SYBASE iAnywhere

Ultra Light J

2009 年 2 月

バージョン 11.0.1

版権と商標

目次

はじめに

このマニュアルの内容

このマニュアルでは、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。

対象読者

このマニュアルは、Java ベースのデータベース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。

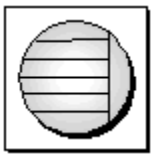
- **環境変数**　マニュアルでは、環境変数設定が引用されます。Windows のシェルでは、環境変数は構文 *%ENVVAR%* を使用して指定されます。UNIX のシェルでは、環境変数は構文 *$ENVVAR* または *${ENVVAR}* を使用して指定されます。

グラフィック・アイコン

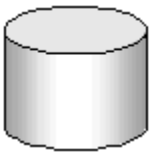
このマニュアルでは、次のアイコンを使用します。

- 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 J の使用

Ultra Light J の概要

目次

Ultra Light J 概要

Ultra Light J は、Java ME と SE の各プラットフォームと、BlackBerry スマートフォン向けに設計された Ultra Light の Java ベースのサブセットです。Ultra Light J を使用すると、トランザクション、プライマリ・キーと外部キー、インデックス、そしてリレーショナル・データベースのその他の機能を BlackBerry スマートフォンで使用できるようになります。Ultra Light J には変更トラッキング機能と同期機能が組み込まれており、データの同期を BlackBerry アプリケーションに組み込むことができます。Ultra Light J を Mobile Link サーバとともに使用すると、Oracle、SQL Server、DB2、Sybase ASE、SQL Anywhere の各データベースをモバイル・デバイスに拡張できます。

Ultra Light J には、データのテーブルへの格納、プライマリ・キーの使用、トランザクション・データベース・ストアなど、リレーショナル・データベースに一般的な多くの特徴があります。

Ultra Light J を再配布するには、Java Archive (JAR) ファイルをアプリケーションに追加します。Ultra Light J は、Java ME、Java SE 1.5 以降、OS 4.1 を実行している BlackBerry スマートフォンでサポートされています。

Ultra Light J の機能

データベース・ストア

Ultra Light J では、データベースをメモリ (非永続的) または記憶装置 (永続的) に格納することができます。Java SE の場合、ファイル・システムは記憶装置です。BlackBerry スマートフォンの場合は、BlackBerry のオブジェクト・ストアです。

トランザクション

トランザクションは、コミットまたはロールバックの間の一連のオペレーションです。永続的なデータベース・ストアの場合、コミット操作は、最後のコミットまたはロールバックの実行後に行われたすべての変更を永続的にします。ロールバック操作は、最後にコミットが呼び出されたときの状態にデータベースを戻します。

Ultra Light J のトランザクションとロー・レベルのオペレーションはそれぞれアトミックです。複数のカラムを対象とした挿入操作では、すべてのカラムにデータが挿入されるか、どのカラムにもデータが挿入されないかのいずれかです。

チェックポイントとリカバリ

Ultra Light J は、自動チェックポイントと手動チェックポイントの両方の機能を提供します。自動に設定されている場合、コミット文を実行したときにテーブルのローとインデックスが更新されます。自動ではなかった場合、Connection インタフェースの checkpoint メソッドを使用してチェックポイントが呼び出されます。

同時実行性とロッキング

Ultra Light J では、最大レベルの同時実行性を実現するために独立性レベル 0 (コミットされない読み出し) が使用されます。

- **ロッキング**　2 つの異なる接続が、同時に同じローを変更することはできません。2 つの接続が同じローを操作しようとすると、一方の接続が終了するまでもう一方の接続はブロックされます。
- **可視性**　データベース上で 1 つの接続が操作を行うと、その他の接続から即座に参照可能になります。

キャッシュ管理

永続ストアは、ページ・ベースです。Ultra Light J は、キャッシュ内のページに対して操作します。ページのワーキング・セットは、キャッシュに保持され、先入れ先出し (FIFO) 方式で管理されます。現在使用中のページは、スワップ・アウトされないようにキャッシュ内でロックされます。

Ultra Light J のキャッシュはサイズを設定できます。

Ultra Light J では、インデックスやロー・ページの遅延ロードを使用して、永続的なデータベースの起動を高速化できます。インデックスとロー・ページは、アプリケーションから初めてアクセスされるときにロードされます。

暗号化

暗号化は Configuration オブジェクトの setEncryption メソッドを使用して設定します。このメソッドは、EncryptionControl を引数に取ってページの暗号化と復号化を行います。独自の暗号化制御を指定する必要があります。

組み込み変更トラッキング

Mobile Link 同期クライアントとして、Ultra Light J には、データベースの変更を同期できるようにトランザクション・ログ・ベースの変更トラッキング・システムが組み込まれています。

HTTP 通信と HTTPS 通信

データの同期は、HTTP ネットワーク・プロトコルまたは HTTPS ネットワーク・プロトコルを使用して実行できます。HTTPS 同期を使用すると、Mobile Link サーバまで安全に暗号化されます。

同期パブリケーション

ネットワーク・リソースを有効に活用するために、Ultra Light J は、データベースから選択したテーブルを同期可能にするパブリケーション・モデルを提供しています。

文字セットと照合

Ultra Light J は Unicode を使用します (データベースでは UTF-8 にコード化されます)。照合は Java のデフォルトのソート順であり、SQL Anywhere がサポートする UTF8BIN 照合と同等です。Ultra Light J は、Mobile Link サーバとの同期中に、UTF8 文字セットと照合を使用することを Mobile Link に通知します。

Ultra Light J の制限事項

全般的な制限事項

Ultra Light J のデータベースに適用される全般的な制限事項を次に示します。

機能	制限
blob サイズ	最大 2^{24} バイト。
BlackBerry SD カード	サポートされていません。
データ型	「Domain インタフェース」 159 ページを参照してください。
データベース・アクセス	1 つのデータベースは、一度に 1 アプリケーションだけがアクセスできます。同時アクセスはサポートされていません。
データベース・ページ・サイズ	最小 256 バイト、最大 32 KB。
ロー・サイズ	ローの内容 (圧縮後) はデータベースのページ・サイズ以下である必要があります。
データベースあたりのテーブル数	最大 32 KB。

Ultra Light J のデータベース・ストア

サポートされているデータベース・ストア

Ultra Light J でサポートされているデータベース・ストアのリストを次に示します。

データベース・ストアのタイプ	プラットフォームのサポート
ファイル・システム・ストア	Java SE
RIM オブジェクト・ストア (RIM11)	Java ME
レコード・ストア	Java ME
非永続ストア	すべての Java プラットフォーム

BlackBerry オブジェクト・ストアの制限事項

BlackBerry スマートフォン上でのデータベース・ストアのサイズは、利用可能なオブジェクト・ハンドル数によって制限されます。利用可能なオブジェクト・ハンドル数は、フラッシュ・メモリのサイズで決まります。

フラッシュ・メモリ	永続的ハンドル数	ハンドル数
8 MB	12000	24000
16 MB	27000	56000
32 MB	65000	132000

Ultra Light J では、データベースに値を格納するためにオブジェクト・ハンドルが必要です。たとえば、10 個のカラムと 2 つのインデックスがあるテーブルのローには、最低 12 個のオブジェクト・ハンドルが必要です。

より大きなデータベース・ストアを許可するには、データベース・ページごとに永続的なオブジェクト・ハンドルが必要です。

永続ストアの設定とリカバリ

データベースを作成するときに、Configuration オブジェクトを使用して、アプリケーションに次のいずれかの永続性の形式を選択します。

- **非永続的**　NonPersist オブジェクトを作成すると、メモリ内だけに存在するデータベース・ストアが構成されます。データベースは起動時に作成され、アプリケーションの実行中に使用され、アプリケーションの終了時に破棄されます。アプリケーションの終了時には、非永続ストアに含まれるデータがすべて削除されます。
- **終了時書き込み型の永続性**　永続的設定オブジェクトの setWriteAtEnd メソッドを使用して終了時書き込み型の永続性を有効にすると、接続の解放時にのみデータがデータベースに書き

込まれます。この形式の永続性では、トランザクションが全体的に高速になりますが、アプリケーションが異常終了した場合はデータが失われる可能性があります。

- **シャドー・ページング型の永続性**　シャドー・ページング型は、最も永続性が強力な形式です。永続的設定オブジェクトの setShadowPaging メソッドを使用して有効にできます。起動時に有効にすると、アプリケーションが予期せず終了した場合でも、最後のチェックポイント時の状態にデータベースを回復できます。
- **シンプル・ページング型の永続性**　シャドー・ページングが有効ではないときは、永続性のシンプル・ページング実装が使用されます。すべてのデータが、データが読み込まれたページと同じページに書き込まれます。更新の開始時から更新が完了するまでの間はデータベースは破損しているとみなされます。破損したデータベースは、起動時に検出できますが、回復させることはできません。シンプル・ページングは、シャドー・ページングと比べてメモリ使用量が大幅に少なく、パフォーマンスが高くなります。

データの同期

Ultra Light J は、Mobile Link 11 とデータを同期することができます。Mobile Link と同期する場合は、-x Mobile Link サーバ・オプションを使用します。

Ultra Light J は、次の機能をサポートしています。

- Mobile Link ユーザ認証
- Mobile Link ユーザ認証スクリプト
- パブリケーション・ベースの同期
- HTTP と HTTPS の各ネットワーク・プロトコル
- アップロード専用、ダウンロード専用、ping 専用、フル・アップロード／ダウンロードの各モード
- 同期 observer API

BlackBerry 環境では、データはデバイスと BlackBerry Enterprise Server (BES) 間で常に暗号化されます。HTTPS は、BES と Mobile Link サーバ間で暗号化が必要な場合に使用されます。

同時同期処理

通常、Ultra Light J ランタイムでは、一度に 1 つのスレッドのみ許可されています。しかし、同期中はこの規則の限りではなく、1 つの接続が同期操作を行う間、その他の接続は Ultra Light J ランタイムにアクセスできます。ただし、次のような制限があります。

- 同期中にデータベースの Connection オブジェクトに対してスキーマ操作 (disableSynchronization、dropDatabase、dropTable、dropPublication、enableSynchronization、renameTable、または schemaCreateBegin) を実行できない。
- 同期操作の実行中、他のどのスレッドも同期されているデータベースに対して synchronize メソッドを呼び出すことはできない。

接続は、現在実行中の同期が失敗した場合に消去される可能性のあるダウンロードされたローに、同期中 (ローがコミットされる前) はアクセスできます。同期によって後から変更されるローを、同期中にある接続が修正してしまうと、同期は失敗します。同期によって変更されたローを、同期中にある接続が修正しようとすると、修正の試行は失敗します。

Ultra Light J アプリケーションの開発

目次

この項では、Ultra Light J アプリケーション・プログラミング・インタフェース (API) の概要について説明します。

Ultra Light J 開発の概要

Ultra Light J は、ユーザの Java アプリケーションに基本的なデータベース機能を追加します。特に BlackBerry スマートフォンと連携して動作するように設計されていますが、J2ME 環境および J2SE 環境とも完全な互換性があります。Ultra Light J の API には、Ultra Light J データベースに接続し、スキーマ操作を実行し、SQL 文を使用してデータを管理するために必要なすべてのメソッドが含まれます。データの暗号化や同期などの高度な操作もサポートされています。

サポートされている各プラットフォーム用の API が、Ultra Light J ディレクトリの *UltraLite.jar* ファイルに保存されています。通常、Ultra Light J ディレクトリは SQL Anywhere のインストール・ディレクトリ内の *UltraLite¥UltraLiteJ* に対応します。

Ultra Light J アプリケーションを作成する基本的な手順

Ultra Light J アプリケーションを作成するときは、一般に次の手順に従います。

1. 新しい Configuration オブジェクトを作成します。

   Configuration オブジェクトは、Ultra Light J データベースの保存場所または作成先を定義します。また、データベースへの接続に必要なユーザ名とパスワードを指定します。異なるデバイスや非永続データベース・ストア用にさまざまな Configuration オブジェクトがあります。「Configuration インタフェース」 123 ページを参照してください。

2. 新しい Connection オブジェクトを作成します。

   Connection オブジェクトは、Configuration オブジェクトで定義されている指定内容に従って Ultra Light J データベースに接続します。データベースが存在しない場合は自動的に作成されます。「Connection インタフェース」 125 ページを参照してください。

3. TableSchema、IndexSchema、ForeignKeySchema の各オブジェクトを適用します。

   Connection オブジェクトが提供するスキーマ・メソッドを使用して、テーブル、カラム、インデックス、外部キーを作成できます。「schemaCreateBegin メソッド」 143 ページを参照してください。

4. PreparedStatement オブジェクトを生成します。

   PreparedStatement オブジェクトは、Connection オブジェクトに関連付けられているデータベースに問い合わせます。引数には、サポートされている SQL 文を文字列として指定します。PreparedStatement オブジェクトを使用して、データベースの内容を更新できます。「PreparedStatement インタフェース」 180 ページを参照してください。

5. ResultSet オブジェクトを生成します。

   ResultSet オブジェクトは、SQL SELECT 文を含む PreparedStatement が Connection オブジェクトによって実行されたときに作成されます。ResultSet オブジェクトを使用して、データベースのテーブルの内容を確認できます。「ResultSet インタフェース」 184 ページを参照してください。

Ultra Light J アプリケーションの設定

任意の Java IDE で Ultra Light J アプリケーションを設定するときは、Ultra Light J ディレクトリにある *UltraLite.jar* リソース・ファイルを使用するようにプロジェクトが正しく設定されていることを確認します。

次の文を使用して、Ultra Light J パッケージを Java ファイルにインポートします。

```
import ianywhere.ultralitej.*;
```

このマニュアル内のすべてのサンプル・コードとチュートリアルでは、上記の文が指定されていること、使用している IDE での Java アプリケーション開発に精通していることを前提としています。

Ultra Light J データベース・ストアへのアクセス

アプリケーションでデータを操作するには、先に Ultra Light J データベースに接続する必要があります。この項では、パスワードを指定してデータベースを作成するか、データベースに接続する方法について説明します。

Configuration オブジェクトの実装

Configuration を使用してデータベースを作成し、データベースに接続します。API には複数の異なる Configuration の実装が用意されています。Ultra Light J でサポートされているデータベース・ストアのタイプごとに固有の実装があります。各実装には、データベース・ストアへのアクセスに使用する異なるメソッドのセットがあります。

- **RIM オブジェクト・ストア**　ConfigObjectStore でサポートされます。
- **レコード・ストア**　ConfigRecordStore でサポートされます。
- **ファイル・システム・ストア**　ConfigFile でサポートされます。
- **非永続ストア**　ConfigNonPersistent でサポートされます。

Connection オブジェクトのプロパティ

- **トランザクション**　トランザクションは、Connection の commit メソッドを使用してデータベースにコミットされる必要があります。トランザクションは rollback メソッドを使用してロールバックできます。
- **SQL 準備文**　SQL 文を処理するメソッドが PreparedStatement インタフェースに用意されています。PreparedStatement は、Connection の prepareStatement メソッドを使用して作成できます。
- **同期**　Mobile Link 同期を管理するオブジェクトのセットは、Connection からアクセスできます。
- **テーブル操作**　Ultra Light J データベース・テーブルには、Connection インタフェースに用意されているメソッドを使用してアクセスと管理を行います。

新しい Ultra Light J データベースの作成

Ultra Light J データベースを作成するには API を使用する必要があります。Sybase Central または Ultra Light のコマンド・ライン・ユーティリティを使用して新しいデータベースを作成することはできません。

♦ **データベースを作成するには、次の手順に従います。**

1. データベース名を参照する新しい Configuration を作成します。

   正しい構文は Java プラットフォームとクライアント・デバイスによって異なります。次の例では、config が Configuration オブジェクトの名前で、DBname.ulj が新しいデータベースの名前です。

   J2ME BlackBerry デバイスの場合：


```
ConfigObjectStore config =
   DatabaseManager.createConfigurationObjectStore("DBname.ulj");
```

その他の J2ME デバイスの場合：

```
ConfigRecordStore config =
   DatabaseManager.createConfigurationRecordStore("DBname.ulj");
```

J2SE デバイスの場合：

```
ConfigFile config =
   DatabaseManager.createConfigurationFile("DBname.ulj");
```

また、次の構文で、すべてのプラットフォームでサポートされる非永続データベースの Configuration を作成することもできます。

```
ConfigNonPersistent config =
   DatabaseManager.createConfigurationNonPersistent("DBname.ulj");
```

2. setPassword メソッドを使用して新しいデータベース・パスワードを設定します。

   ```
   config.setPassword("my_password");
   ```

3. 新しい Connection を作成します。

   ```
   Connection conn = DatabaseManager.createDatabase(config);
   ```

   createDatabase メソッドはデータベース作成処理を完了し、データベースに接続します。このメソッドの呼び出し後、スキーマとデータの操作を行うことはできますが、データベースの名前、パスワード、またはページ・サイズは変更できません。

既存のデータベースへの接続

Ultra Light J データベースに接続するには、データベースがクライアント・デバイスに存在している必要があります。

♦ **既存のデータベースに接続するには、次の手順に従います。**

1. データベース名を参照する新しい Configuration を作成します。

   正しい構文は Java プラットフォームとクライアント・デバイスによって異なります。次の例では、config が Configuration オブジェクトの名前で、DBname.ulj がデータベースの名前です。

   J2ME BlackBerry デバイスの場合：

   ```
   ConfigObjectStore config =
      DatabaseManager.createConfigurationObjectStore("DBname.ulj");
   ```

   その他の J2ME デバイスの場合：

   ```
   ConfigRecordStore config =
      DatabaseManager.createConfigurationRecordStore("DBname.ulj");
   ```

   J2SE デバイスの場合：

   ```
   ConfigFile config =
      DatabaseManager.createConfigurationFile("DBname.ulj");
   ```

また、次の構文で、すべてのプラットフォームでサポートされる非永続データベースの Configuration に接続することもできます。

```
ConfigNonPersistent config =
   DatabaseManager.createConfigurationNonPersistent("DBname.ulj");
```

2. setPassword メソッドを使用してデータベースのパスワードを指定します。

   ```
   config.setPassword("my_password");
   ```

3. 新しい Connection を作成します。

   ```
   Connection conn = DatabaseManager.connect(config);
   ```

   connect メソッドはデータベース接続処理を完了します。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。

データベースとの接続の切断

DatabaseManager クラスの release メソッドを使用して、Ultra Light J データベースとの接続を切断します。release メソッドは Connection と、関連付けられているすべてのプロパティを閉じます。

参照

- 「サンプル：データベースの作成」 34 ページ
- 「DatabaseManager クラス」 150 ページ

スキーマ操作の実行

Ultra Light J には、テーブル、カラム、インデックス、キーをデータベースに作成できるスキーマ・メソッドがあります。この項では、スキーマ操作を行う方法について説明します。

スキーマ・オブジェクトのタイプ

- **テーブル・スキーマ**　Connection インタフェースのメソッドを使用してテーブルのプロパティにアクセスします。スキーマには TableSchema インタフェースを使用して直接アクセスします。「TableSchema インタフェース」 245 ページを参照してください。
- **カラム・スキーマ**　TableSchema インタフェースのメソッドを使用してカラムのプロパティにアクセスします。スキーマには ColumnSchema インタフェースを使用して直接アクセスします。「ColumnSchema インタフェース」 106 ページを参照してください。
- **インデックス・スキーマ**　TableSchema インタフェースのメソッドを使用してインデックスにアクセスします。スキーマには IndexSchema インタフェースを使用して直接アクセスします。「IndexSchema インタフェース」 177 ページを参照してください。
- **外部キー・スキーマ**　Connection インタフェースのメソッドを使用して外部キーにアクセスします。スキーマには ForeignKeySchema インタフェースを使用して直接アクセスします。「ForeignKeySchema インタフェース」 175 ページを参照してください。

新しいテーブル、カラム、インデックス、キーの作成

テーブル、カラム、インデックス、キーのすべての操作は、API を使用して実行する必要があり、接続しているデータベースがスキーマ作成モードになっている場合に限られます。

♦ **スキーマ操作を実行するには、次の手順に従います。**

1. Ultra Light J データベースに接続します。

   この例は、データベースが Connection オブジェクトである conn に接続していることを前提としています。Ultra Light J データベースへの接続方法については、「Ultra Light J データベース・ストアへのアクセス」 14 ページを参照してください。

