージ』を参照。
SQLE_INVALID_UNION	「UNION、INTERSECT、または EXCEPT の select リストの長さが一致していません。」 『エラー・メッセージ』を参照。
SQLE_KEYLESS_ENCRYPTION	「このデータベースはキー不使用の暗号化を使用するため、要求された操作を実行できません。」 『エラー・メッセージ』を参照。
SQLE_LOCKED	「'%2' のローは、ユーザ '%1' によってロックされています。」 『エラー・メッセージ』を参照。
SQLE_MEMORY_ERROR	「メモリ・エラー -- トランザクションはロールバックされました。」 『エラー・メッセージ』を参照。
SQLE_NAME_NOT_UNIQUE	「アイテム '%1' はすでに存在しています。」 『エラー・メッセージ』を参照。
SQLE_NO_COLUMN_NAME	「派生テーブル '%1' にはカラム %2 に対する名前がありません。」 『エラー・メッセージ』を参照。
SQLE_NO_CURRENT_ROW	「カーソルの現在のローがありません。」 『エラー・メッセージ』を参照。
SQLE_NO_INDICATOR	「NULL に対して、インジケータ変数が用意されていません。」 『エラー・メッセージ』を参照。
SQLE_NO_MATCHING_SELECT_ITEM	「派生テーブル '%1' の select リストに '%2' と一致する式がありません。」 『エラー・メッセージ』を参照。
SQLE_NO_PRIMARY_KEY	「テーブル '%1' にはプライマリ・キーが定義されていません。」 『エラー・メッセージ』を参照。
SQLE_NOERROR	SQLE_NOERROR(0) - このコードは、エラーまたは警告がなかったことを示します。

メンバ	説明
SQLE_NON_UPDATEABLE_COLUMN	「式を更新できません。」『エラー・メッセージ』を参照。
SQLE_NON_UPDATEABLE_CURSOR	「FOR UPDATE が READ ONLY カーソルに誤って指定されました。」『エラー・メッセージ』を参照。
SQLE_NOT_IMPLEMENTED	「'%1' の機能は実装されていません。」『エラー・メッセージ』を参照。
SQLE_NOT_SUPPORTED_IN_ULTRALITE	「Ultra Light では使用できない機能です。」『エラー・メッセージ』を参照。
SQLE_NOTFOUND	「ローが見つかりません。」『エラー・メッセージ』を参照。
SQLE_ONLY_ONE_TABLE	「カーソルの INSERT/DELETE は、1 つのテーブルしか変更できません。」『エラー・メッセージ』を参照。
SQLE_OVERFLOW_ERROR	「値 %1 は、対象先にとって大きすぎます。」『エラー・メッセージ』を参照。
SQLE_PAGE_SIZE_INVALID	「無効なデータベース・ページ・サイズです。」『エラー・メッセージ』を参照。
SQLE_PARTIAL_DOWNLOAD_NOT_FOUND	「部分ダウンロードが見つかりませんでした。」『エラー・メッセージ』を参照。
SQLE_PERMISSION_DENIED	「パーミッションがありません : %1」『エラー・メッセージ』を参照。
SQLE_PRIMARY_KEY_NOT_UNIQUE	「テーブル '%1' のプライマリ・キーがユニークではありません : プライマリ・キー値 ('%2')」『エラー・メッセージ』を参照。
SQLE_PRIMARY_KEY_TWICE	「テーブルに 2 つのプライマリ・キーを定義することはできません。」『エラー・メッセージ』を参照。
SQLE_PRIMARY_KEY_VALUE_REF	「テーブル '%1' 内のローのプライマリ・キーがテーブル '%3' 内の外部キー '%2' によって参照されています。」『エラー・メッセージ』を参照。
SQLE_PUBLICATION_NOT_FOUND	「パブリケーション '%1' が見つかりません。」『エラー・メッセージ』を参照。
SQLE_PUBLICATION_PREDICATE_IGNORED	「パブリケーションの述部は評価されませんでした。」『エラー・メッセージ』を参照。

メンバ	説明
SQLE_RESOURCE_GOVERNOR_EXCEEDED	「%1 のリソース・ガバナーが制限を超えています。」 『エラー・メッセージ』を参照。
SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY	「参照整合性を保つためにテーブル %1 からローが削除されました。」 『エラー・メッセージ』を参照。
SQLE_SCHEMA_UPGRADE_NOT_ALLOWED	「スキーマのアップグレードは現在有効ではありません。」 『エラー・メッセージ』を参照。
SQLE_SERVER_SYNCHRONIZATION_ERROR	「サーバ %1 でエラーが発生したため、同期に失敗しました。」 『エラー・メッセージ』を参照。
SQLE_START_STOP_DATABASE_DENIED	「データベースの起動／停止の要求は拒否されました。」 『エラー・メッセージ』を参照。
SQLE_STATEMENT_ERROR	「SQL 文にエラーがあります。」 『エラー・メッセージ』を参照。
SQLE_STRING_RIGHT_TRUNCATION	「文字列データの右側がトランケートされます。」 『エラー・メッセージ』を参照。
SQLE_SUBQUERY_SELECT_LIST	「select リストの中にカラムが 2 つ以上指定されています。」 『エラー・メッセージ』を参照。
SQLE_SYNC_INFO_INVALID	「同期の情報が不完全か無効です。'%1' を確認してください。」 『エラー・メッセージ』を参照。
SQLE_SYNC_INFO_REQUIRED	「同期の情報が指定されていません。」 『エラー・メッセージ』を参照。
SQLE_SYNC_NOT_REENTRANT	「同期処理に戻ることができませんでした。」 『エラー・メッセージ』を参照。
SQLE_SYNC_STATUS_UNKNOWN	「最後の同期アップロードのステータスは不明です。」 『エラー・メッセージ』を参照。
SQLE_SYNTAX_ERROR	「'%1' %2 の近くに構文エラーがあります。」 『エラー・メッセージ』を参照。
SQLE_TABLE_ALREADY_INCLUDED	「テーブル '%1' はすでにインクルードされています。」 『エラー・メッセージ』を参照。
SQLE_TABLE_IN_USE	「テーブルは使用されています。」 『エラー・メッセージ』を参照。

メンバ	説明
SQLE_TABLE_NOT_FOUND	「テーブル '%1' が見つかりません。」 『エラー・メッセージ』を参照。
SQLE_TOO_MANY_BLOB_REFS	「BLOB への参照が多すぎます。」 『エラー・メッセージ』を参照。
SQLE_TOO_MANY_CONNECTIONS	「データベース・サーバに接続できる限界数を超えています。」 『エラー・メッセージ』を参照。
SQLE_TOO_MANY_PUBLICATIONS	「オペレーションに指定されているパブリケーションが多すぎます」 『エラー・メッセージ』を参照。
SQLE_TOO_MANY_TEMP_TABLES	「接続しているテンポラリ・テーブルが多すぎます。」 『エラー・メッセージ』を参照。
SQLE_TOO_MANY_USERS	「データベースのユーザが多すぎます。」 『エラー・メッセージ』を参照。
SQLE_ULTRALITE_DATABASE_NOT_FOUND	「データベース '%1' が見つかりませんでした。」 『エラー・メッセージ』を参照。
SQLE_ULTRALITE_OBJ_CLOSED	「閉じられたオブジェクトに対する操作は無効です。」 『エラー・メッセージ』を参照。
SQLE_ULTRALITE_WRITE_ACCESS_DENIED	「書き込みアクセスが拒否されました。」 『エラー・メッセージ』を参照。
SQLE_UNABLE_TO_CONNECT	「データベースが起動できません -- %1」 『エラー・メッセージ』を参照。
SQLE_UNABLE_TO_START_DATABASE	「指定されたデータベースを起動できません : %1」 『エラー・メッセージ』を参照。
SQLE_UNABLE_TO_START_DATABASE_VER_NEWER	「指定されたデータベースを起動できません : データベース %1 を起動するにはサーバをアップグレードする必要があります。」 『エラー・メッセージ』を参照。
SQLE_UNCOMMITTED_TRANSACTIONS	「コミットされていないトランザクションとの同期またはアップグレードはできません。」 『エラー・メッセージ』を参照。
SQLE_UNKNOWN_FUNC	「関数 '%1' はありません。」 『エラー・メッセージ』を参照。
SQLE_UNKNOWN_OPTION	「'%1' は認識できないオプションです。」 『エラー・メッセージ』を参照。