2. 次のコードを使用して、Connection をスキーマ作成モードにします。

   ```
   conn.schemaCreateBegin();
   ```

   スキーマ作成モードでは、データ操作の実行が禁止され、データベースへの追加接続がロック・アウトされます。

3. テーブル、カラム、インデックス、キーの操作をすべて実行します。

   次の手順は、emp_number という整数カラムを含む新しい Employee テーブルを作成する方法を示しています。emp_number カラムは Employee テーブルのプライマリ・インデックスであり、また Security テーブルの 整数カラムである access_number を参照する外部キーとして機能します。

   - **新しいテーブルを作成するには：**　createTable メソッドを使用してテーブルの名前を定義し、結果を TableSchema に割り当てます。

     ```
     TableSchema table_schema = conn.createTable("Employee");
     ```

結果を TableSchema に割り当てると、テーブルに対してより詳細なスキーマ操作を実行できます。詳細については、「TableSchema インタフェース」 245 ページを参照してください。

- **テーブルにカラムを追加するには：** createColumn メソッドを使用して、カラムの名前と型を定義します。

```
table_schema.createColumn("emp_number", Domain.INTEGER);
```

- **カラムにプライマリ・インデックスを割り当てるには：**

  a. createPrimaryIndex メソッドを使用してテーブルに新しいプライマリ・インデックスを作成し、結果を IndexSchema に割り当てます。

```
IndexSchema index_schema =
   table_schema.createPrimaryIndex("prime_keys");
```

  b. addColumn メソッドを使用して、プライマリ・インデックス・カラムとソート順を指定します。

```
index_schema.addColumn("emp_number", IndexSchema.ASCENDING);
```

- **カラムに外部キーを割り当てるには：** この手順は、access_number という整数カラムがある Security というテーブルがデータベース内にすでに存在することを前提としています。このテーブルが存在しない場合は、前述の手順で作成してください。

  a. createForeignKey メソッドを使用して、外部キーの作成に必要なテーブルを指定します。

```
ForeignKeySchema foreign_key_schema = conn.createForeignKey(
   "Employee",
   "Security",
   "fk_emp_to_sec"
);
```

  最初のパラメータは、外部キーを格納するテーブルを指定します。2 つ目のパラメータは、外部キーの参照先カラムを格納するテーブルを指定します。

  b. addColumnReference メソッドを使用して、外部キーの作成に必要な 2 つのカラムを指定します。

```
foreign_key_schema.addColumnReference("emp_number", "access_number");
```

  最初のパラメータは、最初のテーブルで外部キーとなるカラムの名前を指定します。2 つ目のパラメータは、2 つ目のテーブルで外部キーの参照先となるカラムの名前を指定します。

4. Connection のスキーマ作成モードを終了します。

```
conn.schemaCreateComplete();
```


参照

- 「Connection インタフェース」 125 ページ
- 「ColumnSchema インタフェース」 106 ページ
- 「ForeignKeySchema インタフェース」 175 ページ
- 「IndexSchema インタフェース」 177 ページ
- 「TableSchema インタフェース」 245 ページ

SQL を使用したデータへのアクセスと操作

サポートされている SQL 文

Ultra Light でサポートされている SQL 文の一部は、Ultra Light J ではサポートされていません。Ultra Light J でサポートされているすべての SQL 文のリストを次に示します。

SQL 文	注意事項と制限事項
ALTER TABLE	「Ultra Light ALTER TABLE 文」 『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。MAX HASH SIZE はサポートされません。
COMMIT	「Ultra Light COMMIT 文」 『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。
CREATE INDEX	「Ultra Light CREATE INDEX 文」 『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。MAX HASH SIZE はサポートされません。
CREATE TABLE	「Ultra Light CREATE TABLE 文」 『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。MAX HASH SIZE はサポートされません。
DELETE	「Ultra Light DELETE 文」 『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。
DROP INDEX	「Ultra Light DROP INDEX 文」 『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。
DROP TABLE	「Ultra Light DROP TABLE 文」 『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。
INSERT	「Ultra Light INSERT 文」 『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。
ROLLBACK	「Ultra Light ROLLBACK 文」 『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。
SELECT	SELECT 文の詳細については、「Ultra Light SELECT 文」 『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。 次の制限事項があります。 ● SQLCODE 関数はサポートされていません。 ● ACOS、ASIN、ATAN、ATAN2、POWER の各算術関数はサポートされていません。

SQL 文	注意事項と制限事項
START SYNCHRONIZATION DELETE	「Ultra Light START SYNCHRONIZATION DELETE 文」『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。
STOP SYNCHRONIZATION DELETE	「Ultra Light STOP SYNCHRONIZATION DELETE 文」『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。
TRUNCATE TABLE	「Ultra Light TRUNCATE TABLE 文」『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。
UPDATE	SELECT 文の詳細については、「Ultra Light UPDATE 文」『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。 JOIN 句はサポートされていません。

INSERT、UPDATE、DELETE を使用したデータ操作

PreparedStatement の execute メソッドを使用して SQL データ操作を実行できます。PreparedStatement は、データベースに対してユーザ定義の SQL 文を実行します。

PreparedStatement に SQL 文を適用するときは、**?** 文字でクエリ・パラメータを指定します。INSERT 文、UPDATE 文、DELETE 文では、文での順序位置に従ってそれぞれの **?** パラメータが参照されます。たとえば、最初の **?** はパラメータ 1、2 番目の **?** はパラメータ 2、のようになります。

♦ テーブルにローを挿入するには、次の手順に従います。

1. 新しい SQL 文を String として準備します。

   ```
   String sql_string =
      "INSERT INTO Department(dept_no, name) VALUES( ?, ? )";
   ```

2. String を PreparedStatement に渡します。

   ```
   PreparedStatement inserter =
      conn.prepareStatement(sql_string);
   ```

3. set メソッドを使用して、入力値を PreparedStatement に渡します。

   この例では、パラメータ 1 として参照している dept_no に 101 を設定し、パラメータ 2 として参照している name に "Electronics" を設定しています。

   ```
   inserter.set(1, 101);
   inserter.set(2, "Electronics");
   ```

4. 文を実行します。

```
inserter.execute();
```

5. PreparedStatement を閉じてリソースを解放します。

```
inserter.close()
```

6. データベースへのすべての変更をコミットします。

```
conn.commit();
```

♦ **テーブル内のローを更新するには、次の手順に従います。**

1. 新しい SQL 文を String として準備します。

```
String sql_string =
   "UPDATE Department SET dept_no = ? WHERE dept_no = ?";
```

2. String を PreparedStatement に渡します。

```
PreparedStatement updater =
   conn.prepareStatement(sql_string);
```

3. set メソッドを使用して、入力値を PreparedStatement に渡します。

```
updater.set(1, 102);
updater.set(2, 101);
```

上の例は、次の SQL 文を宣言することと同等です。

```
UPDATE Department SET dept_no = 102 WHERE dept_no = 101
```

4. 文を実行します。

```
updater.execute();
```

5. PreparedStatement を閉じてリソースを解放します。

```
updater.close()
```

6. データベースへのすべての変更をコミットします。

```
conn.commit();
```

♦ **テーブル内のローを削除するには、次の手順に従います。**

1. 新しい SQL 文を String として準備します。

```
String sql_string =
   "DELETE FROM Department WHERE dept_no = ?";
```

2. String を PreparedStatement に渡します。

```
PreparedStatement deleter =
   conn.prepareStatement(sql_string);
```

3. set メソッドを使用して、入力値を PreparedStatement に渡します。

```
deleter.set(1, 102);
```

上の例は、次の SQL 文を宣言することと同等です。


```
DELETE FROM Department WHERE dept_no = 102
```

4. 文を実行します。

```
deleter.execute();
```

5. PreparedStatement を閉じてリソースを解放します。

```
deleter.close()
```

6. データベースへのすべての変更をコミットします。

```
conn.commit();
```

SELECT を使用したデータの検索

データベースに対してユーザ定義の SQL 文を実行する PreparedStatement の executeQuery メソッドを使用してデータを検索することができます。このメソッドはクエリ結果を ResultSet として返します。その後 ResultSet をトラバースして、問い合わせたデータをフェッチできます。

ResultSet オブジェクトのナビゲーション

ResultSet には、SQL SELECT 文のクエリ結果内をナビゲーションするための次のメソッドがあります。

- **next**　次のローに移動します。
- **previous**　前のローに移動します。

ResultSet を使用したデータの取り出し

♦ **データベースからデータを選択するには、次の手順に従います。**

1. 新しい SQL 文を String として準備します。

```
String sql_string =
   "SELECT * FROM Department ORDER BY dept_no";
```

2. String を PreparedStatement に渡します。

```
PreparedStatement select_statement =
   conn.prepareStatement(sql_string);
```

3. 文を実行し、クエリの結果を ResultSet に割り当てます。

```
ResultSet cursor =
   select_statement.executeQuery();
```

4. ResultSet をトラバースしてデータを取り出します。

```
// Get the next row stored in the ResultSet.
cursor.next();

// Store the data from the first column in the table.
int dept_no = cursor.getInt(1);

// Store the data from the second column in the table.
String dept_name = cursor.getString(2);
```

5. ResultSet を閉じてリソースを解放します。

   ```
   cursor.close();
   ```

6. PreparedStatement を閉じてリソースを解放します。

   ```
   select_statement.close()
   ```

COMMIT、ROLLBACK を使用したトランザクションの管理

Ultra Light J では、オートコミット・モードはサポートされていません。トランザクションは、Connection インタフェースでサポートされているメソッドを使用して明示的にコミットまたはロールバックする必要があります。

トランザクションをコミットするには commit メソッドを使用します。

トランザクションをロールバックするには rollback メソッドを使用します。

参照

- 「commit メソッド」 131 ページ
- 「rollback メソッド」 143 ページ

データの暗号化と難読化

Ultra Light J データベース内のデータはデフォルトで暗号化されません。データは API を使用して暗号化または難読化できます。暗号化ではデータを安全に表現できますが、難読化ではデータベースの内容を不用意に閲覧されないことを目的とした簡易的なセキュリティを実現します。

♦ **データベース内のデータを暗号化または難読化するには、次の手順に従います。**

1. EncryptionControl インタフェースを実装するクラスを作成します。

   次の例では、暗号化インタフェースを実装する新しいクラス Encryptor を作成しています。

   ```
   static class Encryptor
      implements EncryptionControl
   {
   ```

2. 新しいクラスで initialize、encrypt、decrypt の各メソッドを実装します。

   クラスは次のようになります。

   ```
   static class Encryptor
      implements EncryptionControl
   {
      /** Decrypt a page stored in the database.
       * @param page_no the number of the page being decrypted
       * @param src the encrypted source page which was read from the database
       * @param tgt the decrypted page (filled in by method)
       */
      public void decrypt( int page_no, byte[] src, byte[] tgt )
         throws ULjException
      {
         // Your decryption method goes here.
      }

      /** Encrypt a page stored in the database.
       * @param page_no the number of the page being encrypted
       * @param src the unencrypted source
       * @param tgt the encrypted target page which will be written to the database (filled in by method)
       */
      public void encrypt( int page_no, byte[] src, byte[] tgt )
         throws ULjException
      {
         // Your encryption method goes here.
      }

      /** Initialize the encryption control with a password.
       * @param password the password
       */
      public void initialize(String password)
         throws ULjException
      {
         // Your initialization method goes here.
      }
   }
   ```

   EncryptionControl メソッドの詳細については、「EncryptionControl インタフェース」 173 ページを参照してください。

3. 新しいクラスを暗号化制御に使用するようにデータベースを設定します。

暗号化制御は setEncryption メソッドで指定できます。次の例は、データベースの名前を参照する新しい Configuration オブジェクトである config が作成してあることを前提としています。

```
config.setEncryption(new Encryptor());
```

4. データベースに接続します。

   データベース内で追加または変更されるデータは暗号化されます。

注意
暗号化と難読化は非永続データベース・ストアには使用できません。

参照

- 「サンプル：データの難読化」 45 ページ
- 「サンプル：データの暗号化」 48 ページ
- 「EncryptionControl インタフェース」 173 ページ

Mobile Link との同期

Mobile Link クライアントとしての Ultra Light J の使用

データを同期するには、アプリケーションで次の手順を実行する必要があります。

1. 統合データベースに関する情報 (サーバ名、ポート番号)、同期するベータベース名、同期するテーブルの定義を含む syncParms オブジェクトをインスタンス化します。
2. syncParms オブジェクトを渡した接続オブジェクトから synchronize メソッドを呼び出して、同期を実行します。

同期するデータはテーブル・レベルで定義できます。テーブルの一部分を同期するように設定することはできません。

参照

- 「SyncParms クラス」 224 ページ
- 「SyncResult クラス」 239 ページ

例

この例は、Ultra Light J アプリケーションでデータを同期する方法を示しています。

SQL Anywhere 11 CustDB を統合データベースとして Mobile Link サーバを起動するには、*samples-dir¥UltraLiteJ* ディレクトリから *start_ml.bat* を実行します。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * Sync: sample program to demonstrate Database synchronization.
 *
 * Requires starting the MobiLink Server Sample using start_ml.bat
 */
public class Sync
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Demo1.ulj" );

            Connection conn = DatabaseManager.createDatabase( config );
            conn.schemaCreateBegin();

            TableSchema table_schema = conn.createTable( "ULCustomer" );
            table_schema.createColumn( "cust_id", Domain.INTEGER );
            table_schema.createColumn( "cust_name", Domain.VARCHAR, 30 );
            IndexSchema index_schema = table_schema.createPrimaryIndex( "prime_keys" );
            index_schema.addColumn( "cust_id", IndexSchema.ASCENDING );
```

```
            conn.schemaCreateComplete();

            //
            // Synchronization
            //

            // Version set for MobiLink 11.0.x
            SyncParms syncParms = conn.createSyncParms( SyncParms.HTTP_STREAM, "50", "custdb
11.0" );
            syncParms.getStreamParms().setPort( 9393 );
            conn.synchronize( syncParms );
            SyncResult result = syncParms.getSyncResult();
            Demo.display(
                  "*** Synchronized *** bytes sent=" + result.getSentByteCount()
                  + ", bytes received=" + result.getReceivedByteCount()
                  + ", rows received=" + result.getReceivedRowCount()
               );

            conn.release();

         } catch( ULjException exc ) {
            Demo.displayException( exc );
         }
      }
   }
```

Ultra Light J 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

Mobile Link サーバと同期するときは、アプリケーション内でネットワーク・プロトコルを設定する必要があります。各データベースはネットワーク・プロトコルを通じて同期されます。Ultra Light J では、HTTP と HTTPS の 2 つのネットワーク・プロトコルを使用できます。

設定したネットワーク・プロトコルでは、対応するプロトコル・オプションのセットから選択することによって、Ultra Light J アプリケーションが Mobile Link サーバを特定して通信できるようにします。ネットワーク・プロトコルのオプションは、アドレス情報 (ホストとポート) やプロトコル固有の情報などを提供します。使用しているストリームのタイプに使用できるオプションについては、「Mobile Link クライアント・ネットワーク・プロトコル・オプションの一覧」『Mobile Link - クライアント管理』を参照してください。

HTTP ネットワーク・プロトコルの設定

HTTP ネットワーク・プロトコルは、Ultra Light J API の StreamHTTPParms インタフェースを使用して設定します。インタフェースのメソッドを使用して、Mobile Link サーバで定義されているネットワーク・プロトコルのオプションを指定します。ネットワーク・オプションの完全なリストについては、「StreamHTTPParms インタフェース」 209 ページを参照してください。

HTTPS ネットワーク・プロトコルの設定

HTTPS ネットワーク・プロトコルは、Ultra Light J API の StreamHTTPSParms インタフェースを使用して設定します。インタフェースのメソッドを使用して、Mobile Link サーバで定義されているネットワーク・プロトコルのオプションを指定します。ネットワーク・オプションの完全なリストについては、「StreamHTTPSParms インタフェース」 214 ページを参照してください。

CustDB アプリケーションの同期

CustDB (顧客データベース) は、SQL Anywhere と同時にインストールされるサンプル・データベースです。CustDB データベースは、販売注文用の簡単なデータベースです。

アプリケーションの検索と配備

Ultra Light J のインストールには、CustDB データベースに関連するサンプルの BlackBerry アプリケーションが含まれています。このアプリケーションの名前は CustDB で、ソース・コードと関連ファイルは、*sample-dir¥ultralitej¥CustDB¥* ディレクトリにあります。CustDB ディレクトリには、Research In Motion (RIM) JDE を使用して開くことのできるプロジェクト・ファイルが含まれています。

CustDB アプリケーションは、BlackBerry ブラウザに次の URL を指定して BlackBerry に直接ダウンロードし、動作を確認できます。

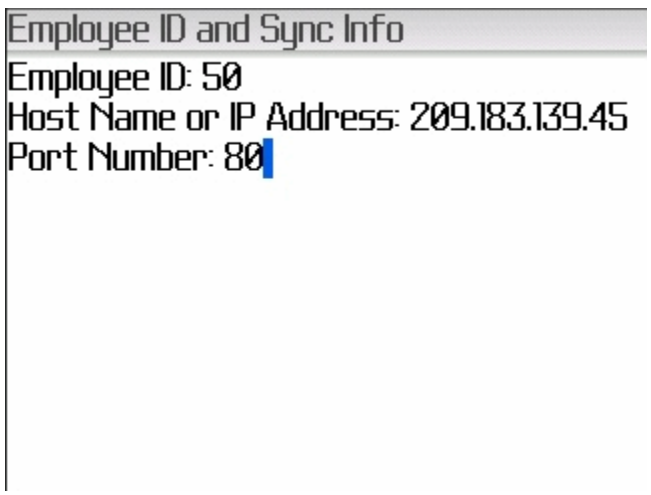
http://ultralitej.sybase.com/

CustDB アプリケーションの関連ファイル

- ***CustDB.java*** このファイルには、基本的なデータベース・アクセス・メソッドがすべて含まれています。これらのメソッドには、データベースの作成、データベースへの接続、注文の挿入、削除、更新などのメソッドがあります。このファイルには、バックエンド・サーバと通信するデータベース呼び出しの多くが含まれています。
- ***SchemaCreator.java*** このファイルには、Ultra Light J を使用してデバイス上にテーブルを作成するコードが含まれています。

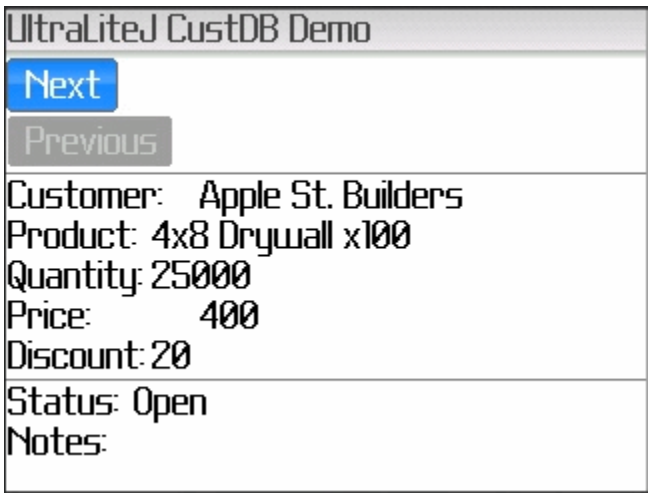
CustDB アプリケーションの使用

CustDB プログラムは、最初の起動時に、CustDB データベースをホストするサーバと対話するために必要な情報を収集します。ここで、クエリに使用する Employee ID (50 を推奨します)、データベースをホストするサーバのホスト名または IP アドレス、サーバとの接続に使用するポート番号を指定します。



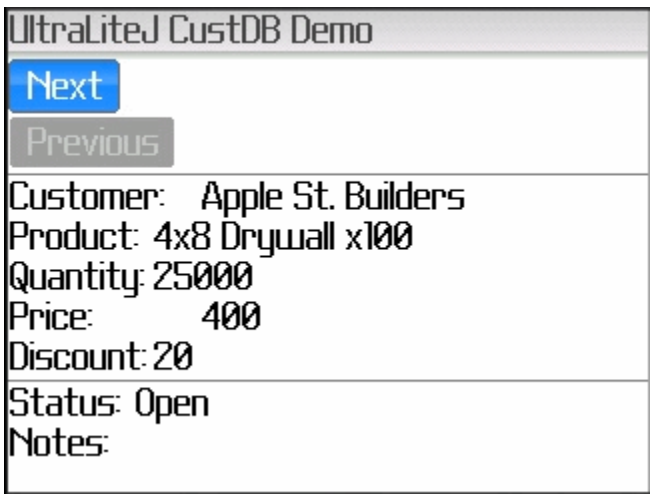
これらの値が指定されて設定が保存されると **([Menu] - [Save])**、アプリケーションは指定されたサーバと同期します。アプリケーションは、指定した従業員番号 (50) に対応する Employee ID と一致する注文だけをサーバからダウンロードします。オープン状態になっている注文のみが選択されます (注文のステータスは、Open、Approved、Denied のいずれかです)。

それぞれの注文が、顧客名 (Customer Name)、注文品 (Ordered Product)、注文数 (Ordered Quantity)、価格 (Price)、割引 (Discount) の情報とともに画面に表示されます。また、画面には注文の現在のステータス (Status) とその注文に関するメモ (Notes) も表示されます。



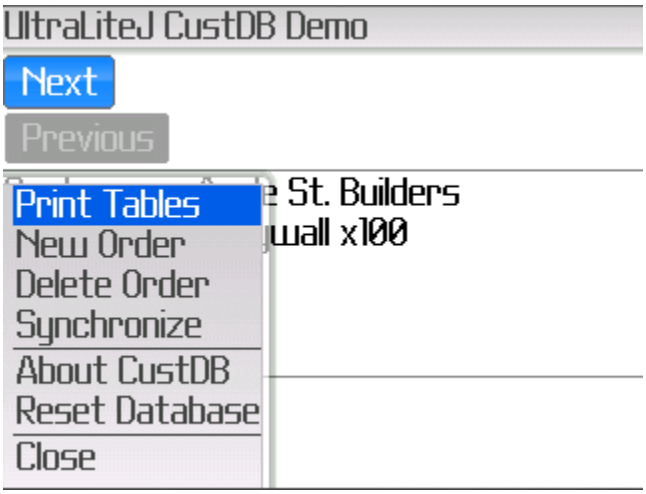
この画面では、注文にメモを書き加えたり、注文のステータスを Approved や Denied に変更したりすることができます。**[Next]** ボタンと **[Previous]** ボタンを使用して注文をナビゲーションできます。

CustDB プログラムでも、データベースに新しい注文を追加できます。新しい注文を追加するには、**[Menu] - [New Order]** をクリックします。



必要な注文数と割引情報を入力できます。

アプリケーションを終了する前に、メイン・メニューから **[Synchronize]** を選択して、変更内容と新しい注文をサーバと同期します。



Ultra Light J アプリケーションの配備

Ultra Light J アプリケーションが正常に実行されるためには、配布ファイルとともに Ultra Light J API も配備する必要があります。次の表に、Ultra Light J のさまざまな配備環境に必要なファイルのリストを示します。ファイルのパスはすべて SQL Anywhere のインストール・ディレクトリ内の *UltraLite¥UltraLiteJ* ディレクトリからの相対パスです。

配備のタイプ	必要なファイル
BlackBerry スマートフォン	*J2meRim11¥UltraLiteJ.cod* *J2meRim11¥UltraLiteJ.jad*[1]
J2ME	*J2me11¥UltraLiteJ.jar*
J2SE	*J2se¥UltraLiteJ.jar*

[1] 無線配信 (OTA：over-the-air) での配備にのみ必要です。また、アプリケーションとともに Ultra Light J を配備する独自の jad ファイルを作成することもできます。

サンプル・コード

この項では、Ultra Light J API を使用する Java コードのサンプルを示します。これらのサンプルでは、デバッグのためにメッセージを表示し、ULjException オブジェクトを処理するデモ・クラスを使用しています。

サンプル・コードはすべて *samples-dir/UltraLiteJ* ディレクトリにあります。ファイルの内容を変更する前に、元のソース・コードのバックアップ・コピーを作成してください。

サンプル：Demo クラス

このクラスは、マニュアルのこの項に含まれるすべてのサンプルで使用します。

```
// ************************************************************************
// Copyright 2006-2008 iAnywhere Solutions, Inc.  All rights reserved.
// ************************************************************************
package ianywhere.ultralitej.demo;

//import java.io.*;
import ianywhere.ultralitej.*;
//import ianywhere.ultralitej.implementation.*;

/**
 * Demonstration class.
 *
 * <p>This class is not part of the Database library.  It is used
 * only by the demonstration programs.
 *
 * @author ianywhere
 * @version 1.0
 */
public class Demo
{
    /** Display a message.
     * @param msg message to be displayed
     */
    public static void display( String msg )
    {
        System.out.println( msg );
    }

    /** Display a message.
     * @param msg1 message(1) to be displayed
     * @param msg2 message(2) to be displayed
     */
    public static void display( String msg1, String msg2 )
    {
        display( msg1 + msg2 );
    }

    /** Display a message.
     * @param msg1 message(1) to be displayed
     * @param msg2 message(2) to be displayed
     * @param msg3 message(3) to be displayed
     */
    public static void display( String msg1, String msg2, String msg3 )
    {
```

```
        display( msg1 + msg2 + msg3 );
    }

    /** Display a message.
     * @param msg1 message(1) to be displayed
     * @param msg2 message(2) to be displayed
     * @param msg3 message(3) to be displayed
     * @param msg4 message(4) to be displayed
     */
    public static void display( String msg1, String msg2, String msg3, String msg4 )
    {
        display( msg1 + msg2 + msg3 + msg4 );
    }

    /** Display message for an exception.
     * @param exc ULjException containing message
     */
    public static void displayException( ULjException exc )
    {
        display( exc.getMessage() );
    }

    /** Display message for an exception.
     * @param exc ULjException containing message
     */
    public static void displayExceptionFull( ULjException exc )
    {
        display( exc.getMessage() );
    }
}
```

サンプル：データベースの作成

このサンプルは、J2SE Java 環境でファイル・システムのデータベース・ストアを作成する方法を示しています。データベースの作成には、Configuration オブジェクトを使用します。作成すると、Connection オブジェクトが返されます。テーブルを作成するには、schemaUpdateBegin メソッドを呼び出して基本となるスキーマへの変更を開始し、schemaUpdateComplete メソッドでスキーマの変更を完了します。

注意：

- Ultra Light J のテーブルには所有者が存在せず、名前のみによって識別されます。
- Domain インタフェースは定数を定義し、Ultra Light J テーブルのカラムでサポートされているさまざまなデータ型を示します。
- プライマリ・インデックス (createPrimaryIndex) は、ユニークであることが保証されます。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * CreateDb: sample program to demonstrate Database creation.
 */
public class CreateDb
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
```



```
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Demo1.ulj" );

            Connection conn = DatabaseManager.createDatabase( config );
            conn.schemaCreateBegin();

            TableSchema table_schema = conn.createTable( "Employee" );
            table_schema.createColumn( "number", Domain.INTEGER );
            table_schema.createColumn( "last_name", Domain.VARCHAR, 32 );
            table_schema.createColumn( "first_name", Domain.VARCHAR, 32 );
            table_schema.createColumn( "age", Domain.INTEGER );
            table_schema.createColumn( "dept_no", Domain.INTEGER );
            IndexSchema index_schema = table_schema.createPrimaryIndex( "prime_keys" );
            index_schema.addColumn( "number", IndexSchema.ASCENDING );

            table_schema = conn.createTable( "Department" );
            table_schema.createColumn( "dept_no", Domain.INTEGER );
            table_schema.createColumn( "name", Domain.VARCHAR, 50 );
            index_schema = table_schema.createPrimaryIndex( "prime_keys" );
            index_schema.addColumn( "dept_no", IndexSchema.ASCENDING );

            ForeignKeySchema foreign_key_schema = conn.createForeignKey( "Employee", "Department",
"fk_emp_to_dept" );
            foreign_key_schema.addColumnReference( "dept_no", "dept_no" );

            conn.schemaCreateComplete();

            conn.release();

            Demo.display( "CreateDb completed successfully" );

        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

サンプル：ローの挿入

このサンプルは、Ultra Light J データベースにローを挿入する方法を示しています。

注意：

- 挿入されたデータは、Connection オブジェクトから commit メソッドが呼び出されたときに初めてデータベースで永続的になります。
- ローが挿入され、まだコミットされていないときは、他の接続からそのローを参照できます。このため、ある接続によって、実際にコミットされていないロー・データが取り出される可能性があります。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * LoadDb -- sample program to demonstrate loading a Database.
```

```
 */
public class LoadDb
{
    /**
     * Add a Department row.
     * @param conn connection to Database
     * @param dept_no department number
     * @param dept_name department name
     */
    private static void addDepartment( PreparedStatement inserter, int dept_no, String dept_name )
        throws ULjException
    {
        inserter.set( 1 /* "dept_no" */, dept_no );
        inserter.set( 2 /* "name" */, dept_name );
        inserter.execute();
    }

    /**
     * Add an Employee row.
     * @param conn connection to Database
     * @param emp_no employee number
     * @param last_name employee last name
     * @param first_name employee first name
     * @param age employee age
     * @param dept_no department number where employee works
     */
    private static void addEmployee( PreparedStatement inserter, int emp_no, String last_name
                                , String first_name, int age, int dept_no )
        throws ULjException
    {
        inserter.set( 1 /* "number" */, emp_no );
        inserter.set( 2 /* "last_name" */, last_name );
        inserter.set( 3 /* "first_name" */, first_name );
        inserter.set( 4 /* "age" */, age );
        inserter.set( 5 /* "dept_no" */, dept_no );
        inserter.execute();
    }

    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Demo1.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement inserter;

            inserter = conn.prepareStatement( "INSERT INTO Department( dept_no, name )
VALUES( ?, ? )" );
            addDepartment( inserter, 100, "Engineering" );
            addDepartment( inserter, 110, "Sales" );
            addDepartment( inserter, 103, "Marketing" );
            inserter.close();

            inserter = conn.prepareStatement(
                "INSERT INTO employee( ¥"number¥", last_name, first_name, age, dept_no )
VALUES( ?, ?, ?, ?, ? )"
            );
            addEmployee( inserter, 1000, "Welch", "James", 58, 100 );
```



```
            addEmployee( inserter, 1010, "Iverson", "Victoria", 23, 103 );
            inserter.close();

            conn.commit();
            conn.release();
            Demo.display( "LoadDb completed successfully" );
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

サンプル：テーブルの読み込み

このサンプルは、接続から PreparedStatement オブジェクトを取得し、PreparedStatement から ResultSet オブジェクトを取得しています。次のローが取得可能であると、ResultSet の next メソッドはそのたびに true を返します。true が返されると、ResultSet オブジェクトから現在のローのカラムの値を取得できます。

注意：

- ResultSet が作成されると、結果セットの最初のローの前に配置されます。コードで next メソッドを呼び出して、テーブル内の最初のローに移動する必要があります。
- Ultra Light J のテーブル名およびカラム名は、大文字と小文字が区別されません。カラムは、カラム名のみ ("age") またはテーブル名を伴ったカラム名 ("Employee.age") を指定することによって参照できます。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * ReadSeq -- sample program to demonstrate reading a Database table
 * sequentially.
 */
public class ReadSeq
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Demo1.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement stmt = conn.prepareStatement( "SELECT * FROM Employee ORDER BY
number" );
            ResultSet cursor = stmt.executeQuery();
            for( ; cursor.next(); ) {
                /* Can't access columns by name because no meta data */
                int emp_no = cursor.getInt( 1 /* "number" */ );
                String last_name = cursor.getString( 2 /* "last_name" */ );
                String first_name = cursor.getString( 3 /* "first_name" */ );
                int age = cursor.getInt( 4 /* "age" */ );
                Demo.display( first_name + ' ' + last_name );
```

```
                Demo.display( "  empl. no = "
                        , Integer.toString( emp_no )
                        , "  age = "
                        , Integer.toString( age ) );
            }
            cursor.close();
            stmt.close();
            conn.release();
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

サンプル：内部ジョイン操作

このサンプルは、内部ジョイン操作を実行する方法を示しています。このシナリオでは、各従業員に対応する部署情報があります。ジョイン操作によって、従業員 (Employee) テーブルのデータを部署 (Department) テーブルのそれに対応するデータと関連付けます。関連付けは従業員テーブルの部署番号を基準にして行われ、部署テーブル内の関連情報を特定します。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * ReadInnerJoin -- sample program to demonstrate reading the Employee table
 * and joining to each row the corresponding Department row.
 */
public class ReadInnerJoin
{
    /**
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Demo1.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement stmt = conn.prepareStatement(
                "SELECT E.number, E.last_name, E.first_name, E.age,"
                + " E.dept_no, D.name"
                + " FROM Employee E"
                + " JOIN Department D ON E.dept_no = D.dept_no"
                + " ORDER BY E.number"
            );
            ResultSet cursor = stmt.executeQuery();
            for( ; cursor.next(); ) {
                /* Can't access columns by name because no meta data */
                int emp_no = cursor.getInt( 1 /* "E.number" */ );
                String last_name = cursor.getString( 2 /* "E.last_name" */ );
                String first_name = cursor.getString( 3 /* "E.first_name" */ );
                int age = cursor.getInt( 4 /* "E.age" */ );
                int dept_no = cursor.getInt( 5 /* "E.dept_no" */ );
                String dept_name = cursor.getString( 6 /* "D.name" */ );
                System.out.println( first_name + ' ' + last_name );
                System.out.print( "  empl. no = " );
```



```
                System.out.print( emp_no );
                System.out.print( "  dept = " );
                System.out.println( dept_no );
                System.out.print( "  age = " );
                System.out.print( age );
                System.out.println( ", " + dept_name );
            }
            cursor.close();
            stmt.close();
            conn.release();
            Demo.display( "ReadInnerJoin completed successfully" );
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

サンプル：販売データベースの作成

このサンプルでは、販売指向型のデータベースを作成します。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * CreateDb: sample program to demonstrate creation of simple sales Database and
 * load it with some data.
 * <p>The program also illustrates the use of ordinals when inserting rows into tables.
 */
public class CreateSales
{
    static int ORDINAL_INVOICE_INV_NO;
    static int ORDINAL_INVOICE_NAME;
    static int ORDINAL_INVOICE_DATE;

    static int ORDINAL_INV_ITEM_INV_NO;
    static int ORDINAL_INV_ITEM_ITEM_NO;
    static int ORDINAL_INV_ITEM_PROD_NO;
    static int ORDINAL_INV_ITEM_QUANTITY;
    static int ORDINAL_INV_ITEM_PRICE;

    static int ORDINAL_PROD_NO;
    static int ORDINAL_PROD_NAME;
    static int ORDINAL_PROD_PRICE;

    /** Create the Database.
     * @return connection for a new Database
     */
    private static Connection createDatabase()
        throws ULjException
    {
        Configuration config = DatabaseManager.createConfigurationFile( "Sales.ulj" );
        Connection conn = DatabaseManager.createDatabase( config );

        conn.schemaCreateBegin();

        TableSchema table_schema = conn.createTable( "Product" );
        table_schema.createColumn( "prod_no", Domain.INTEGER );
        table_schema.createColumn( "prod_name", Domain.VARCHAR, 32 );
        table_schema.createColumn( "price", Domain.NUMERIC, 9, (short)2 );
        IndexSchema index_schema = table_schema.createPrimaryIndex( "prime_keys" );
```

```
        index_schema.addColumn( "prod_no", IndexSchema.ASCENDING );

        table_schema = conn.createTable( "Invoice" );
        table_schema.createColumn( "inv_no", Domain.INTEGER );
        table_schema.createColumn( "name", Domain.VARCHAR, 50 );
        table_schema.createColumn( "date", Domain.DATE );
        index_schema = table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "inv_no", IndexSchema.ASCENDING );

        table_schema = conn.createTable( "InvoiceItem" );
        table_schema.createColumn( "inv_no", Domain.INTEGER );
        table_schema.createColumn( "item_no", Domain.INTEGER );
        table_schema.createColumn( "prod_no", Domain.INTEGER );
        table_schema.createColumn( "quantity", Domain.INTEGER );
        table_schema.createColumn( "price", Domain.NUMERIC, 9, (short)2 );
        index_schema = table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "inv_no", IndexSchema.ASCENDING );
        index_schema.addColumn( "item_no", IndexSchema.ASCENDING );

        conn.schemaCreateComplete();

        return conn;
    }

    /** Populate the Database.
     * @param conn connection to Database
     */
    private static void populateDatabase( Connection conn )
        throws ULjException
    {
        PreparedStatement ri_product = conn.prepareStatement(
            "INSERT INTO Product( prod_no, prod_name, price ) VALUES( ?, ?, ? )"
            );
        ORDINAL_PROD_NO = 1;
        ORDINAL_PROD_NAME = 2;
        ORDINAL_PROD_PRICE = 3;
        addProduct( ri_product, 2001, "blue screw", ".03" );
        addProduct( ri_product, 2002, "red screw", ".09" );
        addProduct( ri_product, 2004, "hammer", "23.99" );
        addProduct( ri_product, 2005, "vice", "39.99" );
        ri_product.close();

        PreparedStatement ri_invoice = conn.prepareStatement(
            "INSERT INTO Invoice( inv_no, name, ¥"date¥" )"
            + " VALUES( :inv_no, :name, :inv_date )"
        );
        ORDINAL_INVOICE_INV_NO = ri_invoice.getOrdinal( "inv_no" );
        ORDINAL_INVOICE_NAME = ri_invoice.getOrdinal( "name" );
        ORDINAL_INVOICE_DATE = ri_invoice.getOrdinal( "inv_date" );

        PreparedStatement ri_item = conn.prepareStatement(
            "INSERT INTO InvoiceItem( inv_no, item_no, prod_no, quantity, price )"
            + " VALUES( ?, ?, ?, ?, ? )"
        );
        ORDINAL_INV_ITEM_INV_NO = 1;
        ORDINAL_INV_ITEM_ITEM_NO = 2;
        ORDINAL_INV_ITEM_PROD_NO = 3;
        ORDINAL_INV_ITEM_QUANTITY = 4;
        ORDINAL_INV_ITEM_PRICE = 5;

        addInvoice( ri_invoice, 2006001, "Jones Mfg.", "2006/12/23" );
        addInvoiceItem( ri_item, 2006001, 1, 2001, 3000, ".02" );
        addInvoiceItem( ri_item, 2006001, 2, 2002, 5000, ".08" );
```



```
    addInvoice( ri_invoice, 2006002, "Smith Inc.", "2006/12/24" );
    addInvoiceItem( ri_item, 2006002, 1, 2004, 2, "23.99" );
    addInvoiceItem( ri_item, 2006002, 2, 2005, 3, "39.99" );

    addInvoice( ri_invoice, 2006003, "Lee Ltd.", "2006/12/24" );
    addInvoiceItem( ri_item, 2006003, 1, 2004, 5, "23.99" );
    addInvoiceItem( ri_item, 2006003, 2, 2005, 4, "39.99" );
    addInvoiceItem( ri_item, 2006003, 3, 2001, 800, ".03" );
    addInvoiceItem( ri_item, 2006003, 4, 2002, 700, ".09" );

    ri_item.close();
    ri_invoice.close();

    conn.commit();
}

/**
 * mainline for program.
 *
 * @param args command-line arguments
 *
 */
public static void main
    ( String[] args )
{
    try {
        Connection conn = createDatabase();
        populateDatabase( conn );

        conn.release();

        Demo.display( "CreateSales completed successfully" );

    } catch( ULjException exc ) {
        Demo.displayExceptionFull( exc );
    }
}

/** Add an invoice row.
 * @param conn connection to Database
 * @param inv_no invoice number
 * @param name name to whom invoice was sent
 */
private static void addInvoice( PreparedStatement ri, int inv_no, String name
                        , String date )
    throws ULjException
{
    ri.set( ORDINAL_INVOICE_INV_NO, inv_no );
    ri.set( ORDINAL_INVOICE_NAME, name );
    ri.set( ORDINAL_INVOICE_DATE, date );
    ri.execute();
}

/** Add an invoice-item row.
 * @param conn connection to Database
 * @param inv_no invoice number
 * @param item_no line number for item
 * @param prod_no product number sold
 * @param quantity quantity sold
 * @param price price of one item
 */
private static void addInvoiceItem( PreparedStatement ri, int inv_no, int item_no
                            , int prod_no, int quantity, String price )
    throws ULjException
```

```
    {
        ri.set( ORDINAL_INV_ITEM_INV_NO, inv_no );
        ri.set( ORDINAL_INV_ITEM_ITEM_NO, item_no );
        ri.set( ORDINAL_INV_ITEM_PROD_NO, prod_no );
        ri.set( ORDINAL_INV_ITEM_QUANTITY, quantity );
        ri.set( ORDINAL_INV_ITEM_PRICE, price );
        ri.execute();
    }

    /** Add a product row.
     * @param conn connection to Database
     * @param prod_no product number
     * @param prod_name product name
     * @param price selling price
     */
    private static void addProduct( PreparedStatement ri, int prod_no, String prod_name, String price )
        throws ULjException
    {
        ri.set( ORDINAL_PROD_NO, prod_no );
        ri.set( ORDINAL_PROD_NAME, prod_name );
        ri.set( ORDINAL_PROD_PRICE, price );
        ri.execute();
    }
}
```