メンバ	説明
SQLE_UNKNOWN_USERID	「'%1' というユーザ ID はありません。」 『エラー・メッセージ』を参照。
SQLE_UNRECOGNIZED_OPTION	「オプション '%1' が認識されません。」 『エラー・メッセージ』を参照。
SQLE_UPLOAD_FAILED_AT_SERVER	「同期サーバがアップロードのコミットに失敗しました。」 『エラー・メッセージ』を参照。
SQLE_VALUE_IS_NULL	「要求されたデータ型として NULL の結果を返すことができません。」 『エラー・メッセージ』を参照。
SQLE_VARIABLE_INVALID	「無効なホスト変数です。」 『エラー・メッセージ』を参照。
SQLE_WRONG_NUM_OF_INSERT_COLS	「INSERT 文に指定した値の数が正しくありません。」 『エラー・メッセージ』を参照。
SQLE_WRONG_PARAMETER_COUNT	「関数 '%1' のパラメータ数が誤りです。」 『エラー・メッセージ』を参照。

SQLType クラス

この列挙体は、テーブル・カラムの型として使用されている、Ultra Light SQL データベースの型を定数としてリストします。

定数	Ultra Light データベースの型
BAD_INDEX	
S_LONG	INT
U_LONG	UNSIGNED INT
S_SHORT	SMALLINT
U_SHORT	UNSIGNED SMALLINT
S_BIG	BIGINT
U_BIG	UNSIGNED BIGINT
TINY	TINYINT
BIT	BIT
TIMESTAMP	TIMESTAMP
DATE	DATE
TIME	TIMESTAMP
DOUBLE	DOUBLE
REAL	REAL
NUMERIC	NUMERIC
BINARY	BINARY
CHAR	CHAR または VARCHAR
LONGVARCHAR	LONG VARCHAR
LONGBINARY	LONG BINARY
MAX_INDEX	

toString メソッド

指定された SQL カラム型定数の文字列名、または認識されない型の場合は BAD_SQL_TYPE を返します。

構文

String **toString(**UInt16 *code***)**

パラメータ

- **code**　SQL カラム型定数。

SyncParms クラス

Ultra Light データベースの同期方法を定義する同期パラメータを表します。それぞれの接続には、固有の SyncParms インスタンスがあります。

定数

定数	値	説明
STREAM_TYPE_TCPIP	0	TCP/IP ストリーム
STREAM_TYPE_HTTP	1	HTTP ストリーム
STREAM_TYPE_HTTPS	2	HTTPS 同期
STREAM_TYPE_TLS	3	TLS 同期
STREAM_TYPE_HOTSYNC	4	HotSync 同期用

getAdditionalParms メソッド

名前と値のペアの文字列を返します。

構文

String **getAdditionalParms()**

参照

- 「Additional 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』

getAuthenticationParms メソッド

カスタム・ユーザ認証スクリプトに渡されたパラメータ、またはパラメータが指定されなかった場合は NULL を返します。

構文

Array **getAuthenticationParms()**

getDownloadOnly メソッド

アップロードが無効な場合は true、有効な場合は false を返します。

構文

Boolean **getDownloadOnly()**

getKeepPartialDownload メソッド

部分的なダウンロードが保持される場合は true、ロールバックされる場合は false を返します。

構文

Boolean getKeepPartialDownload()

getNewPassword メソッド

次回の同期以降、Mobile Link ユーザに関連付けられる新しいパスワードを返します。

構文

String **getNewPassword()**

getPartialDownloadRetained メソッド

通信エラーが原因でダウンロードが失敗し、部分的なダウンロードが保持された場合は true、ダウンロードが中断されなかった場合または部分的なダウンロードがロールバックされた場合は false を返します。

構文

Boolean **getPartialDownloadRetained( )**

getPassword メソッド

setUserName で指定されたユーザの Mobile Link パスワードを返します。

構文

String **getPassword()**;

getPingOnly メソッド

クライアントがサーバに ping のみを実行する場合は true、クライアントが同期を実行する場合は false を返します。

構文

Boolean **getPingOnly()**

getResumePartialDownload メソッド

前回の部分的なダウンロードが再開される場合は true、ロールバックされる場合は false を返します。

構文

Boolean **getResumePartialDownload( )**

getSendColumnNames メソッド

クライアントが同期中にカラム名を Mobile Link サーバに送信する場合は true、送信しない場合は false を返します。

構文

Boolean **getSendColumnNames()**

getSendDownloadAck メソッド

クライアントが Mobile Link サーバにダウンロード確認を送信する場合は true、送信しない場合は false を返します。

構文

Boolean **getSendDownloadAck()**

getStream メソッド

同期に使用する Mobile Link 同期ストリームのタイプを返します。

構文

UInt16 **getStream()**;

getStreamParms メソッド

同期ストリームに使用されるすべてのネットワーク・プロトコル・オプションを含む文字列を返します。

構文

String **getStreamParms()**;

getUploadOnly メソッド

ダウンロードが無効な場合は true、有効な場合は false を返します。

構文

Boolean **getUploadOnly()**

getUserName メソッド

Mobile Link ユーザ名を返します。

構文

String **getUserName()**

getVersion メソッド

使用される同期スクリプトを示すバージョン文字列を返します。

構文

String **getVersion()**

setAdditionalParms メソッド

「キーワード=値」のペアの文字列を指定します。通常、このペアのリストには、使用頻度の低い同期パラメータが入ります。

構文

getAdditionalParms(String *additional_params***)**

パラメータ

- **additional_params**　「キーワード=値」のペアの文字列。

参照

- 「Additional 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』

setAuthenticationParms メソッド

カスタム・ユーザ認証スクリプト (Mobile Link authenticate_parameters 接続イベント) のパラメータを指定します。

構文

setAuthenticationParms(Array *value* **)**

パラメータ

- **value** それぞれに認証パラメータが格納された、文字列の配列 (配列のエントリが NULL であると、同期エラーになります)。

備考

最初の 255 文字のみが使用されます。また、各文字列は 128 文字以下である必要があります (長すぎる文字列は、Mobile Link に送信されるときにトランケートされます)。

setDownloadOnly メソッド

同期時のアップロードを無効にするか、有効にするかを指定します。

構文

setDownloadOnly(Boolean *value* **)**

パラメータ

- **value** アップロードを無効にする場合は true、有効にする場合は false に設定します。

setKeepPartialDownload メソッド

同期時の部分的なダウンロードを無効にするか、有効にするかを指定します。

構文

setKeepPartialDownload(Boolean *value* **)**

パラメータ

- **value** 同期中に部分的なダウンロードを保持する場合は true、破棄する場合は false に設定します。

備考

Ultra Light では、通信エラーが原因で失敗したダウンロードを再開することができます。Ultra Light は、ダウンロードを受信しながら処理します。ダウンロードが中断した場合は、部分的なダウンロード・トランザクションがデータベース内に残るため、次の同期中に再開できます。

Ultra Light で部分的なダウンロードを保存する必要があることを示すには、Connection.syncParms.setKeepPartialDownload(true); と指定します。指定しないと、エラーが発生した場合にダウンロードがロールバックされます。