サンプル：集約とグループ化

このサンプルは、Ultra Light J でサポートされている、結果の集約を示しています。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/** Create a sales report to illustrate aggregation support.
 */
public class SalesReport
{
    /** Mainline.
     * @param args program arguments (not used)
     */
    public static void main( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Sales.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement stmt = conn.prepareStatement(
                "SELECT inv_no, SUM( quantity * price ) AS total"
                + " FROM InvoiceItem"
                + " GROUP BY inv_no ORDER BY inv_no"
            );
            ResultSet agg_cursor = stmt.executeQuery();
            for( ; agg_cursor.next(); ) {
                int inv_no = agg_cursor.getInt( 1 /* "inv_no" */ );
                String total = agg_cursor.getString( 2 /* "total" */ );
                Demo.display( Integer.toString( inv_no ) + ' ' + total );
            }
            Demo.display( "SalesReport completed successfully" );
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```


サンプル：別の順序でのローの取り出し

このサンプルは、Ultra Light J でサポートされている、別の順序でのローの処理を示しています。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;

public class SortTransactions
{
    /** Mainline.
     * @param args program arguments (not used)
     */
    public static void main( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Sales.ulj" );
            Connection conn = DatabaseManager.connect( config );
            PreparedStatement stmt = conn.prepareStatement(
                "SELECT inv_no, prod_no, quantity FROM InvoiceItem"
                + " ORDER BY prod_no"
            );
            ResultSet cursor = stmt.executeQuery();
            for( ; cursor.next(); ) {
                /* Can't access columns by name because no meta data */
                int inv_no = cursor.getInt( 1 /* "inv_no" */ );
                int prod_no = cursor.getInt( 2 /* "prod_no" */ );
                int quantity = cursor.getInt( 3 /* "quantity" */ );
                Demo.display( Integer.toString( prod_no ) + ' '
                    + Integer.toString( inv_no ) + ' '
                    + Integer.toString( quantity )
                );
            }
            conn.release();
            Demo.display( "SortTransactions completed successfully" );
        } catch( ULjException exc ) {
            Demo.displayException( exc );
        }
    }
}
```

サンプル：テーブル定義の変更

このサンプルは、テーブル定義の変更方法を示しています。このシナリオでは、カラム長を 50 文字から 100 文字に拡張するように Invoice テーブルを変更しています。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/** Reorganize the Invoice table to have a name with an increased size
 *
 * <p>This shows a possible strategy which can be used to reorganize
 * tables, since UltraLiteJ has no table-altering API.
 * <p>The (contrived) example expands the name column to 100 characters.
 *
 */
public class Reorg
{
    /**
```

```
     * mainline for program.
     *
     * @param args command-line arguments
     *
     */
    public static void main
        ( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Sales.ulj" );
            Connection conn = DatabaseManager.connect( config );
            createNewInvoiceTable( conn );
            copyInvoicesToNewTable( conn );
            deleteOldInvoicesTable( conn );
            renameNewInvoicesTable( conn );
            enableSynchronizationForNewTable( conn );
            conn.release();
        } catch( ULjException exc ) {
            Demo.displayExceptionFull( exc );
        }
    }

    private static void createNewInvoiceTable( Connection conn )
        throws ULjException
    {
        conn.schemaCreateBegin();
        TableSchema table_schema = conn.createTable( "NewInvoice" );
        table_schema.createColumn( "inv_no", Domain.INTEGER );
        table_schema.createColumn( "name", Domain.VARCHAR, 100 );   // was 50 in old table
        table_schema.createColumn( "date", Domain.DATE );
        IndexSchema index_schema = table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "inv_no", IndexSchema.ASCENDING );
        table_schema.setNoSync( true );         // we don't want to sync inserts yet
        conn.schemaCreateComplete();
    }

    private static void copyInvoicesToNewTable( Connection conn )
        throws ULjException
    {
        PreparedStatement inserter = conn.prepareStatement(
            "INSERT INTO NewInvoice( inv_no, name, ¥"date¥" ) VALUES( ?, ?, ? )"
        );
        int ordinal_inv_no = 1;
        int ordinal_inv_name = 2;
        int ordinal_inv_date = 3;

        PreparedStatement stmt = conn.prepareStatement(
            "SELECT inv_no, name, ¥"date¥" FROM Invoice"
        );
        ResultSet cursor = stmt.executeQuery();
        for( ; cursor.next(); ) {
            inserter.set( ordinal_inv_no, cursor.getInt( ordinal_inv_no ) );
            inserter.set( ordinal_inv_name, cursor.getString( ordinal_inv_name ) );
            inserter.set( ordinal_inv_date, cursor.getString( ordinal_inv_date ) );
            inserter.execute();
            // in memory-low conditions, we could delete the row from the old table (specify
            // stopSynchronizationDelete, so these deletes would not synchronize.
        }
        inserter.close();
        cursor.close();
        stmt.close();
        conn.commit();
    }
```



```
    private static void deleteOldInvoicesTable( Connection conn )
        throws ULjException
    {
        conn.dropTable( "Invoice" );
    }

    private static void renameNewInvoicesTable( Connection conn )
        throws ULjException
    {
        conn.renameTable( "NewInvoice", "Invoice" );
    }

    private static void enableSynchronizationForNewTable( Connection conn )
        throws ULjException
    {
        conn.enableSynchronization( "Invoice" );
    }
}
```

サンプル：データの難読化

このサンプルは、データベースのデータを難読化する方法を示しています。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
/**
 * Obfuscate -- sample program to a possible obfuscation of the database.
 *
 * Obfuscation is not very good encryption.  It merely makes the data unreadable
 * with a file dumping program.  The original data can be probably recovered by
 * someone with knowledge of the algorithms used.
 */
public class Obfuscate
{
    /** Create the database.
     * @return connection for a new database
     */
    private static Connection createDatabase()
        throws ULjException
    {
        ConfigPersistent config = DatabaseManager.createConfigurationFile( "Obfuscate.ulj" );
        config.setEncryption( new Obfuscator() );
        Connection conn = DatabaseManager.createDatabase( config );

        conn.schemaCreateBegin();

        TableSchema table_schema = conn.createTable( "Product" );
        table_schema.createColumn( "prod_no", Domain.INTEGER );
        table_schema.createColumn( "prod_name", Domain.VARCHAR, 32 );
        table_schema.createColumn( "price", Domain.NUMERIC, 9, (short)2 );
        IndexSchema index_schema = table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "prod_no", IndexSchema.ASCENDING );

        conn.schemaCreateComplete();

        return conn;
    }

    /** Add a product row.
     * @param ri PreparedStatement for the Product table
```

```
     * @param prod_no product number
     * @param prod_name product name
     * @param price selling price
     */
    private static void addProduct( PreparedStatement ri, int prod_no, String prod_name, String price )
        throws ULjException
    {
        ri.set( "prod_no", prod_no );
        ri.set( "prod_name", prod_name );
        ri.set( "price", price );
        ri.execute();
    }

    /** Populate the database.
     * @param conn connection to database
     */
    private static void populate( Connection conn )
        throws ULjException
    {
        PreparedStatement ri = conn.prepareStatement(
            "INSERT INTO Product( prod_no, prod_name, price )"
            + " VALUES( :prod_no, :prod_name, :price )"
        );
        addProduct( ri, 2001, "blue screw", ".03" );
        addProduct( ri, 2002, "red screw", ".09" );
        addProduct( ri, 2004, "hammer", "23.99" );
        addProduct( ri, 2005, "vise", "39.99" );
        ri.close();
        conn.commit();
    }

    /** Display contents of Product table.
     * @param conn connection to database
     */
    private static void displayProducts( Connection conn )
        throws ULjException
    {
        PreparedStatement stmt = conn.prepareStatement(
            "SELECT prod_no, prod_name, price FROM Product"
            + " ORDER BY prod_no"
        );
        ResultSet cursor = stmt.executeQuery();
        for( ; cursor.next(); ) {
            String prod_no = cursor.getString( 1 /* "prod_no" */ );
            String prod_name = cursor.getString( 2 /* "prod_name" */ );
            String price = cursor.getString( 3 /* "price" */ );
            Demo.display( prod_no + " " + prod_name + " " + price );
        }
        cursor.close();
        stmt.close();
    }

    /** mainline for program.
     * @param args command-line arguments (not used)
     */
    public static void main
        ( String[] args )
    {
        try {
            Connection conn = createDatabase();
            populate( conn );
            displayProducts( conn );
            conn.release();
        } catch( ULjException exc ) {
```



```
            Demo.displayException( exc );
        }
    }

    /** Class to implement encryption/decryption of the database.
     */
    static class Obfuscator
        implements EncryptionControl
    {
        /** seed for obfuscator         */      private int _seed;

        /** (un)Obfuscate a page.
         * @param page_no the number of the page being encrypted
         * @param src the encrypted source page which was read from the database
         * @param tgt the unencrypted page (filled in by method)
         */
        private void transform( int page_no, byte[] src, byte[] tgt )
        {
            int seed = ( _seed + page_no ) % 256;
            for( int i = 0; i < src.length; ++i ) {
                tgt[ i ] = (byte)( seed ^ src[ i ] );
                seed = ( seed + 93 ) % 256;
            }
        }

        /** Encrypt a page stored in the database.
         * @param page_no the number of the page being encrypted
         * @param src the encrypted source page which was read from the database
         * @param tgt the unencrypted page (filled in by method)
         */
        public void decrypt( int page_no, byte[] src, byte[] tgt )
            throws ULjException
        {
            transform( page_no, src, tgt );
        }

        /** Encrypt a page stored in the database.
         * @param page_no the number of the page being encrypted
         * @param src the unencrypted source
         * @param tgt the encrypted target page which will be written to the database (filled in by method)
         */
        public void encrypt( int page_no, byte[] src, byte[] tgt )
            throws ULjException
        {
            transform( page_no, src, tgt );
        }

        /** Initialize the encryption control with a password.
         * @param password the password
         */
        public void initialize( String password )
            throws ULjException
        {
            byte[] bytes = null;
            try {
                bytes = password.getBytes( "UTF8" );
            } catch( Exception e ) {
                Demo.display( "Encryption initialization failure" );
                throw new ObfuscationError();
            }
            _seed = 0;
            for( int i = bytes.length; i > 0; ) {
                _seed ^= bytes[ --i ];
            }
```

```
        }
    }

    /** Error class for encryption errors.
     */
    static class ObfuscationError
        extends ULjException
    {
        /** Constructor.
         */
        ObfuscationError()
        {
            super( "Obfuscation Error" );
        }

        /**
         * Get the error code, associated with this exception.
         * @return the error code (from the list at the top of this class) associated
         *  with this exception
         */
        public int getErrorCode()
        {
            return ULjException.SQLE_ERROR;
        }

        /** Get exception causing this exception, if it exists.
         * @return null, if there exists no causing exception; otherwise, the exception causing this exception
         */
        public ULjException getCausingException()
        {
            return null;
        }

        /** Get offset of error within a SQL string.
         * @return (-1) when there is no SQL string associated with the error message; otherwise,
         * the (base 0) offset within that string where the error occurred.
         */
        public int getSqlOffset()
        {
            return -1;
        }
    }
}
```

サンプル：データの暗号化

このサンプルは、データベースのデータを暗号化する方法を示しています。このシナリオでは、データの復号化が原因でパフォーマンスの低下が生じます。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
/**
 * Encrypted -- sample program to demonstrate encryption of database.
 *
 * This sample requires either the Sun JDK 1.4.2 (or later) or a freeware
 * version of the Java encryption classes (JCE).
```



```
 */
public class Encrypted
{
    /** Create the database.
     * @return connection for a new database
     */
    private static Connection createDatabase()
        throws ULjException
    {
        ConfigPersistent config = DatabaseManager.createConfigurationFile( "Encrypt.ulj" );
        config.setEncryption( new Encryptor() );
        Connection conn = DatabaseManager.createDatabase( config );

        conn.schemaCreateBegin();

        TableSchema table_schema = conn.createTable( "Product" );
        table_schema.createColumn( "prod_no", Domain.INTEGER );
        table_schema.createColumn( "prod_name", Domain.VARCHAR, 32 );
        table_schema.createColumn( "price", Domain.NUMERIC, 9, (short)2 );
        IndexSchema index_schema = table_schema.createPrimaryIndex( "prime_keys" );
        index_schema.addColumn( "prod_no", IndexSchema.ASCENDING );

        conn.schemaCreateComplete();

        return conn;
    }

    /** Add a product row.
     * @param ri PreparedStatement for the Product table
     * @param prod_no product number
     * @param prod_name product name
     * @param price selling price
     */
    private static void addProduct( PreparedStatement ri, int prod_no, String prod_name, String price )
        throws ULjException
    {
        ri.set( "prod_no", prod_no );
        ri.set( "prod_name", prod_name );
        ri.set( "price", price );
        ri.execute();
    }

    /** Populate the database.
     * @param conn connection to database
     */
    private static void populate( Connection conn )
        throws ULjException
    {
        PreparedStatement ri = conn.prepareStatement(
            "INSERT INTO Product( prod_no, prod_name, price )"
            + " VALUES( :prod_no, :prod_name, :price )"
        );
        addProduct( ri, 2001, "blue screw", ".03" );
        addProduct( ri, 2002, "red screw", ".09" );
        addProduct( ri, 2004, "hammer", "23.99" );
        addProduct( ri, 2005, "vise", "39.99" );
        ri.close();
        conn.commit();
    }

    /** Display contents of Product table.
     * @param conn connection to database
     */
    private static void displayProducts( Connection conn )
```

```
    throws ULjException
{
    PreparedStatement stmt = conn.prepareStatement(
        "SELECT prod_no, prod_name, price FROM Product"
        + " ORDER BY prod_no"
    );
    ResultSet cursor = stmt.executeQuery();
    for( ; cursor.next(); ) {
        /* Can't access columns by name because no meta data */
        String prod_no = cursor.getString( 1 /* "prod_no" */ );
        String prod_name = cursor.getString( 2 /* "prod_name" */ );
        String price = cursor.getString( 3 /* "price" */ );
        Demo.display( prod_no + " " + prod_name + " " + price );
    }
    cursor.close();
    stmt.close();
}

/** mainline for program.
 * @param args command-line arguments (not used)
 */
public static void main
    ( String[] args )
{
    try {
        Connection conn = createDatabase();
        populate( conn );
        displayProducts( conn );
        conn.release();
    } catch( ULjException exc ) {
        Demo.displayException( exc );
    }
}

/** Class to implement encryption/decryption of the database.
 */
static class Encryptor
    implements EncryptionControl
{
    private SecretKeySpec _key = null;
    private Cipher _cipher = null;

    /** Encrypt a page stored in the Database.
     * @param page_no the number of the page being encrypted
     * @param src the encrypted source page which was read from the Database
     * @param tgt the unencrypted page (filled in by method)
     */
    public void decrypt( int page_no, byte[] src, byte[] tgt )
        throws ULjException
    {
        byte[] decrypted = null;
        try {
            _cipher.init( Cipher.DECRYPT_MODE, _key );
            decrypted = _cipher.doFinal( src );
        } catch( Exception e ) {
            Demo.display( "Error: decrypting" );
            throw new EncryptionError();
        }
        for( int i = tgt.length; i > 0; ) {
            --i;
            tgt[ i ] = decrypted[ i ];
        }
    }
```



```
    /** Encrypt a page stored in the Database.
     * @param page_no the number of the page being encrypted
     * @param src the unencrypted source
     * @param tgt the encrypted target page which will be written to the Database (filled in by method)
     */
    public void encrypt( int page_no, byte[] src, byte[] tgt )
        throws ULjException
    {
        byte[] encrypted = null;
        try {
            _cipher.init( Cipher.ENCRYPT_MODE, _key );
            encrypted = _cipher.doFinal( src );
        } catch( Exception e ) {
            Demo.display( "Error: encrypting" );
            throw new EncryptionError();
        }
        for( int i = tgt.length; i > 0; ) {
            --i;
            tgt[ i ] = encrypted[ i ];
        }
    }

    /** Initialize the encryption control with a password.
     * @param password the password
     */
    public void initialize( String password )
        throws ULjException
    {
        try {
            byte[] bytes = password.getBytes( "UTF8" );
            MessageDigest md = MessageDigest.getInstance( "SHA" );
            bytes = md.digest( bytes );
            byte[] key_bytes = new byte[16];
            for( int i = key_bytes.length; i > 0; ) {
                --i;
                key_bytes[ i ] = bytes[ i ];
            }
            _key = new SecretKeySpec( key_bytes, "AES" );
            _cipher = Cipher.getInstance( "AES/ECB/NoPadding" );
        } catch( Exception e ) {
            Demo.display( "Error: initializing encryption" );
            throw new EncryptionError();
        }
    }
}

/** Error class for encryption errors.
 */
static class EncryptionError
    extends ULjException
{
    /** Constructor.
     */
    EncryptionError()
    {
        super( "Encryption Error" );
    }

    /**
     * Get the error code, associated with this exception.
     * @return the error code (from the list at the top of this class) associated
     *  with this exception
     */
    public int getErrorCode()
```

```
        {
            return ULjException.SQLE_ERROR;
        }

        /** Get exception causing this exception, if it exists.
         * @return null, if there exists no causing exception; otherwise, the exception causing this exception
         */
        public ULjException getCausingException()
        {
            return null;
        }

        /** Get offset of error within a SQL string.
         * @return (-1) when there is no SQL string associated with the error message; otherwise,
         * the (base 0) offset within that string where the error occurred.
         */
        public int getSqlOffset()
        {
            return -1;
        }
    }
}
```

サンプル：データベースのスキーマ情報の表示

このサンプルは、Ultra Light J データベースのシステム・テーブルをナビゲーションしてスキーマ情報を確認する方法を示しています。テーブルの各ローのデータも表示されます。

```
package ianywhere.ultralitej.demo;
import ianywhere.ultralitej.*;

/** Sample program to dump schema of a database.
 * This sample extracts schema information into a set of data structures
 * (TableArray, OptionArray) before dumping out the meta data so as to
 * provide for future schema information lookup.
 */
public class DumpSchema
{
    /** Mainline.
     * @param args program arguments (not used)
     */
    public static void main( String[] args )
    {
        try {
            Configuration config = DatabaseManager.createConfigurationFile( "Sales.ulj" );
            Connection conn = DatabaseManager.connect( config );

            Demo.display(
                    TableSchema.SYS_TABLES
                    + " table_flags are:¥nTableSchema.TABLE_IS_SYSTEM(0x"
                    + Integer.toHexString( ((int)TABLE_FLAG_SYSTEM) & 0xffff )
                    + "),¥nTableSchema.TABLE_IS_NOSYNC(0x"
                    + Integer.toHexString( ((int)TABLE_FLAG_NO_SYNC) & 0xffff )
                    + ")"
                );
            getSchema( conn );
            dumpSchema( conn );

        } catch( ULjException exc ) {
            Demo.displayException( exc );
```



```
    }
}

// Some constants for metadata
private static String SQL_SELECT_TABLE_COLS =
        "SELECT T.table_id, T.table_name, T.table_flags,"
        + " C.column_id, C.column_name, C.column_flags,"
        + " C.column_domain, C.column_length, C.column_default"
        + " FROM " + TableSchema.SYS_TABLES + " T"
        + " JOIN " + TableSchema.SYS_COLUMNS + " C"
        + " ON T.table_id = C.table_id"
        + " ORDER BY T.table_id"
        ;
private static final int TABLE_ID = 1;
private static final int TABLE_NAME = 2;
private static final int TABLE_FLAGS = 3;
private static final int COLUMN_ID = 4;
private static final int COLUMN_NAME = 5;
private static final int COLUMN_FLAGS = 6;
private static final int COLUMN_DOMAIN_TYPE = 7;
private static final int COLUMN_DOMAIN_LENGTH = 8;
private static final int COLUMN_DEFAULT = 9;

private static final int TABLE_FLAG_SYSTEM = TableSchema.TABLE_IS_SYSTEM;
private static final int TABLE_FLAG_NO_SYNC = TableSchema.TABLE_IS_NOSYNC;

private static final int COLUMN_FLAG_IN_PRIMARY_INDEX = 0x01;
private static final int COLUMN_FLAG_IS_NULLABLE = 0x02;

private static String SQL_SELECT_INDEX_COLS =
        "SELECT I.table_id, I.index_id, I.index_name, I.index_flags,"
        + " X.¥"order¥", X.column_id, X.index_column_flags"
        + " FROM " + TableSchema.SYS_INDEX_COLUMNS + " X"
        + " JOIN " + TableSchema.SYS_INDEXES + " I"
        + " ON I.table_id = X.table_id AND I.index_id = X.index_id"
        + " ORDER BY X.table_id, X.index_id, X.¥"order¥""
        ;
private static final int INDEX_TABLE_ID = 1;
private static final int INDEX_ID = 2;
private static final int INDEX_NAME = 3;
private static final int INDEX_FLAGS = 4;
private static final int INDEX_COLUMN_ORDER = 5;
private static final int INDEX_COLUMN_COLUMN_ID = 6;
private static final int INDEX_COLUMN_FLAGS = 7;

private static final int INDEX_COLUMN_FLAG_FORWARD = 1;

private static final int INDEX_FLAG_UNIQUE_KEY = 0x01;
private static final int INDEX_FLAG_UNIQUE_INDEX = 0x02;
private static final int INDEX_FLAG_PERSISTENT = 0x04;
private static final int INDEX_FLAG_PRIMARY_INDEX = 0x08;

private static String SQL_SELECT_OPTIONS =
        "SELECT name, value FROM " + TableSchema.SYS_INTERNAL
        + " ORDER BY name"
        ;
private static final int OPTION_NAME = 1;
private static final int OPTION_VALUE = 2;

// Metadata:
private static TableArray tables = new TableArray();
private static OptionArray options = new OptionArray();

/**
```

```
 * Extracts the schema of a database
 */
private static void getSchema( Connection conn ) throws ULjException
{
    PreparedStatement stmt = conn.prepareStatement(
            SQL_SELECT_TABLE_COLS
        );
    ResultSet cursor = stmt.executeQuery();
    Table table = null;
    int last_table_id = -1;
    for( ; cursor.next(); ) {
        int table_id = cursor.getInt( TABLE_ID );
        if( table_id != last_table_id ) {
            String table_name = cursor.getString( TABLE_NAME );
            int table_flags = cursor.getInt( TABLE_FLAGS );
            table = new Table( table_id, table_name, table_flags );
            tables.append( table );
            last_table_id = table_id;
        }
        int column_id = cursor.getInt( COLUMN_ID );
        String column_name = cursor.getString( COLUMN_NAME );
        int column_flags = cursor.getInt( COLUMN_FLAGS );
        int column_domain = cursor.getInt( COLUMN_DOMAIN_TYPE );
        int column_length = cursor.getInt( COLUMN_DOMAIN_LENGTH );
        int column_default = cursor.getInt( COLUMN_DEFAULT );
        Column column = new Column(
                conn, column_id, column_name, column_flags,
                column_domain, column_length, column_default
            );
        table.addColumn( column );
    }
    cursor.close();
    stmt.close();

    // read indexes
    stmt = conn.prepareStatement( SQL_SELECT_INDEX_COLS );
    cursor = stmt.executeQuery();
    int last_index_id = -1;
    Index index = null;
    last_table_id = -1;
    for( ; cursor.next(); ) {
        int table_id = cursor.getInt( INDEX_TABLE_ID );
        int index_id = cursor.getInt( INDEX_ID );
        if( last_table_id != table_id || last_index_id != index_id ) {
            String index_name = cursor.getString( INDEX_NAME );
            int index_flags = cursor.getInt( INDEX_FLAGS );
            index = new Index( index_id, index_name, index_flags );
            table = findTable( table_id );
            table.addIndex( index );
            last_index_id = index_id;
            last_table_id = table_id;
        }
        int order = cursor.getInt( INDEX_COLUMN_ORDER );
        int column_id = cursor.getInt( INDEX_COLUMN_COLUMN_ID );
        int index_column_flags = cursor.getInt( INDEX_COLUMN_FLAGS );
        IndexColumn index_column = new IndexColumn( order, column_id, index_column_flags );
        index.addColumn( index_column );
    }
    cursor.close();
    stmt.close();

    // read database options
    stmt = conn.prepareStatement( SQL_SELECT_OPTIONS );
    cursor = stmt.executeQuery();
```



```
    for( ; cursor.next(); ) {
        String option_name = cursor.getString( OPTION_NAME );
        String option_value = cursor.getString( OPTION_VALUE );
        Option option = new Option( option_name, option_value );
        options.append( option );
    }
    cursor.close();
    stmt.close();
}

/** Dump the schema of a database
 */
private static void dumpSchema( Connection conn ) throws ULjException
{
    // Display the metadata options
    Demo.display( "¥nMetadata options:¥n" );
    for( int opt_no = 0; opt_no < options.count(); ++ opt_no ) {
        Option option = options.elementAt( opt_no );
        option.display();
    }
    // Display the metadata tables
    Demo.display( "¥nMetadata tables:" );
    for( int table_no = 0; table_no < tables.count(); ++ table_no ) {
        Table table = tables.elementAt( table_no );
        table.display( table_no );
    }
    // Display the rows for non-system tables.
    for( int table_no = 0; table_no < tables.count(); ++ table_no ) {
        Table table = tables.elementAt( table_no );
        if( 0 == ( table.getFlags() & TABLE_FLAG_SYSTEM ) ) {
            Demo.display( "¥nRows for table: ", table.getName(), "¥n" );
            Index index = table.getIndex( 0 );
            PreparedStatement stmt = conn.prepareStatement(
                    "SELECT * FROM ¥"" + table.getName() + "¥""
                );
            ResultSet cursor = stmt.executeQuery();
            int column_count = table.getColumnCount();
            int row_count = 0;
            for( ; cursor.next(); ) {
                StringBuffer buf = new StringBuffer();
                buf.append( "Row[" );
                buf.append( Integer.toString( ++row_count ) );
                buf.append( "]:" );
                char joiner = ' ';
                for( int col_no = 1; col_no <= column_count; ++col_no ) {
                    String value = cursor.isNull( col_no )
                                ? "<NULL>"
                                : cursor.getString( col_no );
                    buf.append( joiner );
                    buf.append( value );
                    joiner = ',';
                }
                Demo.display( buf.toString() );
            }
            cursor.close();
            stmt.close();
        }
    }
}

/** Find a table.
 */
private static Table findTable( int table_id )
{
```

```
        Table retn = null;
        for( int i = tables.count(); i > 0; ) {
            Table table = tables.elementAt( --i );
            if( table_id == table.getId() ) {
                retn = table;
                break;
            }
        }
        return retn;
    }

    /** Representation of a column.
     */
    private static class Column
    {
        private int _column_id;
        private String _column_name;
        private int _column_flags;
        private Domain _domain;
        private int _column_default;

      Column( Connection conn, int column_id, String column_name, int column_flags, int column_domain,
int column_length, int column_default )
            throws ULjException
        {
            _column_id = column_id;
            _column_name = column_name;
            _column_flags = column_flags;
            _column_default = column_default;
            int scale = 0;
            switch( column_domain ) {
              case Domain.NUMERIC :
                scale = column_length >> 8;
                column_length &= 255;
                _domain = conn.createDomain( Domain.NUMERIC, column_length, scale );
                break;
              case Domain.VARCHAR :
              case Domain.BINARY :
                _domain = conn.createDomain( column_domain, column_length );
                break;
              default :
                _domain = conn.createDomain( column_domain );
                break;
            }
        }

        String getName()
        {
            return _column_name;
        }

        void display()
        {
            StringBuffer buf = new StringBuffer();
            buf.append( "  ¥"" );
            buf.append( _column_name );
            buf.append( "¥"¥t" );
            buf.append( _domain.getName() );
            switch( _domain.getType() ) {
              case Domain.NUMERIC :
                buf.append( '(' );
                buf.append( Integer.toString( _domain.getPrecision() ) );
                buf.append( ',' );
                buf.append( Integer.toString( _domain.getScale() ) );
```



```
                buf.append( ')' );
                break;
              case Domain.VARCHAR :
              case Domain.BINARY :
                buf.append( '(' );
                buf.append( Integer.toString( _domain.getSize() ) );
                buf.append( ')' );
                break;
            }
            if( 0 != ( _column_flags & COLUMN_FLAG_IS_NULLABLE ) ) {
                buf.append( " NULL" );
            } else {
                buf.append( " NOT NULL" );
            }
            switch( _column_default ) {
                case ColumnSchema.COLUMN_DEFAULT_NONE:
                default:
                    break;
                case ColumnSchema.COLUMN_DEFAULT_AUTOINC:
                    buf.append( " DEFAULT AUTOINCREMENT" );
                    break;
                case ColumnSchema.COLUMN_DEFAULT_GLOBAL_AUTOINC:
                    buf.append( " DEFAULT GLOBAL AUTOINCREMENT" );
                    break;
                case ColumnSchema.COLUMN_DEFAULT_CURRENT_DATE:
                    buf.append( " DEFAULT CURRENT DATE" );
                    break;
                case ColumnSchema.COLUMN_DEFAULT_CURRENT_TIME:
                    buf.append( " DEFAULT CURRENT TIME" );
                    break;
                case ColumnSchema.COLUMN_DEFAULT_CURRENT_TIMESTAMP:
                    buf.append( " DEFAULT CURRENT TIMESTAMP" );
                    break;
                case ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID:
                    buf.append( " DEFAULT NEWID()" );
                    break;
            }
            buf.append( ", /* column_id=" );
            buf.append( Integer.toString( _column_id ) );
            buf.append( " column_flags=" );
            int c = 0;
            if( 0 != ( _column_flags & COLUMN_FLAG_IN_PRIMARY_INDEX ) ) {
                buf.append( "IN_PRIMARY_INDEX" );
                c++;
            }
            if( 0 != ( _column_flags & COLUMN_FLAG_IS_NULLABLE ) ) {
                if( c > 0 ) {
                    buf.append( "," );
                }
                buf.append( "NULLABLE" );
            }
            buf.append( " */" );
            Demo.display( buf.toString() );
        }
    }

    /** Representation of Index schema.
     */
    private static class Index
    {
        private String _index_name;
        private int _index_id;
        private int _index_flags;
        private IndexColumnArray _columns;
```

```
Index( int index_id, String index_name, int index_flags )
{
    _index_id = index_id;
    _index_name = index_name;
    _index_flags = index_flags;
    _columns = new IndexColumnArray();
}

void addColumn( IndexColumn column )
{
    _columns.append( column );
}

void display( Table table, boolean constraints )
{
    StringBuffer buf = new StringBuffer();
    String flags = "";
    String indent = "  ";
    if( 0 != ( _index_flags & INDEX_FLAG_PRIMARY_INDEX ) ) {
        if( !constraints )      return;
        buf.append( "  CONSTRAINT ¥"" );
        buf.append( _index_name );
        buf.append( "¥" PRIMARY KEY (" );
        flags = "PRIMARY_KEY,UNIQUE_KEY";
    } else if( 0 != ( _index_flags & INDEX_FLAG_UNIQUE_KEY ) ) {
        if( !constraints )      return;
        buf.append( "  CONSTRAINT ¥"" );
        buf.append( _index_name );
        buf.append( "¥" UNIQUE (" );
        flags = "UNIQUE_KEY";
    } else {
        if( constraints )       return;
        if( 0 != ( _index_flags & INDEX_FLAG_UNIQUE_INDEX ) ) {
            buf.append( "UNIQUE " );
            flags = "UNIQUE_INDEX";
        }
        indent = "";
        buf.append( "INDEX ¥"" );
        buf.append( _index_name );
        buf.append( "¥" ON ¥"" );
        buf.append( table.getName() );
        buf.append( "¥" (" );
    }
    buf.append( "¥n" );
    buf.append( indent );
    buf.append( "  /* index_id=" );
    buf.append( Integer.toString( _index_id ) );
    buf.append( " index_flags=" );
    buf.append( flags );
    if( 0 != ( _index_flags & INDEX_FLAG_PERSISTENT ) ) {
        buf.append( ",PERSISTENT" );
    }
    buf.append( " */" );
    Demo.display( buf.toString() );
    int bounds = _columns.count();
    for( int col_no = 0; col_no < bounds; ++ col_no ) {
        IndexColumn column = _columns.elementAt( col_no );
        column.display( table, indent, col_no + 1 < bounds );
    }
    Demo.display( indent + ")" );
}

String getName()
```



```
    {
        return _index_name;
    }
}

/** Representation of IndexColumn schema.
 */
private static class IndexColumn
{
    private int _index_column_id;
    private int _index_column_column_id;
    private int _index_column_flags;

    IndexColumn( int index_column_id, int index_column_column_id, int index_column_flags )
    {
        _index_column_id = index_column_id;
        _index_column_column_id = index_column_column_id;
        _index_column_flags = index_column_flags;
    }

    void display( Table table, String indent, boolean notlast )
    {
        StringBuffer buf = new StringBuffer( indent );
        buf.append( "  ¥"" );
        Column column = table.getColumn( _index_column_column_id );
        buf.append( column.getName() );
        if( 0 != ( _index_column_flags & INDEX_COLUMN_FLAG_FORWARD ) ) {
            buf.append( "¥" ASC" );
        } else {
            buf.append( "¥" DESC" );
        }
        if( notlast ) {
            buf.append( "," );
        }
        Demo.display( buf.toString() );
    }
}

/** Representation of a database Option.
 */
private static class Option
{
    private String _option_name;
    private String _option_value;

    Option( String name, String value )
    {
        _option_name = name;
        _option_value = value;
    }

    void display()
    {
        StringBuffer buf = new StringBuffer();
        buf.append( "Option[ " );
        buf.append( _option_name );
        buf.append( " ] = "" );
        buf.append( _option_value );
        buf.append( """ );
        Demo.display( buf.toString() );
    }

}
```

```
/** Representation of Table schema.
 */
private static class Table
{
    private String _table_name;
    private int _table_id;
    private int _table_flags;
    private ColumnArray _columns;
    private IndexArray _indexes;

    Table( int table_id, String table_name, int table_flags )
    {
        _table_name = table_name;
        _table_id = table_id;
        _table_flags = table_flags;
        _columns = new ColumnArray();
        _indexes = new IndexArray();
    }

    void addColumn( Column column )
    {
        _columns.append( column );
    }

    void addIndex( Index index )
    {
        _indexes.append( index );
    }

    Column getColumn( int id )
    {
        return _columns.elementAt( id );
    }

    int getColumnCount()
    {
        return _columns.count();
    }

    int getFlags()
    {
        return _table_flags;
    }

    Index getIndex( int id )
    {
        return _indexes.elementAt( id );
    }

    String getName()
    {
        return _table_name;
    }

    void display( int logical_number )
    {
        StringBuffer str = new StringBuffer();
        str.append( "¥nTABLE ¥"" );
        str.append( _table_name );
        str.append( "¥" /* table_id=" );
        str.append( Integer.toString( _table_id ) );
        str.append( " table_flags=" );
        if( 0 == _table_flags ) {
            str.append( "0" );
```



```
            } else {
                int c = 0;
                if( 0 != ( _table_flags & TABLE_FLAG_SYSTEM ) ) {
                    str.append( "SYSTEM" );
                    c++;
                }
                if( 0 != ( _table_flags & TABLE_FLAG_NO_SYNC ) ) {
                    if( c > 0 ) {
                        str.append( "," );
                    }
                    str.append( "NO_SYNC" );
                    c++;
                }
            }
            str.append( " */ (" );
            Demo.display( str.toString() );
            int bound = _columns.count();
            for( int col_no = 0; col_no < bound; ++col_no ) {
                Column column = _columns.elementAt( col_no );
                column.display();
            }
            bound = _indexes.count();
            for( int idx_no = 0; idx_no < bound; ++idx_no ) {
                Index index = _indexes.elementAt( idx_no );
                index.display( this, true );
            }
            Demo.display( ")" );
            for( int idx_no = 0; idx_no < bound; ++idx_no ) {
                Index index = _indexes.elementAt( idx_no );
                index.display( this, false );
            }
        }

        int getId()
        {
            return _table_id;
        }
    }

    /** Simple adjustable array of objects.
     */
    private static class ObjArray
    {
        private Object[] _array = new Object[ 10 ];
        private int _used = 0;

        void append( Object str )
        {
            if( _used >= _array.length ) {
                Object[] new_array = new Object[ _used * 2 ];
                for( int i = _used; i > 0; ) {
                    --i;
                    new_array[ i ] = _array[ i ];
                }
                _array = new_array;
            }
            _array[ _used++ ] = str;
        }

        int count()
        {
            return _used;
        }
```

```
    Object getElementAt( int position )
    {
        return _array[ position ];
    }
}

/** Simple adjustable array of Strings.
 */
private static class StrArray
    extends ObjArray
{
    String elementAt( int position )
    {
        return (String)getElementAt( position );
    }
}

/** Simple adjustable array of Table objects.
 */
private static class TableArray
    extends ObjArray
{
    Table elementAt( int position )
    {
        return (Table)getElementAt( position );
    }
}

/** Simple adjustable array of Column objects.
 */
private static class ColumnArray
    extends ObjArray
{
    Column elementAt( int position )
    {
        return (Column)getElementAt( position );
    }
}

/** Simple adjustable array of Index objects.
 */
private static class IndexArray
    extends ObjArray
{
    Index elementAt( int position )
    {
        return (Index)getElementAt( position );
    }
}

/** Simple adjustable array of IndexColumn objects.
 */
private static class IndexColumnArray
    extends ObjArray
{
    IndexColumn elementAt( int position )
    {
        return (IndexColumn)getElementAt( position );
    }
}

/** Simple adjustable array of Option objects.
 */
private static class OptionArray
```



```
        extends ObjArray
    {
        Option elementAt( int position )
        {
            return (Option)getElementAt( position );
        }
    }

}
```

次に、アプリケーションの出力の一部を示します。

```
Metadata options:

Option[ date_format ] = 'YYYY-MM-DD'
Option[ date_order ] = 'YMD'
Option[ global_database_id ] = '0'
Option[ nearest_century ] = '50'
Option[ precision ] = '30'
Option[ scale ] = '6'
Option[ time_format ] = 'HH:NN:SS.SSS'
Option[ timestamp_format ] = 'YYYY-MM-DD HH:NN:SS.SSS'
Option[ timestamp_increment ] = '1'

Metadata tables:

Table[0] name = "systable"  id = 0 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "table_name" flags = 0x0 domain = VARCHAR(128)
  column[2 ]: name = "table_flags" flags = 0x0 domain = UNSIGNED-SHORT
  column[3 ]: name = "table_data" flags = 0x0 domain = INTEGER
  column[4 ]: name = "table_autoinc" flags = 0x0 domain = BIG
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD

Table[1] name = "syscolumn"  id = 1 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "column_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[2 ]: name = "column_name" flags = 0x0 domain = VARCHAR(128)
  column[3 ]: name = "column_flags" flags = 0x0 domain = TINY
  column[4 ]: name = "column_domain" flags = 0x0 domain = TINY
  column[5 ]: name = "column_length" flags = 0x0 domain = INTEGER
  column[6 ]: name = "column_default" flags = 0x0 domain = TINY
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "column_id" flags = 0x1,FORWARD

Table[2] name = "sysindex"  id = 2 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "index_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[2 ]: name = "index_name" flags = 0x0 domain = VARCHAR(128)
  column[3 ]: name = "index_flags" flags = 0x0 domain = TINY
  column[4 ]: name = "index_data" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "index_id" flags = 0x1,FORWARD

Table[3] name = "sysindexcolumn"  id = 3 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "index_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
```

```
  column[2 ]: name = "order" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[3 ]: name = "column_id" flags = 0x0 domain = INTEGER
  column[4 ]: name = "index_column_flags" flags = 0x0 domain = TINY
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "index_id" flags = 0x1,FORWARD
    key[2 ]: name = "order" flags = 0x1,FORWARD

Table[4] name = "sysinternal"  id = 4 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "name" flags = 0x1,IN-PRIMARY-INDEX domain = VARCHAR(128)
  column[1 ]: name = "value" flags = 0x0 domain = VARCHAR(128)
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "name" flags = 0x1,FORWARD

Table[5] name = "syspublications"  id = 5 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "publication_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "publication_name" flags = 0x0 domain = VARCHAR(128)
  column[2 ]: name = "download_timestamp" flags = 0x0 domain = TIMESTAMP
  column[3 ]: name = "last_sync_sent" flags = 0x0 domain = INTEGER
  column[4 ]: name = "last_sync_confirmed" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "publication_id" flags = 0x1,FORWARD

Table[6] name = "sysarticles"  id = 6 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "publication_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "publication_id" flags = 0x1,FORWARD
    key[1 ]: name = "table_id" flags = 0x1,FORWARD

Table[7] name = "sysforeignkey"  id = 7 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "foreign_table_id" flags = 0x0 domain = INTEGER
  column[2 ]: name = "foreign_key_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[3 ]: name = "name" flags = 0x0 domain = VARCHAR(128)
  column[4 ]: name = "index_name" flags = 0x0 domain = VARCHAR(128)
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "foreign_key_id" flags = 0x1,FORWARD

Table[8] name = "sysfkcol"  id = 8 flags = 0xc000,SYSTEM,NO_SYNC
  column[0 ]: name = "table_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[1 ]: name = "foreign_key_id" flags = 0x1,IN-PRIMARY-INDEX domain = INTEGER
  column[2 ]: name = "item_no" flags = 0x1,IN-PRIMARY-INDEX domain = SHORT
  column[3 ]: name = "column_id" flags = 0x0 domain = INTEGER
  column[4 ]: name = "foreign_column_id" flags = 0x0 domain = INTEGER
  index[0 ]: name = "primary" flags = 0xf,UNIQUE-KEY,UNIQUE-INDEX,PERSISTENT,PRIMARY-
INDEX
    key[0 ]: name = "table_id" flags = 0x1,FORWARD
    key[1 ]: name = "foreign_key_id" flags = 0x1,FORWARD
    key[2 ]: name = "item_no" flags = 0x1,FORWARD
```