部分的なダウンロードが保持された場合、connection.synchronize の終了時に、出力フィールド connection.SyncResult.getPartialDownloadRetained が true に設定されます。getPartialDownloadRetained が設定されている場合は、ダウンロードを再開できます。再開するに

は、connection.syncParms.setResumePartialDownload(true) を指定して connection.synchronize を呼び出します。多くの場合、別の通信エラーの発生に備えて、KeepPartialDownload も true に設定する必要があります。ダウンロードが省略された場合は、アップロードは行われません。

再開したダウンロードで受信するダウンロードは、最初にダウンロードを開始したときと同じものです。最新のデータが必要な場合は、再開された特別なダウンロードが完了した直後に、もう一度ダウンロードを行うことができます。

ダウンロードを再開する場合、SyncParms フィールドの多くは関係ありません。受信するパブリケーションは、最初のダウンロード時に要求したものです。設定する必要があるフィールドは、setResumePartialDownload(boolean) と setUserName(String) だけです。setKeepPartialDownload(boolean) フィールド、setDownloadOnly(boolean) フィールド、setDisableConcurrency(boolean) フィールドを必要に応じて設定すると、正常に機能します。

部分的なダウンロードが存在するが、このダウンロードが必要ではなくなった場合は、Connection.rollbackPartialDownload を呼び出して、失敗したダウンロード・トランザクションをロールバックできます。また、同期をもう一度実行したときに ResumePartialDownload を指定しなかった場合は、部分的なダウンロードがロールバックされてから、次の同期が開始されます。

「同期の障害処理の方法」『Mobile Link - クイック・スタート』を参照してください。

setMBAServer メソッド

Mobile Link ホストとポートの同期パラメータを、M-Business Client によって使用される M-Business Anywhere Server の同期パラメータにすばやく設定できます。

構文

setMBAServer(String *host*, String *port*, String *url_suffix* **)**

パラメータ

- **host**　M-Business Anywhere Server のホストまたは IP 値。host が NULL の場合、現在の M-Business Anywhere ホストに設定されます。
- **port**　M-Business Anywhere Server が受信しているポート。port が NULL の場合、現在の M-Business Anywhere ポート値に設定されます。
- **url_suffix**　これは M-Business Anywhere の *sync.conf* ファイルで設定された url_suffix パラメータに対応します。

備考

データを Mobile Link サーバとの間でルート指定するには、M-Business Anywhere 用の Mobile Link リダイレクタを使用します。

ワンタッチ同期を使用している場合は、Connection.saveSyncParms を使用して同期パラメータを保存してください。

M-Business Anywhere リダイレクタを使用して HTTP データベース・トラフィックをルート指定するように M-Business Server を設定する方法については、「M-Business Anywhere リダイレクタ (旧式)」『Mobile Link - サーバ管理』を参照してください。

setMBAServerWithMoreParms メソッド

Mobile Link ホストとポートの同期パラメータを、M-Business Client によって使用される M-Business Anywhere Server の同期パラメータにすばやく設定できます。

構文

setMBAServerWithMoreParms(String *host*, String *port*, String *url_suffix*, String *additional***)**

パラメータ

- **host**　M-Business Anywhere Server のホストまたは IP 値。host が NULL の場合、現在の M-Business Anywhere ホストに設定されます。
- **port**　M-Business Anywhere Server が受信しているポート。port が NULL の場合、現在の M-Business Anywhere ポート値に設定されます。
- **url_suffix**　これは M-Business Anywhere の *sync.conf* ファイルで設定された url_suffix パラメータに対応します。
- **additional**　このパラメータには、先行するパラメータでは扱われない追加のストリーム・パラメータが含まれる可能性があります (プロキシ・ホスト、プロキシ・ポート、セキュリティ関連パラメータなど)。host、port、url_suffix の情報を指定する必要がある場合は、前述した setMBAServer メソッドを使用することもできます。

備考

データを Mobile Link サーバとの間でルート指定するには、M-Business Anywhere 用の Mobile Link リダイレクタを使用します。

このメソッドは setMBAServer で提供される機能を拡張しており、ユーザは additional パラメータ内で別のパラメータを指定できます。

ワンタッチ同期を使用している場合は、Connection.saveSyncParms を使用して同期パラメータを保存してください。

M-Business Anywhere リダイレクタを使用して HTTP データベース・トラフィックをルート指定するように M-Business Server を設定する方法については、「M-Business Anywhere リダイレクタ (旧式)」『Mobile Link - サーバ管理』を参照してください。

setNewPassword メソッド

setUserName で指定されたユーザの新しい Mobile Link パスワードを設定します。

構文

setNewPassword(String *value* **)**

パラメータ

- **value**　Mobile Link ユーザの新しいパスワード。

備考

新しいパスワードが有効になるのは、次の同期の後です。

setPassword メソッド

setUserName で指定されたユーザの Mobile Link パスワードを設定します。

構文

setPassword(String *value* **)**

パラメータ

- **value** Mobile Link ユーザのパスワード。

備考

このユーザ名とパスワードは他のデータベース・ユーザ ID やパスワードとは別のもので、アプリケーションを Mobile Link サーバに対して識別し、認証するために使用されます。

setPingOnly メソッド

実際に同期を行う代わりに、クライアントが Mobile Link サーバに ping のみを行うかどうかを指定します。

構文

setPingOnly(Boolean *value* **)**;

パラメータ

- **value** Mobile Link サーバに ping のみを実行する場合は true、同期を実行する場合は false に設定します。

setSendColumnNames メソッド

同期中に、クライアントが Mobile Link サーバにカラム名を送信するかどうかを指定します。

構文

setSendColumnNames(Boolean *value* **)**

パラメータ

- **value** カラム名を送信する場合は true、送信しない場合は false に設定します。

備考

このパラメータは、ダイレクト・ロー・ハンドリングで使用されます。

setSendDownloadAck メソッド

同期中に、クライアントが Mobile Link サーバにダウンロード確認を送信するかどうかを指定します。

構文

setSendDownloadAck(Boolean *value* **)**

パラメータ

- **value**　ダウンロード確認 (正または負) を送信する場合は true、送信しないことをサーバに通知する場合は false に設定します。

備考

ダウンロード確認は、ダウンロードがリモートで完全に適用されてコミットされた後 (正の確認)、またはダウンロードに失敗した後 (負の確認) に送信されます。

クライアントがダウンロード確認を送信する場合、Mobile Link サーバのデータベース・ワーカ・スレッドは、クライアントがダウンロードを適用してコミットするまで待機します。クライアントがダウンロード確認を送信しない場合、Mobile Link サーバは、次の同期のため、より早く解放されます。

setStream メソッド

同期に使用するように Mobile Link 同期ストリームを設定します。

構文

setStream(UInt16 *value* **)**

パラメータ

- **value**　同期に使用する Mobile Link 同期ストリームのタイプ。有効な選択肢のリストについては、「定数」 130 ページを参照してください。

備考

ほとんどの同期ストリームでは、Mobile Link サーバのアドレスを識別したり、その他の動作を制御したりするパラメータが必要です。これらのパラメータは、**setStreamParms()** メソッドで指定します。

デフォルトのストリーム・タイプは STREAM_TYPE_TCPIP です。

setStreamParms メソッド

同期ストリームの設定パラメータを設定します。

構文

setStreamParms(String *value* **)**

パラメータ

- **value**　同期ストリームに使用されるすべてのネットワーク・プロトコル・オプションを含む文字列。オプションは、name=value のペアをセミコロンで区切ったリスト (**"param1=value1;param2=value2"**) で指定します。

備考

特定のストリームのタイプの設定方法については、「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。

setUploadOnly メソッド

同期時のダウンロードを無効にするか、有効にするかを指定します。

構文

setUploadOnly(Boolean *value* **)**

パラメータ

- **value**　ダウンロードを無効にする場合は true、有効にする場合は false に設定します。

setUserName メソッド

Mobile Link サーバが Mobile Link クライアントをユニークに識別するユーザ名を設定します。

構文

setUserName(String *value* **)**

パラメータ

- **value**　Mobile Link ユーザ名。

備考

Mobile Link では、この値を使用して、ダウンロードする内容の決定、同期ステータスの記録、同期中の割り込みからの復帰を行います。このユーザ名とパスワードは他のデータベース・ユーザ ID やパスワードとは別のもので、アプリケーションを Mobile Link サーバに対して識別し、認証するために使用されます。

setVersion メソッド

使用する同期スクリプト・バージョンを指定します。

構文

setVersion(String *value* **)**

パラメータ

- **value**　スクリプト・バージョン文字列。

備考

統合データベースの同期スクリプトは、それぞれバージョン文字列でマーク付けされます。たとえば、異なるバージョン文字列によって特定される 2 つの download_cursor スクリプトが存在する場合があります。Ultra Light アプリケーションは、バージョン文字列により、同期スクリプトのセットから選択できます。

SyncResult クラス

前回の同期の Ultra Light for M-Business Anywhere メソッドのステータスを表します。それぞれの接続には、固有の SyncResult インスタンスがあります。

このクラスを直接インスタンス化することはできません。

getAuthStatus メソッド

前回試行された同期の認証ステータス・コードを返します。

構文

UInt16 **getAuthStatus()**

getIgnoredRows メソッド

前回の同期中にアップロードされたローが無視された場合は true、アップロードされたローが無視されなかった場合は false を返します。

構文

Boolean **getIgnoredRows()**

パラメータ

- **return**　アップロードされたローが無視された場合は true、無視されなかった場合は false。

getPartialDownloadRetained メソッド

ダウンロードが中断され、部分的なダウンロードが保持された場合は true、ダウンロードが中断されなかった場合または部分的なダウンロードがロールバックされた場合は false を返します。

構文

Boolean getPartialDownloadRetained()

getStreamErrorCode メソッド

同期ストリーム処理によってレポートされるエラー・コードを表す整数を返します。

構文

UInt16 **getStreamErrorCode( )**

パラメータ

- **return**　同期ストリームによってレポートされるエラー・コード。

備考

「Mobile Link 通信エラー・メッセージ」『エラー・メッセージ』を参照してください。

getStreamErrorParameters メソッド

ストリーム・エラー・パラメータをカンマで区切ったリストを返します。

構文

String **StreamErrorParameters ( )**

パラメータ

- **return**　同期ストリーム処理によってレポートされるエラー・パラメータをカンマで区切ったリスト。

参照

- 「getStreamErrorCode メソッド」 141 ページ

getStreamErrorSystem メソッド

ストリーム・エラー・システム固有のコードを返します。

構文

UInt16 **getStreamErrorSystem( )**

パラメータ

- **return**　システム固有のエラー・コード。

getTimestamp メソッド

前回の同期のタイムスタンプを返します。

構文

Date **getTimestamp( )**

getUploadOK メソッド

前回のアップロード同期が成功であった場合は true、不成功であった場合は false を返します。

構文

Boolean **getUploadOK()**

TableSchema クラス

Ultra Light のテーブルのスキーマを表します。

getColumnCount メソッド

このテーブル内の 1 から始まるカラム番号を返します。

構文

UInt16 **getColumnCount( )**

備考

カラム ID の範囲は、1 ～ getColumnCount() です。

getColumnDefaultValue メソッド

指定されたカラムのデフォルト値、またはデフォルト値が **null** の場合は NULL を返します。

構文

String **getColumnDefaultValue(** String *name* **)**

パラメータ

- **name**　カラムの名前。

getColumnDefaultValueByColID メソッド

カラムのデフォルト値、またはデフォルト値が **null** の場合は NULL を返します。

構文

String **getColumnDefaultValueByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

getColumnID メソッド

指定されたカラムの 1 から始まる ID を返します。

構文

UInt16 **getColumnID(** String *name* **)**

パラメータ

- **name**　カラムの名前。

getColumnName メソッド

指定されたカラムの名前を返します。

構文

String **getColumnName(** UInt16 *colID* **)**

パラメータ

- **colID**　カラムの 1 から始まるカラム ID。

getColumnPartitionSize メソッド

カラムのグローバル・オートインクリメントの分割サイズを Double で表される 64 ビット符号なし数値として返します。

構文

UInt64 **getColumnPartitionSize(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

テーブルのすべてのグローバル・オートインクリメント・カラムは、同じグローバル・オートインクリメントの分割サイズを共有します。

getColumnPartitionSizeByColID メソッド

カラムのグローバル・オートインクリメントの分割サイズを Double で表される 64 ビット符号なし数値として返します。

構文

UInt64 **getColumnPartitionSizeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

テーブルのすべてのグローバル・オートインクリメント・カラムは、同じグローバル・オートインクリメントの分割サイズを共有します。

getColumnPrecision メソッド

カラムの精度を返します。

構文

Int32 **getColumnPrecision(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnPrecisionByColID メソッド

カラムの精度を返します。

構文

Int32 **getColumnPrecisionByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnScale メソッド

カラムの位取りを返します。

構文

Int32 **getColumnScale(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnScaleByColID メソッド

カラムの位取りを返します。

構文

Int32 **getColumnScaleByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.NUMERIC 型である必要があります。

getColumnSize メソッド

カラムのサイズを返します。

構文

UInt32 **getColumnSize(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.BINARY 型、または SQLType.CHAR 型である必要があります。

getColumnSizeByColID メソッド

カラムのサイズを返します。

構文

UInt32 **getColumnSizeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.BINARY 型、または SQLType.CHAR 型である必要があります。

getColumnSQLType メソッド

カラムの SQLType を示す SQLType 列挙整数を返します。

構文

Int16 **getColumnSQLType(** String *name* **)**

パラメータ

- **name**　カラムの名前。

getColumnSQLTypeByColID メソッド

カラムの SQLType を示す SQLType 列挙整数を返します。

構文

Int16 **getColumnSQLTypeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

getIndex メソッド

指定されたインデックスのインデックス・スキーマを返します。

構文

IndexSchema **getIndex(** String *name* **)**

パラメータ

- **name**　インデックスの名前。

getIndexCount メソッド

このテーブルのインデックス数を返します。

構文

UInt16 **getIndexCount( )**

備考

インデックス ID の範囲は、1 ～ **getIndexCount()** です。

注意：インデックスの ID とカウントは、スキーマのアップグレード中に変更されることがあります。インデックスを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getIndexName メソッド

指定されたインデックス ID で識別されたインデックスの名前を返します。

構文

String **getIndexName(** UInt16 *indexID* **)**

パラメータ

- **indexID**　インデックスの ID。**indexID** は、**[1,getIndexCount()]** の範囲内であることが必要です。

備考

注意：インデックスの ID とカウントは、スキーマのアップグレード中に変更されることがあります。インデックスを正しく識別するには、名前でアクセスするか、キャッシュされている ID とカウントをスキーマのアップグレード後にリフレッシュします。

getName メソッド

このテーブルの名前を返します。

構文

String **getName( )**

getOptimalIndex メソッド

指定されたカラムを使用してテーブルを検索するために最適なインデックスを返します。

構文

IndexSchema **getOptimalIndex(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

指定したカラムは、インデックス内の最初のカラムですが、インデックスには複数のカラムがある場合があります。

getPrimaryKey メソッド

このテーブルのプライマリ・キーのインデックス・スキーマを返します。

構文

IndexSchema **getPrimaryKey( )**

getUploadUnchangedRows メソッド

テーブルが、すべてのローをアップロードするようにマーク付けされている場合は true、一部のローをアップロードしないようにマーク付けされている場合は false を返します。