チュートリアル：BlackBerry アプリケーションの構築

目次

Ultra Light J 開発の概要

このチュートリアルでは、Research In Motion の BlackBerry Java 開発環境を使用して、BlackBerry スマートフォン用の Ultra Light J アプリケーションを開発する方法について説明します。このチュートリアルでは、アプリケーションを BlackBerry シミュレータで実行してから物理デバイスに配備します。チュートリアル全体に渡ってコード例を示しています。また、チュートリアルの最後にすべてのコードを示します。

このチュートリアルは、次のことを前提にしています。

- ユーザが Java プログラミング言語に精通している。
- コンピュータに Research In Motion の BlackBerry JDE 4.0 以降がインストールされている。
- コンピュータに Ultra Light J がインストールされている。Ultra Light J のデフォルトのインストール先は *install-dir¥UltraLite¥UltraLiteJ* です。
- Research In Motion の MDS Services Simulator がコンピュータにインストールされている。これは第 2 部で必要です。

第 1 部：BlackBerry での Ultra Light J アプリケーションの作成

第 1 部では、シンプルな Ultra Light J データベースで名前のリストを管理する BlackBerry アプリケーションを作成する方法について説明します。第 2 部では、アプリケーションを Mobile Link サーバと同期する方法について説明します。

レッスン 1：BlackBerry JDE プロジェクトの作成

このレッスンでは、BlackBerry の Java Development Environment (JDE) プロジェクトを新規に作成します。

1. JDE の **[File]** メニューから、**[New Workspace]** を選択します。
2. ワークスペースのロケーション (*c:¥tutorials* など) を選択します。ワークスペースに **HelloBlackBerry** という名前を付け、**[OK]** をクリックします。
3. このチュートリアルでは、ワークスペースに 1 つプロジェクトが含まれています。**[Project]** メニューから、**[Create New Project]** を選択します。
4. プロジェクトに **HelloBlackBerry** という名前を付け、**[OK]** をクリックします。
5. プロジェクトに Ultra Light J JAR ファイルを追加します。
   a. **[Workspace]** ウィンドウでプロジェクトを右クリックし、**[Properties]** を選択します。
   b. **[Build]** タブで、**[Imported Jar Files]** フィールドの横にある **[Add]** をクリックします。
   c. Ultra Light J インストール・ディレクトリ内の *UltraLiteJ¥J2meRim11¥UltraLiteJ.jar* ファイルを参照し、**[Open]** をクリックします。
   d. **[OK]** をクリックして **[Properties]** ウィンドウを閉じます。
6. **[Project]** メニューから、**[Set Active Projects]** を選択します。HelloBlackBerry プロジェクトを選択し、**[OK]** をクリックします。
7. プロジェクトを保存します。

レッスン 2：BlackBerry アプリケーション画面の表示

このレッスンでは、HomeScreen を開く main メソッドがあるクラスを作成します。HomeScreen にはタイトルとステータス・メッセージが表示されます。

1. ワークスペース・ビューでプロジェクトを右クリックし、**[Create New File in Project]** を選択します。
2. **[Source File Name]** ボックスに **myapp¥Application.java** と入力して、myapp パッケージの一部である *Application.java* という名前のファイルを作成します。

[OK] をクリックしてファイルを作成します。Application クラスが JDE ウィンドウに表示されます。

3. Application クラスを定義します。このクラスにはインポートは不要です。コンストラクタと main メソッドを追加して、Application クラスを次のように定義します。

```
class Application extends net.rim.device.api.ui.UiApplication {
    public static void main( String[] args )
    {
        Application instance = new Application();
        instance.enterEventDispatcher();
    }
    Application() {
        pushScreen( new HomeScreen() );
    }
}
```

4. プロジェクトに HomeScreen クラスを追加します。
   a. ワークスペース・ビューでプロジェクトを右クリックし、**[Create New File in Project]** を選択します。
   b. **[Source File Name]** ボックスに **myapp¥HomeScreen.java** と入力します。
   c. **[OK]** をクリックしてファイルを作成します。

   HomeScreen クラスが JDE ウィンドウに表示されます。
5. タイトルとステータス・メッセージを表示するように、HomeScreen クラスを定義します。

```
package myapp;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import java.util.*;

class HomeScreen extends MainScreen {

    HomeScreen() {

        // Set the window title
        LabelField applicationTitle = new LabelField("Hello BlackBerry");
        setTitle(applicationTitle);

        // Add a label to show application status
        _statusLabel = new LabelField( "Status: Started" );
        add( _statusLabel );
    }
    private LabelField _statusLabel;
}
```

_statusLabel は、アプリケーションの他の部分からアクセスできるように、クラス変数として定義します。

6. プロジェクトを右クリックして、**[Build]** を選択します。エラーなくコンパイルされることを確認します。
7. [F5] を押して、Device Simulator でアプリケーションを実行します。

   別のウィンドウで BlackBerry シミュレータが起動します。

8. シミュレータ上で **[Applications]** ウィンドウに移動して、HelloBlackBerry アプリケーションを選択します。
9. アプリケーションを起動します。

   タイトル・バー **Hello BlackBerry** とステータス行 **Status: started** が表示されたウィンドウが開きます。
10. JDE の **[Debug]** メニューから **[Stop Debugging]** を選択します。

    シミュレータが終了します。

レッスン 3：Ultra Light J データベースの作成

このレッスンでは、Ultra Light J データベースを作成し、そのデータベースに接続するコードを記述します。新しいデータベースを作成するためのコードは、DataAccess というシングルトン・クラスで定義され、HomeScreen コンストラクタから呼び出されます。シングルトン・クラスを使用すると、データベースへの接続が、一度に 1 つしか開かれないよう制御できます。Ultra Light J は複数の接続をサポートしていますが、一般的な設計パターンでは 1 つの接続を使用します。

1. HomeScreen コンストラクタを変更して DataAccess オブジェクトをインスタンス化します。

   次に、更新した完全な HomeScreen クラスを示します。DataAccess オブジェクトは、コードの他の部分からアクセスできるように、クラスレベル変数として格納されます。

```
class HomeScreen extends MainScreen {

    HomeScreen() {

        // Set the window title
        LabelField applicationTitle = new LabelField("Hello BlackBerry");
        setTitle(applicationTitle);

        // Add a label to show application status
        _statusLabel = new LabelField( "Status: Started");
        add( _statusLabel );

        // Create database and connect
        try{
            _da = DataAccess.getDataAccess(false);
            _statusLabel.setText("Status: Connected");
        }
        catch( Exception ex)
        {
            _statusLabel.setText("Exception: " + ex.toString() );
        }
    }
    private LabelField _statusLabel;
    private DataAccess _da;
}
```

2. HelloBlackBerry プロジェクトに *myapp¥DataAccess.java* というファイルを作成します。
3. データベース接続が 1 つになるようにする getDataAccess メソッドを記述します。

```
package myapp;

import ianywhere.ultralitej.*;
import java.util.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;

class DataAccess {
   DataAccess() {   }

   public static synchronized DataAccess getDataAccess(boolean reset)
            throws Exception
   {
      if( _da == null ){
         _da = new DataAccess();
         ConfigObjectStore _config =
          DatabaseManager.createConfigurationObjectStore("HelloDB");
         if(reset)
         {
            _conn = DatabaseManager.createDatabase( _config );
         }
         else
         {
            try{
               _conn = DatabaseManager.connect( _config );
            }
            catch( ULjException uex1) {
               if( uex1.getErrorCode() !=
                  ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
                  System.out.println( "Exception: " +
                              uex1.toString() );
                  Dialog.alert( "Exception: " + uex1.toString() +
                           ". Recreating database..." );
               }
               _conn = DatabaseManager.createDatabase( _config );
            }
         }
         // _da.createDatabaseSchema();
      }
      return _da;
   }
   private static Connection _conn;
   private static DataAccess _da;
}
```

このクラスは、*UltraLiteJ.jar* ファイルから ianywhere.ultralitej パッケージをインポートします。データベースを作成する、またはデータベースに接続する手順は、次のとおりです。

a. 設定を定義します。この例では、ConfigObjectStore 設定オブジェクトです。これは、Ultra Light J データベースが BlackBerry のオブジェクト・ストア内に永続的に存在することを意味します。

b. データベースへの接続を試行します。

   接続の試行に失敗した場合は、データベースを作成します。createDatabase メソッドが、開いた接続を返します。

4. [F5] を押してアプリケーションを構築し、Device Simulator に配備します。

5. **[File]** メニューから、**[Load Java Program]** を選択します。

6. Ultra Light J インストール・ディレクトリの *J2meRim11* フォルダを参照し、*UltraLiteJ.cod* ファイルを開きます。
7. シミュレータからプログラムを実行します。

   アプリケーションがデータベースに正常に接続したことを示すステータス・メッセージが表示されます。

レッスン 4：データベース・テーブルの作成

このレッスンでは、次のプロパティを持つ 2 つのカラムが含まれる Names という名前のシンプルなテーブルを作成します。

カラム名	データ型	NULL 入力可	デフォルト	プライマリ・キー
ID	UUID	いいえ	なし	はい
Name	varchar(254)	いいえ	なし	いいえ

1. テーブルを作成する DataAccess メソッドを追加します。

```
private void createDatabaseSchema()
{
    try{
        _conn.schemaCreateBegin();
        ColumnSchema column_schema;
        TableSchema table_schema = _conn.createTable("Names");
        column_schema = table_schema.createColumn( "ID", Domain.UUID );
        column_schema.setDefault( ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID);
        table_schema.createColumn( "Name", Domain.VARCHAR, 254 );
        IndexSchema index_schema =
              table_schema.createPrimaryIndex("prime_keys");
        index_schema.addColumn("ID", IndexSchema.ASCENDING);
        _conn.schemaCreateComplete();
    }
    catch( ULjException uex1){
        System.out.println( "ULjException: " + uex1.toString() );
    }
    catch( Exception ex1){
        System.out.println( "Exception: " + ex1.toString() );
    }
}
```

   すでにテーブルが存在する場合は、例外がスローされます。

2. DataAccess.getDataAccess メソッドを呼び出します。

   第 1 部のレッスン 3 の手順 3 で使用したサンプル・コードに含まれる createDatabaseSchema の呼び出しのコメントを解除します。createDatabaseSchema の呼び出しは次のようになります。

```
_da.createDatabaseSchema()
```

3. シミュレータでアプリケーションを再度実行します。

テーブル・スキーマの変更

テーブル定義を追加するなどしてテーブル・スキーマを変更するときは、次の点を考慮する必要があります。

- スキーマの作成は、すべて schemaCreateBegin 呼び出しと schemaCreateEnd 呼び出しの間で行います。
- Connection.createTable メソッドで空のテーブルを作成します。
- TableSchema.createColumn メソッドでカラムを作成します。
- TableSchema.createPrimaryIndex メソッドでプライマリ・キーを作成します。

レッスン 5：テーブルへのデータの追加

このレッスンでは、画面に次のコントロールを追加します。

- 名前を入力できるテキスト・フィールド。
- テキスト・フィールドの名前をデータベースに追加するためのメニュー項目。
- テーブル内の名前を表示するリスト・フィールド。

次に、テキスト・フィールドに名前を挿入し、リストを更新するためのコードを追加します。

1. 画面にコントロールを追加します。
   a. getDataAccess メソッドを呼び出す前に次のコードを追加します。

```
// Add an edit field for entering new names
_nameEditField = new EditField( "Name: ", "", 50, EditField.USE_ALL_WIDTH );
add (_nameEditField );

// Add an ObjectListField for displaying a list of names
_nameListField = new ObjectListField();
add( _nameListField );

// Add a menu item
addMenuItem(_addToListMenuItem);

// Create database and connect
try{
    _da = DataAccess.getDataAccess();
```

   b. _nameEditField と _nameListField に対するクラスレベルの宣言を追加します。さらに、run メソッド (現時点では空の状態) で _addToListMenuItem MenuItem を定義します。これらの宣言は、_statusLabel と _da の宣言の次にあります。

```
private EditField _nameEditField;
private ObjectListField _nameListField;

private MenuItem _addToListMenuItem = new MenuItem("Add", 1, 1){
    public void run() {
        // TODO
    }
};
```

   c. アプリケーションを再コンパイルし、動作することを確認します。

2. 次のメソッドとオブジェクトをアプリケーションに追加します。

   - テーブルにローを追加するための DataAccess メソッド
   - Names テーブルのローをオブジェクトとして保持するためのオブジェクト
   - テーブルのローをオブジェクトのベクトルに読み込むための DataAccess メソッド
   - HomeScreen 上に表示されるリストの内容を再表示するためのメソッド
   - HomeScreen 上のリストに項目を追加するためのメソッド

   a. テーブルにローを挿入するための DataAccess メソッドを追加します。

```
public void insertName( String name ){
   try{
      Value nameID = _conn.createUUIDValue();
      String sql = "INSERT INTO Names( ID, Name ) VALUES
                ( ?, ? )";
      PreparedStatement ps = _conn.prepareStatement(sql);
      ps.set(1, nameID );
      ps.set(2, name );
      ps.execute();
      _conn.commit();
   }
   catch( ULjException uex ){
      System.out.println( "ULjException: " + uex.toString() );
   }
   catch( Exception ex ){
      System.out.println( "Exception: " + ex.toString() );
   }
}
```

   b. Names テーブルのローを保持するクラスを追加します。toString メソッドは、ObjectListField コントロールによって使用されます。

```
package myapp;

class NameRow {

   public NameRow( String nameID, String name ) {
      _nameID = nameID;
      _name = name;
   }

   public String getNameID(){
      return _nameID;
   }

   public String getName(){
      return _name;
   }

   public String toString(){
      return _name;
   }

   private String _nameID;
   private String _name;

}
```

   c. テーブルのローをオブジェクトのベクトルに読み込むための DataAccess メソッドを追加します。

```
public Vector getNameVector(){
    Vector nameVector = new Vector();
    try{
        String sql = "SELECT ID, Name FROM Names";
        PreparedStatement ps = _conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        while ( rs.next() ){
            String nameID = rs.getString(1);
            String name = rs.getString(2);
            NameRow nr = new NameRow( nameID, name);
            nameVector.addElement(nr);
        }
    }
    catch( ULjException uex ){
        System.out.println( "ULjException: " + uex.toString() );
    }
    catch( Exception ex ){
        System.out.println( "Exception: " + ex.toString() );
    }
    finally{
        return nameVector;
    }
}
```

d. ユーザ・インタフェースのメソッドを HomeScreen クラスに追加します。次に、名前のリストを更新するメソッドを示します。

```
public void refreshNameList(){
    //Clear the list
    _nameListField.setSize(0);
    //Refill from the list of names
    Vector nameVector = _da.getNameVector();
    for( Enumeration e = nameVector.elements(); e.hasMoreElements(); ){
        NameRow nr = ( NameRow )e.nextElement();
        _nameListField.insert(0, nr);
    }
}
```

e. HomeScreen コンストラクタの終了直前に refreshNameList メソッドを呼び出して、アプリケーションが起動するときにリストにデータが含まれているようにします。

```
// Fill the ObjectListField
this.refreshNameList();
```

f. ローをリストに追加する HomeScreen メソッドを追加します。

```
private void onAddToList(){
    _da.insertName(_nameEditField.getText());
    this.refreshNameList();
    _nameEditField.setText("");
}
```

g. _addToListMenuItem MenuItem の run メソッド (現時点では //TODO という記述) 内からこのメソッドを呼び出します。

```
public void run() {
    onAddToList();
}
```

3. アプリケーションをコンパイルして実行します。

シミュレータのリセット
シミュレータをクリーンな状態にリセットする必要がある場合は、BlackBerry JDE の **[File]** メニュー (**[Simulator]** メニューではない) から **[Erase Simulator File]** を選択し、サブメニューの項目を消去します。この方法でシミュレータをリセットする場合は、アプリケーションを再度実行する前に *UltraLiteJ.cod* ファイルを再インポートする必要があります。

レッスン 6：スマートフォンへのアプリケーションの配備

BlackBerry スマートフォンにアプリケーションを配備するには、いくつかの方法があります。このレッスンでは、BlackBerry Desktop Manager ソフトウェアを使用してアプリケーションを配備する方法について説明します。

BlackBerry 上で実行するアプリケーションは、BlackBerry Signature Tool を使用して署名する必要があります。このツールは、BlackBerry JDE Component Package の一部として Research in Motion (RIM) から入手できます。*UltraLiteJ.cod* ファイルはすでに署名されていますが、*HelloBlackBerry.cod* ファイルは署名する必要があります。

注意
BlackBerry Signature Tool を使用してアプリケーションに署名できるように、RIM からキーを取得してください。キーの取得については、BlackBerry Developer Program の Web サイト http://na.blackberry.com/eng/developers/ を参照してください。

アプリケーションの署名

1. BlackBerry Signature Tool を起動します。
2. コンパイル済みアプリケーションである *HelloBlackBerry.cod* ファイルを探して選択します。
3. **[Request To Sign The File]** をクリックします。
4. **[Close]** をクリックして Signature Tool を閉じます。

アプリケーションの配備

この手順では、BlackBerry Desktop Manager を使用してファイルをデバイスに配備する方法について説明します。

1. USB ケーブルを使用して BlackBerry をコンピュータに接続し、Desktop Manager からデバイスを認識できることを確認します。
2. **[Application Loader]** をクリックして、ウィザードの指示に従います。
3. *HelloBlackBerry.alx* ファイルを参照し、デバイスに追加します。
4. *J2meRim11¥UltraLiteJ.alx* ファイルを参照し、デバイスに追加します。

これで、BlackBerry スマートフォン上でアプリケーションを使用できるようになったはずです。

第 2 部：BlackBerry アプリケーションへの同期の追加

第 2 部では、同期を処理するようにアプリケーションを拡張します。それには、SQL Anywhere データベースを作成し、Mobile Link サーバを実行し、BlackBerry アプリケーションからの同期機能を追加します。

レッスン 1：SQL Anywhere データベースの作成

データの同期には、Ultra Light J が同期する統合データベースが必要です。このレッスンでは、SQL Anywhere データベースを作成します。

1. アプリケーション・ディレクトリ内に、SQL Anywhere データベースを保持する *c:¥tutorial¥database* というサブディレクトリを作成します。
2. *c:¥tutorial¥database* から次のコマンドを実行して空の SQL Anywhere データベースを作成します。

   ```
   dbinit HelloBlackBerry.db
   ```

3. ODBC データ・ソースを作成してデータベースに接続します。
   a. ODBC アドミニストレータを開きます。

      [スタート] - [プログラム] - [SQL Anywhere 11] - [ODBC アドミニストレータ] を選択します。

   b. **[ユーザー DSN]** タブをクリックして現在のユーザの ODBC データ・ソースを作成します。
   c. **[追加]** をクリックします。
   d. ドライバのリストから **SQL Anywhere 11** を選択し、**[完了]** をクリックします。
   e. **[ODBC]** タブをクリックします。
   f. **[データ・ソース名]** フィールドに **HelloBlackBerry** と入力します。
   g. **[ログイン]** タブをクリックします。
   h. **[ユーザ ID]** フィールドに **DBA** と入力し、**[パスワード]** フィールドに **SQL** と入力します。

      これらはすべての SQL Anywhere データベースにログインする際のデフォルトのユーザ名とパスワードですが、運用環境では使用しないでください。

   i. **[データベース]** タブをクリックして、**[サーバ名]** として **HelloBlackBerry**、**[データベース・ファイル]** として *c:¥tutorial¥database¥HelloBlackBerry.db* を入力します。**[OK]** をクリックします。
4. 次のコマンドを実行して、Interactive SQL を起動し、SQL Anywhere データベースに接続します。

   ```
   dbisql -c dsn=HelloBlackBerry
   ```

5. 次の文を実行してデータベース内にテーブルを作成します。


```
CREATE TABLE Names (
 ID UNIQUEIDENTIFIER NOT NULL DEFAULT newID(),
 Name varchar(254),
 PRIMARY KEY (ID)
);
```

6. Interactive SQL を閉じます。

レッスン 2：Mobile Link スクリプトの作成と Mobile Link サーバの起動

このレッスンでは、Sybase Central を使用して、同期する統合データベースを準備します。

1. **[スタート] - [プログラム] - [SQL Anywhere 11] - [Sybase Central]** を選択します。
2. Sybase Central の **[タスク]** ウィンドウ枠で、Mobile Link 11 のタスクから **[同期モデルの作成]** を選択します。
   a. 同期モデル名 **HelloBlackBerrySyncModel** を入力し、モデルを *c:¥tutorial¥database* フォルダに保存します。**[次へ]** をクリックします。
   b. **[統合データベースの選択]** をクリックします。
   c. **[統合データベースへの接続]** ウィンドウで、ODBC データ・ソース HelloBlackBerry を選択して **[OK]** をクリックします。
   d. **[はい]** をクリックして Mobile Link システム設定を作成します。
   e. **[いいえ、新しいリモート・データベース・スキーマを作成します]** を選択します。
   f. ダウンロードは、**[タイムスタンプベースのダウンロード]** を選択します。
   g. **[完了]** をクリックして同期モデルの作成を完了し、プロジェクトを保存します。
3. 同期モデルを右クリックして、**[展開]** を選択します。統合データベースのみに展開するよう選択します。
4. **同期モデル展開ウィザード**で、**[次の SQL ファイルに変更を保存する]** を選択して、データベースに展開します。統合データベースを指定するプロンプトが表示されたら、ODBC データ・ソース **HelloBlackBerry** を指定します。
5. Mobile Link ユーザとパスワードには、ユーザ名 **mluser** とパスワード **mlpassword** を選択します。
6. **[完了]** をクリックして、同期モデルを統合データベースに展開します。

レッスン 3 ：アプリケーションへの同期の追加

このレッスンでは、アプリケーションに同期機能を追加します。

1. HomeScreen コンストラクタに同期メニュー項目を追加します。

```
// Add a menu item
addMenuItem(_addToListMenuItem);
```

```
// Add sync menu item
addMenuItem(_syncMenuItem);

// Create database and connect
try{ ...
```

2. クラス変数宣言でメニュー項目を定義します。

```
private MenuItem _addToListMenuItem = new MenuItem("Add", 1, 1){
        public void run() {
            onAddToList();
        }
};
private MenuItem _syncMenuItem = new MenuItem("Sync", 2, 1){
        public void run() {
            onSync();
        }
};
```

3. onSync メソッドを作成します。

```
private void onSync(){
    try{
        if( _da.sync() ){
            _statusLabel.setText("Synchronization succeeded");
        } else {
            _statusLabel.setText("Synchronization failed");
          }
        this.refreshNameList();
    } catch ( Exception ex){
        System.out.println( ex.toString() );
    }
}
```

4. syncParms 変数と streamParms 変数をクラスレベルで定義します。

```
private static SyncParms _syncParms;
private static StreamHTTPParms _streamParms;
```

5. DataAccess クラスに sync メソッドを追加します。

```
public boolean sync() {
    try {
        if( _syncParms == null ){
            String host = "ultralitej.sybase.com";
            _syncParms = _conn.createSyncParms( "mluser", "HelloBlackBerrySyncModel" );
            _syncParms.setPassword("mlpassword");
            _streamParms = _syncParms.getStreamParms();
            _streamParms.setPort( 80 ); // use your own
            _streamParms.setHost( host ); // use your own
            if(host.equals("ultralitej.sybase.com"))
            {
                _streamParms.setURLSuffix("scripts/iaredirect.dll/ml/HelloBlackBerry/");
            }
        }
        System.out.println( "Synchronizing" );
        _conn.synchronize( _syncParms );
        return true;
    }
    catch( ULjException uex){
        System.out.println(uex.toString());
        return false;
```

```
    }
}
```

同期パラメータ・オブジェクト SyncParms には、同期モデルの展開時に指定したユーザ名とパスワードが含まれています。また、作成した同期モデルの名前も含まれています。Mobile Link では、この名前は統合データベースに展開された同期バージョン (同期論理セット) を参照するようになります。

ストリーム・パラメータ・オブジェクト StreamHTTPParms は、Mobile Link サーバのホスト名とポート番号を示します。次のレッスンで Mobile Link サーバを起動するときに、自分のコンピュータ名を使用し、使用可能なポートを選択します。コンピュータ名として localhost は使用しないでください。自分のコンピュータで Web サーバを実行していないかぎり、ポート 80 を使用できます。

6. アプリケーションをコンパイルします。

レッスン 4：シミュレータでのアプリケーションの実行

BlackBerry アプリケーションを実行して同期する前に、Mobile Link サーバが実行中である必要があります。Device Simulator と Mobile Link 間の通信チャネルを提供するためには、MDS Simulator も実行中である必要があります。

1. *c:¥tutorial¥database¥* から次のコマンドを実行して Mobile Link を起動します。

```
mlsrv11 -c " DSN=HelloBlackBerry" -v+ -x http(port=8081) -ot ml.txt
```

-c オプションは、Mobile Link を SQL Anywhere データベースに接続します。-v+ オプションは、高い冗長レベルを設定して、処理中の内容をサーバ・ウィンドウで確認できるようにします。-x オプションは、通信に使用されているポート番号を示します。-ot オプションは、ログ・ファイル (*ml.txt*) が、Mobile Link サーバを起動したディレクトリに作成されるように指定します。

2. **[スタート] - [プログラム] - [Research In Motion] - [BlackBerry Email And MDS Services Simulator 4.1.2] - [MDS]** を選択します。
3. サーバに名前を入力します。
   a. Interactive SQL を起動して HelloBlackBerry データ・ソースに接続します。
   b. 次の SQL 文を実行して名前を追加します。

```
INSERT Names ( Name ) VALUES ( 'ServerName1' );
INSERT Names ( Name ) VALUES ( 'ServerName2' );
COMMIT;
```

4. JDE で [F5] を押してアプリケーションをコンパイルし、Device Simulator 上で実行します。
5. メイン画面に移動し、リストに名前を追加します。
6. アプリケーションを同期します。
7. メイン画面から、メニュー項目を表示して **[Sync]** を選択します。

サーバに入力した名前が画面に表示されます。Interactive SQL から Names テーブル内の名前を問い合わせると、これまでにシミュレータに入力した名前がすべてサーバに到達していることが確認できます。

チュートリアルのコード・リスト

この項では、このチュートリアルの完全なコードを示します。チュートリアルで使用した Java クラスは 4 つあります。

参照

- 「第 1 部：BlackBerry での Ultra Light J アプリケーションの作成」 67 ページ
- 「第 2 部：BlackBerry アプリケーションへの同期の追加」 76 ページ

Application.java

```
/*
 * Application.java
 *
 * c <your company here>, 2003-2005
 * Confidential and proprietary.
 */

package myapp;


/**
 *
 */
class Application extends net.rim.device.api.ui.UiApplication {

    public static void main( String[] args )
    {
        Application instance = new Application();
        instance.enterEventDispatcher();
    }

    Application() {
        pushScreen( new HomeScreen() );
    }
}
```

DataAccess.java

```
/*
 * DataAccess.java
 *
 * c <your company here>, 2003-2005
 * Confidential and proprietary.
 */

package myapp;


import ianywhere.ultralitej.*;
import java.util.*;
import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
```

```
/**
 *
 */
class DataAccess {
    DataAccess() {    }

    public static synchronized DataAccess getDataAccess(boolean reset)
       throws Exception
    {
    try{
        if( _da == null ){
            _da = new DataAccess();
            ConfigObjectStore _config =
               DatabaseManager.createConfigurationObjectStore("HelloDB");
            if(reset)
            {
                _conn = DatabaseManager.createDatabase( _config );
            }
            else
            {
                try{
                    _conn = DatabaseManager.connect( _config );
                }
                catch( ULjException uex1) {
                    if( uex1.getErrorCode() !=
                        ULjException.SQLE_ULTRALITE_DATABASE_NOT_FOUND ) {
                        System.out.println( "Exception: " + uex1.toString() );
                        Dialog.alert( "Exception: " + uex1.toString() +
                           ". Recreating database..." );
                    }
                    _conn = DatabaseManager.createDatabase( _config );
                }
            }
            _da.createDatabaseSchema();
        }
        return _da;
    } catch ( ULjException ue)
    {
        System.out.println("Exception in getDataAccess" + ue.toString() );
        return null;
    }
    }

    /**
     *
     * Create the table in the database.
     * If the table already exists, a harmless exception is thrown
     */
    private void createDatabaseSchema()
    {
        try{
            _conn.schemaCreateBegin();
            ColumnSchema column_schema;
            TableSchema table_schema = _conn.createTable("Names");
            column_schema = table_schema.createColumn( "ID", Domain.UUID );
            column_schema.setDefault( ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID);
            table_schema.createColumn( "Name", Domain.VARCHAR, 254 );
            IndexSchema index_schema =
               table_schema.createPrimaryIndex("prime_keys");
            index_schema.addColumn("ID", IndexSchema.ASCENDING);
            _conn.schemaCreateComplete();
        }
        catch( ULjException uex1){
            System.out.println( "ULjException: " + uex1.toString() );
```



```
    }
    catch( Exception ex1){
        System.out.println( "Exception: " + ex1.toString() );
    }
}

public void insertName( String name ){
    try{
        Value nameID = _conn.createUUIDValue();
        String sql = "INSERT INTO Names( ID, Name ) VALUES ( ?, ? )";
        PreparedStatement ps = _conn.prepareStatement(sql);
        ps.set(1, nameID );
        ps.set(2, name );
        ps.execute();
        _conn.commit();
    }
    catch( ULjException uex ){
        System.out.println( "ULjException: " + uex.toString() );
    }
    catch( Exception ex ){
        System.out.println( "Exception: " + ex.toString() );
    }
}

public Vector getNameVector(){
    Vector nameVector = new Vector();
    try{
        String sql = "SELECT ID, Name FROM Names";
        PreparedStatement ps = _conn.prepareStatement(sql);
        ResultSet rs = ps.executeQuery();
        while ( rs.next() ){
            String nameID = rs.getString(1);
            String name = rs.getString(2);
            NameRow nr = new NameRow( nameID, name);
            nameVector.addElement(nr);
        }
    }
    catch( ULjException uex ){
        System.out.println( "ULjException: " + uex.toString() );
    }
    catch( Exception ex ){
        System.out.println( "Exception: " + ex.toString() );
    }
    finally{
        return nameVector;
    }
}

public boolean sync() {
    try {
        if( _syncParms == null ){
            String host = "ultralitej.sybase.com";
            _syncParms = _conn.createSyncParms( "mluser",
                        "HelloBlackBerrySyncModel" );
            _syncParms.setPassword("mlpassword");
            _streamParms = _syncParms.getStreamParms();
            _streamParms.setPort( 80 ); // use your own
            _streamParms.setHost( host ); // use your own
            if(host.equals("ultralitej.sybase.com"))
            {
                _streamParms.setURLSuffix(
                  "scripts/iaredirect.dll/ml/HelloBlackBerry/");
            }
```

```
            }
            System.out.println( "Synchronizing" );
            _conn.synchronize( _syncParms );
            return true;
        }
        catch( ULjException uex){
            System.out.println(uex.toString());
            return false;
        }
    }

    public boolean complete() {
        try{
            _conn.checkpoint();
            _conn.release();
            _conn = null;
            _da = null;
            _config = null;
            return true;
        }
        catch(Exception e){
            return false;
        }
    }

    private static ConfigObjectStore _config;
    private static Connection _conn;
    private static DataAccess _da;
    private static SyncParms _syncParms;
    private static StreamHTTPParms _streamParms;
}
```

HomeScreen.java

```
/*
 * HomeScreen.java
 *
 * c <your company here>, 2003-2005
 * Confidential and proprietary.
 */

package myapp;


import net.rim.device.api.ui.*;
import net.rim.device.api.ui.component.*;
import net.rim.device.api.ui.container.*;
import java.util.*;
/**
 *
 */
class HomeScreen extends MainScreen {


    HomeScreen() {

        // Set the window title
        LabelField applicationTitle = new LabelField("Hello BlackBerry");
        setTitle(applicationTitle);

        // Add a label to show application status
```



```
        _statusLabel = new LabelField( "Status: Started");
        add( _statusLabel );

        // Add an edit field for entering new names
        _nameEditField = new EditField( "Name: ", "", 50,
            EditField.USE_ALL_WIDTH );
        add (_nameEditField );

        // Add an ObjectListField for displaying a list of names
        _nameListField = new ObjectListField();
        add( _nameListField );

        // Add a menu item
        addMenuItem(_addToListMenuItem);

        // Add sync menu item
        addMenuItem(_syncMenuItem);

        // Add reset menu item
        addMenuItem(_resetMenuItem);

        // Create database and connect
        try{
            _da = DataAccess.getDataAccess(false);
            _statusLabel.setText("Status: Connected");
        }
        catch( Exception ex)
        {
            System.out.println("Exception: " + ex.toString() );
            _statusLabel.setText("Exception: " + ex.toString() );
        }

        // Fill the ObjectListField
        this.refreshNameList();

    }

    public void refreshNameList(){
        try{
            //Clear the list
            _nameListField.setSize(0);
            //Refill from the list of names
            Vector nameVector = _da.getNameVector();
            for( Enumeration e = nameVector.elements(); e.hasMoreElements(); ){
                NameRow nr = ( NameRow )e.nextElement();
                _nameListField.insert(0, nr);
            }
        } catch ( Exception ex){
            System.out.println(ex.toString());
        }
    }

    private void onAddToList(){
        String name = _nameEditField.getText();
        _da.insertName(name);
        this.refreshNameList();
        _nameEditField.setText("");
        _statusLabel.setText(name + " added to list");
    }

    private void onSync(){
        try{
            if( _da.sync() ){
                _statusLabel.setText("Synchronization succeeded");
```

```
            } else {
                _statusLabel.setText("Synchronization failed");
            }
            this.refreshNameList();
        } catch ( Exception ex){
            System.out.println( ex.toString() );
        }
    }

    private void onReset(){
        _da.complete();
        try{
            _da = DataAccess.getDataAccess(true);
            _statusLabel.setText("Status: Connected");
            this.refreshNameList();
        }
        catch( Exception ex)
        {
            System.out.println("Exception: " + ex.toString() );
            _statusLabel.setText("Exception: " + ex.toString() );
        }
    }

    private LabelField _statusLabel;
    private DataAccess _da;
    private EditField _nameEditField;
    private ObjectListField _nameListField;
    private MenuItem _addToListMenuItem = new MenuItem("Add", 1, 1){
        public void run() {
            onAddToList();
        }
    };
    private MenuItem _syncMenuItem = new MenuItem("Sync", 2, 1){
        public void run() {
            onSync();
        }
    };
    private MenuItem _resetMenuItem = new MenuItem("Reset", 3, 1){
        public void run() {
            onReset();
        }
    };

}
```

NameRow.java

```
/*
 * NameRow.java
 *
 * c <your company here>, 2003-2005
 * Confidential and proprietary.
 */

package myapp;



/**
 * Hold a row of the Name table as an object
```



```
 */
class NameRow {

    public NameRow( String nameID, String name ) {
        _nameID = nameID;
        _name = name;
    }

    public String getNameID(){
        return _nameID;
    }

    public String getName(){
        return _name;
    }


    /**
     * Required for use by the ObjectListField in HomeScreen
     *
     * @return The Name as a string
     */
    public String toString(){
        return _name;
    }

    private String _nameID;
    private String _name;

}
```

Ultra Light J リファレンス

Ultra Light J API リファレンス

目次

パッケージ

ianywhere.ultralitej

CollectionOfValueReaders インタフェース

特定のローのカラム値を返すメソッドを提供します。

構文

public **CollectionOfValueReaders**

派生クラス

- 「ResultSet インタフェース」 184 ページ

備考

返される結果は、PreparedStatement で宣言する SQL SELECT 文に基づき、ResultSet に格納されます。

このインタフェースのメソッドによって返される値は、整数パラメータ ordinal に基づいてアクセスします。このパラメータは、SQL SELECT 文でのカラムの順序を指定します。ordinal は基数インデックスを 1 とします。

値は、java プリミティブ型または読み込み専用の Value オブジェクトのいずれかとして返されます。

次の例は、SQL SELECT 文を使用して新しい PreparedStatement を宣言し、その文を実行し、クエリの結果を新しい ResultSet に格納し、get メソッドを使用して column1 の値を String として格納する方法を示しています。

```
// Define a new SQL SELECT statement.
String sql_string = "SELECT column1, column2 FROM SampleTable";

// Create a new PreparedStatement from an existing connection.
PreparedStatement ps = conn.prepareStatement(sql_string);

// Create a new ResultSet to contain the query results of the SQL statement.
ResultSet rs = ps.executeQuery();

// Check if the PreparedStatement contains a ResultSet.
if (ps.hasResultSet()) {
   // Retrieve the column1 value using getString.
   String row1_col1 = rs.getString(1);
}
```

メンバ

CollectionOfValueReaders のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getBlobInputStream メソッド」 94 ページ
- 「getBoolean メソッド」 94 ページ
- 「getBytes メソッド」 95 ページ
- 「getClobReader メソッド」 95 ページ
- 「getDate メソッド」 95 ページ
- 「getDecimalNumber メソッド」 96 ページ
- 「getDouble メソッド」 96 ページ
- 「getFloat メソッド」 96 ページ
- 「getInt メソッド」 97 ページ
- 「getLong メソッド」 97 ページ
- 「getOrdinal メソッド」 98 ページ
- 「getString メソッド」 98 ページ
- 「getValue メソッド」 98 ページ
- 「isNull メソッド」 99 ページ

getBlobInputStream メソッド

InputStream を返します。

構文

```
java.io.InputStream CollectionOfValueReaders.getBlobInputStream(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の InputStream 表現。

getBoolean メソッド

boolean 値を返します。

構文

```
boolean CollectionOfValueReaders.getBoolean(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の boolean 表現。

getBytes メソッド

byte 配列を返します。

構文

```
byte[] CollectionOfValueReaders.getBytes(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の byte 配列表現。

getClobReader メソッド

Reader を返します。

構文

```
java.io.Reader CollectionOfValueReaders.getClobReader(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の Reader 表現。

getDate メソッド

java.util.Date を返します。

構文

```
java.util.Date CollectionOfValueReaders.getDate(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の java.util.Date 表現。

getDecimalNumber メソッド

DecimalNumber を返します。

構文

```
DecimalNumber CollectionOfValueReaders.getDecimalNumber(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の DecimalNumber 表現。

getDouble メソッド

double 値を返します。

構文

```
double CollectionOfValueReaders.getDouble(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の double 表現。

getFloat メソッド

float 値を返します。

構文

float **CollectionOfValueReaders.getFloat(**
  int *ordinal*
) throws **ULjException**

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の float 表現。

getInt メソッド

integer 値を返します。

構文

int **CollectionOfValueReaders.getInt(**
  int *ordinal*
) throws **ULjException**

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の integer 表現。

getLong メソッド

long integer 値を返します。

構文

long **CollectionOfValueReaders.getLong(**
  int *ordinal*
) throws **ULjException**

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の long integer 表現。

getOrdinal メソッド

String で表現された値の (1 から始まる) 順序を返します。

構文

```
int CollectionOfValueReaders.getOrdinal(
  String name
) throws ULjException
```

パラメータ

- **name**　テーブルのカラム名を表す String。

戻り値

順序の値。

getString メソッド

String 値を返します。

構文

```
String CollectionOfValueReaders.getString(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の String 表現。

getValue メソッド

Value オブジェクトを返します。

構文

```
Value CollectionOfValueReaders.getValue(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の Value オブジェクト表現。

isNull メソッド

値が NULL かどうかをテストします。

構文

```
boolean CollectionOfValueReaders.isNull(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