構文

Boolean **getUploadUnchangedRows( )**

備考

このメソッドで true が返されるテーブルでは、その同期時に、変更済みのローと未変更のローが常にアップロードされます。このようなテーブルは、「完全同期」テーブルと呼ばれることもあります。

isColumnAutoIncrement メソッド

カラムがオートインクリメントされる場合は true、オートインクリメントされない場合は false を返します。

構文

Boolean **isColumnAutoIncrement(** String *name* **)**

パラメータ

- **name**　カラムの名前。

isColumnAutoIncrementByColID メソッド

カラムがオートインクリメントされる場合は true、オートインクリメントされない場合は false を返します。

構文

Boolean **isColumnAutoIncrementByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

isColumnCurrentDate メソッド

カラムのデフォルトが現在の日付に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentDate(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.DATE 型である必要があります。

isColumnCurrentDateByColID メソッド

カラムのデフォルトが現在の日付に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentDateByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.DATE 型である必要があります。

isColumnCurrentTime メソッド

カラムのデフォルトが現在の時刻に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentTime(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.TIME 型である必要があります。

isColumnCurrentTimeByColID メソッド

カラムのデフォルトが現在の時刻に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentTimeByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.TIME 型である必要があります。

isColumnCurrentTimestamp メソッド

カラムのデフォルトが現在のタイムスタンプに設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentTimestamp(** String *name* **)**

パラメータ

- **name**　カラムの名前。

備考

カラムは SQLType.TIMESTAMP 型である必要があります。

isColumnCurrentTimestampByColID メソッド

カラムのデフォルトが現在のタイムスタンプに設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnCurrentTimestampByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

カラムは SQLType.TIME 型である必要があります。

isColumnGlobalAutoIncrement メソッド

カラムのデフォルトがグローバル・オートインクリメントに設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnGlobalAutoIncrement(** String *name* **)**

パラメータ

- **name**　カラムの名前。
- **return**　カラムがグローバル・オートインクリメントされる場合は true、グローバル・オートインクリメントされない場合は false。

isColumnGlobalAutoincrementByColID メソッド

カラムのデフォルトがグローバル・オートインクリメントに設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnGlobalAutoincrementByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

isColumnNewUUID メソッド

カラムのデフォルトが新しい UUID に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnNewUUID(** String *name* **)**

パラメータ

- **name**　カラムの名前。

isColumnNewUUIDByColID メソッド

カラムのデフォルトが新しい UUID に設定されている場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnNewUUIDByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

isColumnNullable メソッド

カラムが NULL 入力可の場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnNullable(** String *name* **)**

パラメータ

- **name**　カラムの名前。

isColumnNullableByColID メソッド

カラムが NULL 入力可の場合は true、そうでない場合は false を返します。

構文

Boolean **isColumnNullableByColID(** UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

isInPublication メソッド

テーブルがパブリケーション内にある場合は true、パブリケーション内にない場合は false を返します。

構文

Boolean **isInPublication(** String *pubName* **)**

パラメータ

- **pubName**　パブリケーションの名前。

isNeverSynchronized メソッド

テーブルが、まったく同期されないようにマーク付けされている場合は true、まったく同期されないようにマーク付けされていない場合は false を返します。

構文

Boolean **isNeverSynchronized( )**

備考

このメソッドで true が返されるテーブルは、パブリケーションに含まれている場合でもまったく同期されません。このようなテーブルは、「非同期」テーブルと呼ばれることもあります。

ULTable クラス

Ultra Light テーブルを表します。

プロパティ

このクラスのプロパティは、次のとおりです。

プロパティ	説明
TableSchema schema (読み込み専用)	この結果セットのスキーマ。このプロパティが有効なのは、その準備文が開かれている間だけです。
NULL_TIMESTAMP_VAL	タイムスタンプ値が NULL であることを示す定数。

appendBytes メソッド

指定されたバイト配列の指定されたサブセットを、指定された SQLType.LONGBINARY カラムの新しい値に追加します。

構文

```
appendBytes(
  UInt16 columnID,
  Array value,
  UInt32 srcOffset,
  UInt32 count
)
```

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。
- **srcOffset**　カラムの現在の新しい値に追加する値。
- **count**　コピーされるバイト数。

備考

配列 **value** の srcOffset (0 から始まります) から **srcOffset+count-1** までの位置のバイトが、指定されたカラムの値に追加されます。挿入時には、insertBegin は新しい値をカラムのデフォルト値に初期化します。ローのデータは、**insert** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

次のいずれかに該当する場合、コード SQLCode.SQLE_INVALID_PARAMETER とともにエラーがスローされ、追加先は修正されません。

- **value** 引数が NULL である
- **srcOffset** 引数が負の値である
- **count** 引数が負の値である
- **srcOffset+count** がソース配列の長さ **value.length** よりも大きい

その他のエラーの場合は、それに応じたエラー・コードとともに **SQLException** がスローされます。

appendStringChunk メソッド

指定された文字列を指定された SQLType.LONGVARCHAR カラムの新しい値に追加します。

構文

```
appendStringChunk(
  UInt16 columnID,
  String value
)
```

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

例

次の文は、文字列 **XYZ** のインスタンス 100 個を最初のカラムの値に追加します。

```
for ( I = 0; I < 100; I++ ){
  t.appendStringChunk( 1, "XYZ" );
}
```

deleteRow メソッド

現在の行を削除します。

構文

```
deleteRow( )
```

deleteAllRows メソッド

テーブルのすべてのローを削除します。

構文

```
deleteAllRows( )
```

備考

アプリケーションによっては、テーブル内のローをすべて削除してから、新しいデータ・セットをテーブルにダウンロードする方が便利なことがあります。Connection.startSynchronizationDelete メソッドを使用すると、統合データベースからは削除しないで Ultra Light データベースからローを削除できます。

findBegin メソッド

このテーブルで新規に検索を実行する準備を行います。

構文

findBegin( )

備考

検索する値は、このテーブルを開いたインデックス内のカラムで適切な set*Type* メソッドを呼び出して指定します。

findFirst メソッド

テーブルを先頭から順方向に移動しながら、現在のインデックスの値かそのセット全体に完全に一致するローを検索します。

構文

Boolean **findFirst( )**

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値と完全に一致した最初のローで停止します。失敗すると、カーソル位置は最後のローの後ろ (isEOF) になります。

検索を行う前に findBegin メソッドを呼び出してください。

参照

- 「findBegin メソッド」 158 ページ
- 「isEOF メソッド」 105 ページ

findFirstForColumns メソッド

テーブルを先頭から順方向に移動しながら、現在のインデックスの値かそのセットの一部に完全に一致するローを検索します。

構文

Boolean **findFirstForColumns(**
UInt16 *numColumns*
)

パラメータ

- **numColumns** 複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定してから、**numColumns** 値を **1** に指定します。

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値と完全に一致した最初のローで停止します。失敗すると、カーソル位置は最後のローの後ろ (isEOF) になります。

検索を行う前に findBegin メソッドを呼び出してください。

参照

- 「findBegin メソッド」 158 ページ
- 「isEOF メソッド」 105 ページ

findLast メソッド

テーブルを最後から逆方向に移動しながら、現在のインデックスの値またはそのセット全体に完全に一致するローを検索します。

構文

Boolean **findLast( )**

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値と完全に一致した最初のローで停止します。失敗すると、カーソル位置は最初のローの前 (isBOF) になります。

検索を行う前に findBegin メソッドを呼び出してください。

参照

- 「findBegin メソッド」 158 ページ
- 「isBOF メソッド」 105 ページ

findLastForColumns メソッド

テーブルを最後から逆方向に移動しながら、現在のインデックスの値またはそのセットの一部に完全に一致するローを検索します。

構文

Boolean **findLastForColumns(** UInt16 *numColumns* **)**

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定してから、**numColumns** 値を **1** に指定します。

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値と完全に一致した最初のローで停止します。失敗すると、カーソル位置は最初のローの前 (isBOF) になります。

検索を行う前に findBegin メソッドを呼び出してください。

参照

- 「findBegin メソッド」 158 ページ
- 「isBOF メソッド」 105 ページ

findNext メソッド

現在の位置からテーブルを順方向に移動しながら、次のローが現在のインデックスの値かそのセット全体に完全に一致するかどうかを調べて、findFirst 検索を続行します。

構文

Boolean **findNext( )**

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

インデックスの値と完全に一致すると、カーソルは次のローで停止します。失敗すると、カーソル位置は最後のローの後ろ (isEOF) になります。

検索されるカラムの値がローの更新において修正された場合の findNext メソッドの動作は不確定です。

findNextForColumns メソッド

現在の位置からテーブルを順方向に移動しながら、次のローが現在のインデックスの値かそのセットの一部に完全に一致するかどうかを調べて、findFirst 検索を続行します。

構文

Boolean **findNextForColumns(** UInt16 *numColumns*)

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定してから、**numColumns** 値を **1** に指定します。

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

インデックスの値と完全に一致すると、カーソルは次のローで停止します。失敗すると、カーソル位置は最後のローの後ろ (isEOF) になります。

検索されるカラムの値がローの更新において修正された場合の findNext メソッドの動作は不確定です。

findPrevious メソッド

現在の位置からテーブルを逆方向に移動しながら、前のローが現在のインデックスの値かそのセット全体に完全に一致するかどうかを調べて、findLast 検索を続行します。

構文

Boolean **findPrevious( )**

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

インデックスの値と完全に一致すると、カーソルは前のローで停止します。失敗すると、カーソル位置は最初のローの前 (isBOF) になります。

検索されるカラムの値がローの更新において修正された場合の findPrevious メソッドの動作は不確定です。

findPreviousForColumns メソッド

現在の位置からテーブルを逆方向に移動しながら、前のローが現在のインデックスの値かそのセットの一部に完全に一致するかどうかを調べて、findLast 検索を続行します。

構文

Boolean **findPreviousForColumns(**
  UInt16 *numColumns*
)

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定してから、**numColumns** 値を **1** に指定します。

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

インデックスの値と完全に一致すると、カーソルは前のローで停止します。失敗すると、カーソル位置は最初のローの前 (isBOF) になります。

検索されるカラムの値がローの更新において修正された場合の findPrevious メソッドの動作は不確定です。

getBoolean メソッド

指定されたカラムの値を Boolean として返します。

構文

Boolean **getBoolean(** UInt16 *index* **)**

パラメータ

- **index**　カラムの ID 番号。結果セットの最初のカラムの ID は 1 です。

getBytes メソッド

指定されたカラムの値のバイト配列を返します。

構文

Array **getBytes(** UInt16 *index* **)**

パラメータ

- **index**　カラムの ID 番号。結果セットの最初のカラムの ID は 1 です。

備考

SQLType.BINARY 型、SQLType.LONGBINARY 型のカラムの場合にのみ有効です。

getBytesSection メソッド

指定されたオフセットで始まる、指定された SQLType.LONGBINARY カラムの内容のサブセットを、コピー先のバイト配列の指定されたオフセットにコピーします。

構文

```
UInt32 getBytesSection(
  UInt16 index
  UInt32 srcOffset,
  Array  dst,
  UInt32 dstOffset,
  UInt32 count
)
```

パラメータ

index　バイナリ・データを含むカラムの 1 から始まる順序。

srcOffset　カラム値の開始位置。最初の値は 0 です。

dst　コピー先の配列。

dstOffset　コピー先の配列の開始位置。

count　コピーされるバイト数。

戻り値

読み込まれたバイト数。

備考

コピー元のカラムの srcOffset (0 から始まります) から srcOffset+count-1 までの位置のバイトが、コピー先の配列の dstOffset から dstOffset+count-1 までの位置に、それぞれコピーされます。count のバイト数がコピーされる前に、値の末尾が検出された場合は、コピー先の配列の残りは変更されないままになります。

次のいずれかに該当する場合、エラーがスローされ、Connection.sqlCode が SQLError.SQLE_INVALID_PARAMETER に設定され、コピー先は修正されません。

- dst 引数が NULL である
- srcOffset 引数が負の値である
- dstOffset 引数が負の値である
- count 引数が負の値である
- dstOffset + count がコピー先の配列の長さ dst.length よりも長い

getDate メソッド

Date として値を返します。

構文

Date **getDate(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getDouble メソッド

Double として値を返します。

構文

Double **getDouble(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getFloat メソッド

指定されたカラムの値を返します。

構文

Float **getFloat(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getInt メソッド

指定されたカラムの値を返します。

構文

Int32 **getInt(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getLong メソッド

指定されたカラムの値を返します。

構文

Int64 **getLong(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getRowCount メソッド

結果セット内のロー数を返します。

構文

UInt32 **getRowCount( )**

getRowCountWithThreshold メソッド

指定されたスレッショルドのロー数以内で、テーブル内のロー数を返します。

構文

UInt32 **getRowCount(** UInt32 *threshold***)**

パラメータ

threshold　この値は、ロー・カウント操作に上限値を指定します。スレッショルド値を超える個数のローがある場合、結果はスレッショルド値になります。ローが多い場合、ローのカウントは負荷の高い操作になることがあります。一部のアプリケーションでは、特定のロー数を超えているかどうかを確認するだけで済み (たとえば、ユーザがより多くのローを要求するためのオプションを提供するかどうかを決定する場合など)、正確なロー・カウントは必要としません。この値が 0 の場合、すべてのローがカウントされます。

getShort メソッド

Int16 として値を返します。

構文

Int16 **getShort(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getString メソッド

String として値を返します。

構文

String **getString(** UInt32 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getStringChunk メソッド

指定されたオフセットで始まる、指定された SQLType.LONGVARCHAR カラムの値のサブセットを String オブジェクトにコピーします。

構文

String **getStringChunk(**
  UInt16 *index*,
  UInt32 *srcOffset*,
  UInt32 *count*
)

パラメータ

- **index**　結果セットで取得する 1 から始まる順序。
- **srcOffset**　文字列値で 0 から始まる開始位置。
- **count**　コピーされる文字数。

戻り値

指定された文字数がコピーされた文字列。

getTime メソッド

Date として値を返します。

構文

Date **getTime(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getTimestamp メソッド

Date として値を返します。

構文

Date **getTimestamp(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getULong メソッド

64 ビット符号なし整数として値を返します。

構文

UInt64 **getULong(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

getUUID メソッド

UUID としてカラムの値を返します。

構文

UUID **getUUID(** UInt16 *index* **)**

パラメータ

index　結果セットで取得する 1 から始まる順序。

備考

カラムは長さが 16 の SQLType.BINARY 型である必要があります。

insert メソッド

現在のカラム値 (set メソッドを使用して指定されます) で新しいローを挿入します。

構文

insert()

備考

挿入を行う前に insertBegin を呼び出してください。

insertBegin メソッド

現在のすべてのカラムをデフォルト値に設定して、このテーブルに新しいローを挿入する準備を行います。

構文

insertBegin( )

備考

適切な set*Type* メソッドを呼び出して、挿入するデフォルト以外の値を指定します。

insert メソッドが実行されないと、ローが実際に挿入されることも、ロー内のデータが実際に変更されることもありません。また、その変更がコミットされないかぎり、永続化されません。

lookupBackward メソッド

テーブルを最後から逆方向に移動しながら、現在のインデックスの値またはそのセット全体に一致するか、その値より小さい値を持つローを検索します。

構文

Boolean **lookupBackward( )**

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値に一致する最初のローか、それより少ない値の最初のローで停止します。失敗した場

合 (検索する値より小さい値のローがない場合)、カーソル位置は最初のローの前 (isBOF) になります。

検索を行う前に lookupBegin メソッドを呼び出してください。

lookupBackwardForColumns メソッド

テーブルを最初から逆方向に移動しながら、現在のインデックスの値またはそのセットの一部に一致するか、その値より小さい値を持つローを検索します。

構文

Boolean **lookupBackwardForColumns(** UInt16 *numColumns* **)**

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定してから、**numColumns** 値を **1** に指定します。

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値に一致する最初のローか、それより少ない値の最初のローで停止します。失敗した場合 (検索する値より小さい値のローがない場合)、カーソル位置は最初のローの前 (isBOF) になります。

検索を行う前に lookupBegin メソッドを呼び出してください。

lookupBegin メソッド

このテーブルで新規に検索を実行する準備を行います。

構文

lookupBegin( )

備考

検索する値は、このテーブルを開いたインデックス内のカラムで適切な set*Type* メソッドを呼び出して指定します。

lookupForward メソッド

テーブルを最初から順方向に移動しながら、現在のインデックスの値またはそのセット全体に一致するか、その値より大きい値を持つローを検索します。

構文

Boolean **lookupForward( )**

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値に一致するか、それより大きい値の最初のローで停止します。失敗した場合 (検索する値より大きい値のローがない場合)、カーソル位置は最後のローの後ろ (isEOF) になります。

検索を行う前に lookupBegin メソッドを呼び出してください。

lookupForwardForColumns メソッド

テーブルを最初から順方向に移動しながら、現在のインデックスの値またはそのセットの一部に一致するか、その値より大きい値を持つローを検索します。

構文

Boolean **lookupForwardForColumns(** UInt16 *numColumns* **)**

パラメータ

- **numColumns**　複合インデックスのための、ルックアップで使用するカラムの数。たとえば、3 つのカラムのインデックスがあり、最初のカラムにのみ基づいて一致する値を検索する場合は、最初のカラムに値を設定してから、**numColumns** 値を **1** に指定します。

戻り値

成功の場合は **true**、それ以外の場合は **false**。

備考

検索する値を指定するには、インデックスの各カラムに値を設定します。カーソルは、インデックスの値に一致するか、それより大きい値の最初のローで停止します。失敗した場合 (検索する値より大きい値のローがない場合)、カーソル位置は最後のローの後ろ (isEOF) になります。

検索を行う前に lookupBegin メソッドを呼び出してください。

isBOF メソッド

成功の場合は **true**、それ以外の場合は **false** を返します。

構文

Boolean **isBOF( )**

isEOF メソッド

成功の場合は **true**、それ以外の場合は **false** を返します。

構文

Boolean **isEOF( )**

isNull メソッド

値が NULL の場合は **true**、NULL 以外の場合は **false** を返します。

構文

Boolean **isNull(** UInt16 *index* **)**

パラメータ

index　カラムのインデックス値。

isOpen メソッド

ResultSet が開いている場合は **true**、それ以外の場合は **false** を返します。

構文

Boolean **isOpen( )**

moveAfterLast メソッド

ULResultSet の最後のローの後に移動します。

構文

Boolean **moveAfterLast( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveBeforeFirst メソッド

最初のローの前に移動します。

構文

Boolean **moveBeforeFirst( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveFirst メソッド

最初のローに移動します。

構文

Boolean **moveFirst( )**

戻り値

成功の場合は **True**。

失敗の場合は **False**。たとえば、ローがない場合、メソッドは失敗します。

moveLast メソッド

最後のローに移動します。

構文

Boolean **moveLast( )**

戻り値

成功の場合は **True**。

失敗の場合は **False**。たとえば、ローがない場合、メソッドは失敗します。

moveNext メソッド

次のローに移動します。

構文

Boolean **moveNext( )**

戻り値

成功の場合は **True**。

失敗の場合は **False**。たとえば、ローがない場合、メソッドは失敗します。

movePrevious メソッド

前のローに移動します。

構文

Boolean **movePrevious( )**

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

moveRelative メソッド

いくつかのローを、現在のローを基準にして相対的に移動します。

構文

Boolean **moveRelative(** Int32 *index* **)**

パラメータ

index　移動するローの数。値は、正、負、または 0 です。

戻り値

成功の場合は **true**。

失敗の場合は **false**。たとえば、ローがない場合、メソッドは失敗します。

備考

結果セットでのカーソルの現在位置を基準にして、正のインデックス値は結果セット内を前に移動し、負のインデックス値は結果セット内を後ろに移動し、0 はカーソルを移動しません。

open メソッド

プライマリ・キーを使用して、このテーブルをデータ・アクセス用に開きます。

構文

open()

openWithIndex メソッド

指定されたインデックスを使用して、このテーブルをデータ・アクセス用に開きます。

構文

openWithIndex(String *index*)

パラメータ

- **index**　テーブルを開くインデックスの名前。NULL の場合は、プライマリ・キーが使用されます。

setBoolean メソッド

指定されたカラムの値を、**boolean** を使用して設定します。

構文

setBoolean(short *columnID*, boolean *value*)

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、**insert** または **update** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を **false** に設定します。

```
t.setBoolean( 1, false );
```

setBytes メソッド

指定されたカラムの値を、**byte** 配列を使用して設定します。

構文

setBytes(UInt16 *columnID*, Array *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

SQLType.BINARY 型、**SQLType.LONGBINARY** 型のカラムにのみ適しています。ローのデータは、**insert** または **update** が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を設定します。

```
var blob = new Array( 3 );
blob[ 0 ] = 78;
blob[ 1 ] = 0'
blob[ 2 ] = 68;
t.setBytes( 1, blob );
```

setDate メソッド

指定されたカラムの値を、**Date** を使用して設定します。

構文

setDate(UInt16 *columnID*, Date *value***)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 2004/09/27 に設定します。

```
t.setDate(
  1, new Date( 2004,9,27,0,0,0,0 )
);
```

setDouble メソッド

指定されたカラムの値を、**double** を使用して設定します。

構文

setDouble(UInt16 *columnID*, Double *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の例は、最初のカラムの値を設定します。

```
t.setDouble( 1, Number.MAX_VALUE );
```

setFloat メソッド

指定されたカラムの値を、Float を使用して設定します。

構文

setFloat(UInt16 *columnID*, Float *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を設定します。

```
t.setFloat(
   1,
   (2 - Math.pow(2,-23)) * Math.pow(2,127)
);
```

setInt メソッド

指定されたカラムの値を、Integer を使用して設定します。

構文

setInt(UInt16 *columnID*, Int32 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 2147483647 に設定します。

```
t.setInt( 1, 2147483647 );
```

setLong メソッド

指定されたカラムの値を、Int64 を使用して設定します。

構文

setLong(UInt16 *columnID*, Int64 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 9223372036854770000 に設定します。

```
t.setLong( 1, 9223372036854770000 );
```

setNull メソッド

カラムに SQL NULL を設定します。

構文

setNull(UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

データは、insert または update を実行するまでは、実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setShort メソッド

指定されたカラムの値を、UInt16 を使用して設定します。

構文

setShort(UInt16 *columnID*, Int16 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 32767 に設定します。

```
t.setShort( 1, 32767 );
```

setString メソッド

指定されたカラムの値を、String を使用して設定します。

構文

setString(UInt16 *columnID*, String *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を **abc** に設定します。

```
t.setString( 1, "abc" );
```

setTime メソッド

指定されたカラムの値を、Date を使用して設定します。

構文

setTime(UInt16 *columnID*, Date *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 18:02:13:0000 に設定します。

```
t.setTime(
  1, new Date( 1966,4,1,18,2,13,0 )
);
```

setTimestamp メソッド

指定されたカラムの値を、Date を使用して設定します。

構文

setTimestamp(UInt16 *columnID*, Date *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を 1966/04/01 18:02:13:0000 に設定します。

```
t.setTimestamp(
  1, new Date( 1966,4,1,18,2,13,0 )
);
```

setToDefault メソッド

指定されたカラムの値を、そのデフォルト値に設定します。

構文

setToDefault(UInt16 *columnID* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

setULong メソッド

指定されたカラムの値を、符号なし値として扱われる 64 ビット整数を使用して設定します。

構文

setULong(UInt16 *columnID*, UInt64 *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

例

次の文は、最初のカラムの値を設定します。

```
t.setULong( 1, 9223372036854770000 * 4096 );
```

setUUID メソッド

指定されたカラムの値を、UUID を使用して設定します。

構文

setUUID(UInt16 *columnID*, UUID *value* **)**

パラメータ

- **columnID**　カラムの ID 番号。テーブルの最初のカラムの ID 値は 1 です。
- **value**　カラムの新しい値。

備考

ローのデータは、insert または update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。長さが 16 の SQLType.BINARY 型のカラムの場合にのみ有効です。

例

次の文は、テーブルで最初のカラムに新しい UUID 値を設定します。

```
t.setUUID( 1, conn.getNewUUID(); );
```

参照

- 「UUID の使用」『Mobile Link - サーバ管理』

truncate メソッド

テーブル内のすべてのローを削除し、STOP SYNCHRONIZATION DELETE を一時的にアクティブにします。

構文

truncate()

update メソッド

現在のカラム値 (set メソッドを使用して指定されます) で新しいローを更新します。

構文

update()

備考

update を行う前に updateBegin を呼び出してください。

updateBegin メソッド

このテーブルの現在のローを更新する準備を行います。

構文

updateBegin()

備考

カラム値は、適切な set*Type* メソッドを呼び出すことによって修正します。

ローのデータは、update が実行されるまで実際には変更されません。また、その変更がコミットされないかぎり、永続化されません。

テーブルを開くのに使用されるインデックス内のカラムを修正すると、アクティブな検索処理に予期しない影響を及ぼします。テーブルのプライマリ・キー内のカラムは更新できません。

UUID クラス

UUID を説明します。UUID (ユニバーサル・ユニーク識別子) または GUID (グローバル・ユニーク識別子) は、すべてのコンピュータやデータベースの間でユニークになるように生成された値です。UUID は、SQLType.BINARY(16) 値として Ultra Light データベースに格納され、ローをユニークに識別するために使用できます。UUID クラスは不変の UUID を格納します。

UUID は、その UUID を作成した Connection と関連付けされ、その接続が閉じると文字列に変換できなくなります。

equals メソッド

この UUID が比較対象の引数と同じ場合は **true**、同じでない場合は **false** を返します。

構文

Boolean **equals(** UUID *other* **)**

パラメータ

- **other**　比較する UUID。

toString メソッド

この UUID の文字列表現を返します。

構文

String **toString()**

備考

文字列は、フォーマット XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX (X は 16 進数の数字)、またはこの UUID に関連付けられた Connection が閉じている場合は NULL です。

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」 192 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」 197 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」 201 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)」 201 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」 209 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」 189 ページ
- 「jConnect」 189 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」 189 ページ
- 「iAnywhere JDBC ドライバ」 188 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」 189 ページ
- 「iAnywhere JDBC ドライバ」 188 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」 197 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」 192 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」 216 ページ
- 「同期」 216 ページ
- 「Ultra Light」 193 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」 197 ページ
- 「Listener」 189 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」 191 ページ
- 「サーバ起動同期」 197 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」 197 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」 187 ページ。

Replication Agent

参照：「LTM」 189 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」 189 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」 199 ページ
- 「SQL」 192 ページ
- 「データベース管理システム (DBMS)」 201 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」 193 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」 209 ページ
- 「パブリケーション」 204 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」 203 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」 196 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」 217 ページ
- 「コード・ページ」 197 ページ
- 「照合」 214 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」 193 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」 196 ページ
- 「カーソル位置」 195 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」 195 ページ
- 「カーソル結果セット」 196 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」 195 ページ
- 「カーソル位置」 195 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」 192 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」 202 ページ
- 「ローカル・テンポラリ・テーブル」 210 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」 197 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」 217 ページ
- 「エンコード」 195 ページ
- 「照合」 214 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」 204 ページ
- 「Mobile Link ユーザ」 190 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」 198 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」 198 ページ
- 「生成されたジョイン条件」 215 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」 194 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」 217 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」 216 ページ
- 「SQL ベースの同期」 193 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」 211 ページ
- 「プライマリ・キー」 206 ページ
- 「データベース管理システム (DBMS)」 201 ページ
- 「リレーショナル・データベース管理システム (RDBMS)」 209 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」 187 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)」 209 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)」 201 ページ
- 「SYS」 193 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」 201 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」 202 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」 197 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」 210 ページ
- 「グローバル・テンポラリ・テーブル」 196 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」 202 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」 203 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」 210 ページ
- 「文レベルのトリガ」 217 ページ
- 「整合性」 214 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」 204 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」 209 ページ
- 「アーティクル」 194 ページ
- 「パブリケーションの更新」 204 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」 209 ページ
- 「パブリケーション」 204 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」 209 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」 214 ページ
- 「ユーザ定義データ型」 208 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」 205 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」 211 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」 214 ページ
- 「検査制約」 213 ページ
- 「外部キー制約」 212 ページ
- 「一意性制約」 211 ページ
- 「整合性」 214 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」 193 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」 194 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」 207 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」 202 ページ
- 「ビュー」 205 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」 197 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」 207 ページ
- 「ビュー」 205 ページ

ミラー・ログ

参照：「トランザクション・ログ・ミラー」 203 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」 199 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」 209 ページ
- 「FILE」 188 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」 196 ページ
- 「サーバ・メッセージ・ストア」 197 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」 209 ページ
- 「統合データベース」 216 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」 202 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」 197 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」 216 ページ
- 「統合データベース」 216 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)」 201 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」 209 ページ
- 「パブリケーションの更新」 204 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」 209 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」 203 ページ
- 「文レベルのトリガ」 217 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」 202 ページ
- 「グローバル・テンポラリ・テーブル」 196 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」 203 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」 211 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」 203 ページ
- 「トランザクション・ログ・ミラー」 203 ページ
- 「フル・バックアップ」 206 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」 217 ページ
- 「同時性 (同時実行性)」 217 ページ
- 「整合性」 214 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」 212 ページ
- 「プライマリ・キー制約」 206 ページ
- 「制約」 214 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」 206 ページ
- 「外部テーブル」 212 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」 214 ページ
- 「検査制約」 213 ページ
- 「プライマリ・キー制約」 206 ページ
- 「一意性制約」 211 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」 198 ページ
- 「内部ジョイン」 217 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」 211 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」 214 ページ
- 「外部キー制約」 212 ページ
- 「プライマリ・キー制約」 206 ページ
- 「一意性制約」 211 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」 211 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」 206 ページ
- 「外部キー」 211 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」 206 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」 217 ページ
- 「コード・ページ」 197 ページ
- 「エンコード」 195 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」 205 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」 213 ページ
- 「外部キー制約」 212 ページ
- 「プライマリ・キー制約」 206 ページ
- 「一意性制約」 211 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」 213 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」 198 ページ
- 「ジョイン条件」 199 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」 197 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」 209 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」 216 ページ
- 「レプリケーション」 209 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」 189 ページ
- 「SQL Remote」 192 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」 203 ページ
- 「独立性レベル」 217 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」 199 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」 198 ページ
- 「外部ジョイン」 212 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」 203 ページ
- 「ロー・レベルのトリガ」 210 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」 197 ページ
- 「エンコード」 195 ページ
- 「照合」 214 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

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

あ

い

う

え

お

か

す

せ

そ

た

ち

な

に

ね

は

ひ

ふ

へ

ほ

ま

み

め

も

ゆ

よ

り

る

れ

ろ

わ