値が NULL の場合は true、NULL 以外の場合は false。

CollectionOfValueWriters インタフェース

特定のローのカラム値を設定するメソッドを提供します。

構文

public **CollectionOfValueWriters**

派生クラス

- 「PreparedStatement インタフェース」 180 ページ

備考

すべてのメソッドは、PreparedStatement の最初の宣言で定義する SQL 文に基づいて準備し、execute メソッドでデータベースに適用します。

カラム値を更新するときは、準備された SQL 文での順序番号でカラムを参照する必要があります。この番号は、基数インデックスを 1 とする整数パラメータ ordinal として渡します。

次の例は、SQL UPDATE 文を使用して新しい PreparedStatement を宣言し、set メソッドを使用して変更を準備し、その変更を Ultra Light データベースに適用する方法を示しています。

```
// Define a new prepared SQL statement.
String sql_string = "UPDATE SampleTable SET column1 = ? WHERE pkey = 1";

// Create a new PreparedStatement from an existing connection.
PreparedStatement ps = conn.prepareStatement(sql_string);

// Set a String value to the first column in the SQL statement (column1).
ps.set(1, "New Value");

// Commit the changes to the database.
conn.commit();
```

メンバ

CollectionOfValueWriters のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getBlobOutputStream メソッド」 101 ページ
- 「getClobWriter メソッド」 101 ページ
- 「getOrdinal メソッド」 101 ページ
- 「set メソッド」 102 ページ
- 「set メソッド」 102 ページ
- 「set メソッド」 102 ページ
- 「set メソッド」 103 ページ
- 「set メソッド」 103 ページ
- 「set メソッド」 103 ページ
- 「set メソッド」 104 ページ
- 「set メソッド」 104 ページ
- 「set メソッド」 104 ページ
- 「set メソッド」 105 ページ
- 「setNull メソッド」 105 ページ

getBlobOutputStream メソッド

OutputStream を返します。

構文

```
java.io.OutputStream CollectionOfValueWriters.getBlobOutputStream(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の OutputStream。

getClobWriter メソッド

Writer を返します。

構文

```
java.io.Writer CollectionOfValueWriters.getClobWriter(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

戻り値

指定された値の Writer。

getOrdinal メソッド

name で指定された値の (1 から始まる) 順序を返します。

構文

```
int CollectionOfValueWriters.getOrdinal(
  String name
) throws ULjException
```

パラメータ

- **name**　テーブルのカラム名を表す String。

戻り値

name で指定された値の (1 から始まる) 順序。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に boolean 値を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  boolean value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。
- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に DecimalNumber を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  DecimalNumber value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。
- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に整数値を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  int value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に java.util.Date を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  java.util.Date value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。
- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に long integer 値を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  long value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。
- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に float 値を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  float value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に double 値を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  double value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。
- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に byte 配列値を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  byte[] value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。
- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に String 値を設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  String value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

- **value**　設定する値。

set メソッド

SQL 文内の ordinal で定義されたカラム番号に Value オブジェクトを設定します。

構文

```
void CollectionOfValueWriters.set(
  int ordinal,
  Value value
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。
- **value**　設定する値。

setNull メソッド

SQL 文内の ordinal で定義されたカラム番号に NULL 値を設定します。

構文

```
void CollectionOfValueWriters.setNull(
  int ordinal
) throws ULjException
```

パラメータ

- **ordinal**　SQL 文でのカラムの順序を表す 1 から始まる整数。

ColumnSchema インタフェース

カラムのスキーマを指定します。

構文

public **ColumnSchema**

備考

このインタフェースをサポートするオブジェクトは、TableSchema.createColumn(String,short)、TableSchema.createColumn(String,short,int)、TableSchema.createColumn(String,short,int,int) の各メソッドから返されます。

次の例は、単純なデータベースのスキーマを作成する方法を示しています。オートインクリメントする整数のプライマリ・キー・カラムのある T1 テーブルが作成されます。

```
// Assumes a valid Connection object
TableSchema table_schema;
ColumnSchema col_schema;
IndexSchema index_schema;

table_schema = conn.createTable("T1");
col_schema = table_schema.createColumn("num", Domain.INTEGER);
col_schema.setDefault(ColumnSchema.COLUMN_DEFAULT_AUTOINC);

// BIT columns are not nullable by default.
col_schema = table_schema.createColumn("flag", Domain.BIT);
col_schema.setNullable(true);
col_schema = table_schema.createColumn(
    "cost", Domain.NUMERIC, 10, 2
    );
col_schema.setNullable(false);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("num", IndexSchema.ASCENDING);
conn.schemaCreateComplete();
```

メンバ

ColumnSchema のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「COLUMN_DEFAULT_AUTOINC 変数」 107 ページ
- 「COLUMN_DEFAULT_CURRENT_DATE 変数」 107 ページ
- 「COLUMN_DEFAULT_CURRENT_TIME 変数」 107 ページ
- 「COLUMN_DEFAULT_CURRENT_TIMESTAMP 変数」 108 ページ
- 「COLUMN_DEFAULT_GLOBAL_AUTOINC 変数」 108 ページ
- 「COLUMN_DEFAULT_NONE 変数」 109 ページ
- 「COLUMN_DEFAULT_UNIQUE_ID 変数」 109 ページ
- 「setDefault メソッド」 110 ページ
- 「setNullable メソッド」 110 ページ

COLUMN_DEFAULT_AUTOINC 変数

カラムをオートインクリメントすることを指定します。

構文

final byte **ColumnSchema.COLUMN_DEFAULT_AUTOINC**

備考

AUTOINCREMENT を使用する場合、カラムは整数データ型の 1 つ、または真数値型にします。INSERT 時に AUTOINCREMENT カラムの値を指定しないと、カラム内の任意の値より大きいユニーク値が生成されます。INSERT で、カラムの現在の最大値より大きい値を指定した場合、この値が後続の挿入処理の開始ポイントとして使用されます。

Ultra Light J では、テーブルが作成された時点でのオートインクリメントの初期値は 0 ではありません。カラムに符号付きデータ型が指定されている場合は、オートインクリメントによって負の値が生成されます。このため、オートインクリメントを適用するカラムを符号なし整数として宣言し、負の値が生成されないようにしてください。

既存のテーブルのデフォルト値は、システム・テーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

● 「setDefault メソッド」 110 ページ

COLUMN_DEFAULT_CURRENT_DATE 変数

カラムのデフォルト値を現在の日付 (年、月、日) にすることを指定します。

構文

final byte **ColumnSchema.COLUMN_DEFAULT_CURRENT_DATE**

備考

SQL Anywhere のマニュアル・セットで、「Ultra Light の特別値」の下の「CURRENT DATE 特別値」を参照してください。

既存のテーブルのデフォルト値は、システム・テーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

● 「setDefault メソッド」 110 ページ

COLUMN_DEFAULT_CURRENT_TIME 変数

カラムのデフォルト値を現在の時刻にすることを指定します。

構文

final byte **ColumnSchema.COLUMN_DEFAULT_CURRENT_TIME**

備考

SQL Anywhere のマニュアル・セットで、「Ultra Light の特別値」の下の「CURRENT TIME 特別値」を参照してください。

既存のテーブルのデフォルト値は、システム・テーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- 「setDefault メソッド」 110 ページ

COLUMN_DEFAULT_CURRENT_TIMESTAMP 変数

カラムのデフォルト値を現在のタイムスタンプにすることを指定します。

構文

final byte **ColumnSchema.COLUMN_DEFAULT_CURRENT_TIMESTAMP**

備考

この定数は、CURRENT DATE と CURRENT TIME を結合して、TIMESTAMP 値を形成します。この値は、年、月、日、時、分、秒、秒の小数位で構成されます。秒の精度は小数点以下第 3 位に設定されます。この定数の精度はシステム・クロックの精度によって制限されます。SQL Anywhere のマニュアル・セットで、「Ultra Light の特別値」の下の「CURRENT TIMESTAMP 特別値」を参照してください。

既存のテーブルのデフォルト値は、システム・テーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- 「setDefault メソッド」 110 ページ

COLUMN_DEFAULT_GLOBAL_AUTOINC 変数

カラムをグローバル・オートインクリメントすることを指定します。

構文

final byte **ColumnSchema.COLUMN_DEFAULT_GLOBAL_AUTOINC**

備考

この定数は AUTOINCREMENT と同じですが、ドメインはパーティションに分割されます。各分割には同じ数の値が含まれます。データベースの各コピーにユニークなグローバル・データ

ベース ID 番号を割り当てる必要があります。Ultra Light J では、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割から設定されます。

既存のテーブルのデフォルト値は、システム・テーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- 「setDefault メソッド」 110 ページ
- 「setDatabaseId メソッド」 144 ページ

COLUMN_DEFAULT_NONE 変数

カラムに特別なデフォルト値がないことを指定します。

構文

final byte **ColumnSchema.COLUMN_DEFAULT_NONE**

備考

NULL 入力可のカラムのデフォルト値は NULL、NULL 入力不可の数値カラムのデフォルト値は 0、NULL 入力不可の可変長カラムのデフォルト値は長さ 0 の値になります。

既存のテーブルのデフォルト値は、システム・テーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- 「setDefault メソッド」 110 ページ
- 「setNullable メソッド」 110 ページ

COLUMN_DEFAULT_UNIQUE_ID 変数

カラムのデフォルト値を新しいユニークな識別子にすることを指定します。

構文

final byte **ColumnSchema.COLUMN_DEFAULT_UNIQUE_ID**

備考

UUID を使用して、テーブルのローをユニークに識別できます。生成される値は、すべてのコンピュータまたはデバイスでユニークになります。つまり、同期やレプリケーション環境でキーとして使用できます。

既存のテーブルのデフォルト値は、システム・テーブル TableSchema.SYS_COLUMNS の column_default カラムに問い合わせることで確認できます。

参照

- 「setDefault メソッド」 110 ページ

setDefault メソッド

カラムのデフォルト値を設定します。

構文

```
ColumnSchema ColumnSchema.setDefault(
  byte default_code
)
```

パラメータ

- **default_code**　ColumnSchema コードの 1 つ。カラムのデフォルト値の型を示す、COLUMN_DEFAULT サフィックスが付いた定数です。

備考

デフォルトは COLUMN_DEFAULT_NONE です。

参照

- 「COLUMN_DEFAULT_AUTOINC 変数」 107 ページ
- 「COLUMN_DEFAULT_CURRENT_DATE 変数」 107 ページ
- 「COLUMN_DEFAULT_CURRENT_TIME 変数」 107 ページ
- 「COLUMN_DEFAULT_CURRENT_TIMESTAMP 変数」 108 ページ
- 「COLUMN_DEFAULT_GLOBAL_AUTOINC 変数」 108 ページ
- 「COLUMN_DEFAULT_NONE 変数」 109 ページ
- 「COLUMN_DEFAULT_UNIQUE_ID 変数」 109 ページ

戻り値

デフォルト値が定義された ColumnSchema。

setNullable メソッド

カラムを NULL 入力可に設定します。

構文

```
ColumnSchema ColumnSchema.setNullable(
  boolean nullable
)
```

パラメータ

- **nullable**　このカラムに NULL 値を入力可能にする場合は true、それ以外の場合は false に設定します。

備考

プライマリ・キーとユニーク・キーに含まれるカラムは常に NULL 入力不可です。BIT 型のカラムはデフォルトでは NULL 入力不可です。

戻り値

NULL 入力可と定義された ColumnSchema。

ConfigFile インタフェース

ファイルに保存される永続的なデータベース用の設定を確立します。

構文

public **ConfigFile**

基本クラス

- 「Configuration インタフェース」 123 ページ
- 「ConfigPersistent インタフェース」 115 ページ

メンバ

ConfigFile のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getAutoCheckpoint メソッド」 115 ページ
- 「getCacheSize メソッド」 116 ページ
- 「getDatabaseName メソッド」 123 ページ
- 「getLazyLoadIndexes メソッド」 116 ページ
- 「getPageSize メソッド」 123 ページ
- 「hasPersistentIndexes メソッド」 116 ページ
- 「setAutocheckpoint メソッド」 116 ページ
- 「setCacheSize メソッド」 117 ページ
- 「setEncryption メソッド」 117 ページ
- 「setIndexPersistence メソッド」 118 ページ
- 「setLazyLoadIndexes メソッド」 118 ページ
- 「setPageSize メソッド」 124 ページ
- 「setPassword メソッド」 124 ページ
- 「setRowMaximumThreshold メソッド」 119 ページ
- 「setRowMinimumThreshold メソッド」 119 ページ
- 「setShadowPaging メソッド」 120 ページ
- 「setWriteAtEnd メソッド」 121 ページ
- 「writeAtEnd メソッド」 121 ページ

ConfigNonPersistent インタフェース

非永続的なデータベース用の設定を確立します。

構文

public **ConfigNonPersistent**

基本クラス

- 「Configuration インタフェース」 123 ページ

メンバ

ConfigNonPersistent のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getDatabaseName メソッド」 123 ページ
- 「getPageSize メソッド」 123 ページ
- 「setPageSize メソッド」 124 ページ
- 「setPassword メソッド」 124 ページ

ConfigObjectStore インタフェース (J2ME BlackBerry のみ)

オブジェクト・ストアに保存される永続的なデータベース用の設定を確立します。

構文

public **ConfigObjectStore**

基本クラス

- 「Configuration インタフェース」 123 ページ
- 「ConfigPersistent インタフェース」 115 ページ

メンバ

ConfigObjectStore のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getAutoCheckpoint メソッド」 115 ページ
- 「getCacheSize メソッド」 116 ページ
- 「getDatabaseName メソッド」 123 ページ
- 「getLazyLoadIndexes メソッド」 116 ページ
- 「getPageSize メソッド」 123 ページ
- 「hasPersistentIndexes メソッド」 116 ページ
- 「setAutocheckpoint メソッド」 116 ページ
- 「setCacheSize メソッド」 117 ページ
- 「setEncryption メソッド」 117 ページ
- 「setIndexPersistence メソッド」 118 ページ
- 「setLazyLoadIndexes メソッド」 118 ページ
- 「setPageSize メソッド」 124 ページ
- 「setPassword メソッド」 124 ページ
- 「setRowMaximumThreshold メソッド」 119 ページ
- 「setRowMinimumThreshold メソッド」 119 ページ
- 「setShadowPaging メソッド」 120 ページ
- 「setWriteAtEnd メソッド」 121 ページ
- 「writeAtEnd メソッド」 121 ページ

ConfigPersistent インタフェース

永続的なデータベース用の設定を確立します。

構文

public **ConfigPersistent**

基本クラス

- 「Configuration インタフェース」 123 ページ

派生クラス

- 「ConfigFile インタフェース」 112 ページ
- 「ConfigObjectStore インタフェース (J2ME BlackBerry のみ)」 114 ページ
- 「ConfigRecordStore インタフェース (J2ME のみ)」 122 ページ

メンバ

ConfigPersistent のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getAutoCheckpoint メソッド」 115 ページ
- 「getCacheSize メソッド」 116 ページ
- 「getDatabaseName メソッド」 123 ページ
- 「getLazyLoadIndexes メソッド」 116 ページ
- 「getPageSize メソッド」 123 ページ
- 「hasPersistentIndexes メソッド」 116 ページ
- 「setAutocheckpoint メソッド」 116 ページ
- 「setCacheSize メソッド」 117 ページ
- 「setEncryption メソッド」 117 ページ
- 「setIndexPersistence メソッド」 118 ページ
- 「setLazyLoadIndexes メソッド」 118 ページ
- 「setPageSize メソッド」 124 ページ
- 「setPassword メソッド」 124 ページ
- 「setRowMaximumThreshold メソッド」 119 ページ
- 「setRowMinimumThreshold メソッド」 119 ページ
- 「setShadowPaging メソッド」 120 ページ
- 「setWriteAtEnd メソッド」 121 ページ
- 「writeAtEnd メソッド」 121 ページ

getAutoCheckpoint メソッド

自動チェックポイントがオンになっているかどうかを確認します。

構文

boolean **ConfigPersistent.getAutoCheckpoint()**

戻り値

データベースで自動チェックポイントがオンになっている場合は true、それ以外の場合は false。

getCacheSize メソッド

データベースのキャッシュ・サイズ (バイト単位) を返します。

構文

int **ConfigPersistent.getCacheSize()**

戻り値

キャッシュ・サイズ。

getLazyLoadIndexes メソッド

インデックスの遅延ロードがオンになっているかどうかを確認します。

構文

boolean **ConfigPersistent.getLazyLoadIndexes()**

戻り値

遅延ロードがオンの場合は true、それ以外の場合は false。

hasPersistentIndexes メソッド

インデックスが永続的かどうかを確認します。

構文

boolean **ConfigPersistent.hasPersistentIndexes()**

戻り値

インデックスが永続的な場合は true、それ以外の場合は false。

setAutocheckpoint メソッド

自動チェックポイントをオンに設定します。

構文

```
ConfigPersistent ConfigPersistent.setAutocheckpoint(
  boolean auto_checkpoint
) throws ULjException
```


パラメータ

- **auto_checkpoint**　自動チェックポイントをオンに設定する場合は true。

備考

データベースは、コミットされた変更操作が、永続ストア内の永続的なローに適用されるときにチェックポイントがオンになります。自動チェックポイントが有効になっている場合は、コミット操作ごとにチェックポイントが発生します。有効ではない場合は、変更内容を記録するトランザクション・レコードが書き込まれ、永続的なローの記憶領域は、アプリケーションによって checkpoint メソッドが呼び出されるまで変更されません。

変更とコミットの操作は、自動チェックポイントが有効になっていないときのほうが高速ですが、チェックポイントがないトランザクションが多数あると、データベースの起動に時間がかかる場合があります。

インデックスが永続的ではないか、ロー制限が有効になっている場合は、自動チェックポイントは常に true です。

戻り値

自動チェックポイントが設定された ConfigPersistent。

setCacheSize メソッド

データベースのキャッシュ・サイズ (バイト単位) を設定します。

構文

```
ConfigPersistent ConfigPersistent.setCacheSize(
  int cache_size
) throws ULjException
```

パラメータ

- **cache_size**　キャッシュ・サイズ。すべてのプラットフォームで、デフォルトのキャッシュ・サイズは 20480 (20 KB)です。

備考

キャッシュ・サイズによって、ページ・キャッシュに常駐するデータベースのページ数が決まります。サイズを拡大すると、データベース・ページの読み込みと書き込みの回数が減りますが、キャッシュ内でページを検索する時間が長くなります。

戻り値

キャッシュ・サイズが設定された ConfigPersistent。

setEncryption メソッド

暗号化を設定します。

構文

```
ConfigPersistent ConfigPersistent.setEncryption(
  EncryptionControl control
)
```

パラメータ

- **control**　データベースの暗号化に使用する EncryptionControl オブジェクト。

戻り値

暗号化が設定された ConfigPersistent。

setIndexPersistence メソッド

永続的なインデックスをオンに設定します。

構文

```
ConfigPersistent ConfigPersistent.setIndexPersistence(
  boolean store
) throws ULjException
```

パラメータ

- **store**　インデックスを格納するには true に設定します。インデックスを初めて使用する前に構築するには false に設定します。

備考

この設定は、データベースの作成時にのみ使用されます。データベースに使用するインデックスの永続性の方法を決定します。

既存のデータベースを開くときは、作成時の設定が使用され、その値を反映して設定が更新されます。

戻り値

インデックスの永続性が設定された ConfigPersistent。

setLazyLoadIndexes メソッド

必要なときにインデックスをロードするか、起動時にすべてのインデックスを一度にロードするかを設定します。

構文

```
ConfigPersistent ConfigPersistent.setLazyLoadIndexes(
  boolean lazy_load
) throws ULjException
```

パラメータ

- **lazy_load**　必要なときにインデックスをロードする場合は true、起動時にすべてのインデックスを一度にロードする場合は false に設定します。

備考

このオプションを有効にすると、データベースの起動時間が短くなりますが、その後の操作が低速になる可能性があります。

戻り値

インデックスの遅延ロードが設定された ConfigPersistent。

setRowMaximumThreshold メソッド

メモリに維持するローの最大数のスレッショルドを設定します。

構文

```
ConfigPersistent ConfigPersistent.setRowMaximumThreshold(
  int threshold
)
```

パラメータ

- **threshold**　スレッショルドの最大値。

備考

ローの最大数に達したら、ローはトランケートされ、setRowMinimumThreshold メソッドで定義されたローの最小数が維持されます。

参照

- 「setRowMinimumThreshold メソッド」 119 ページ

戻り値

最大スレッショルドが設定された ConfigPersistent。

setRowMinimumThreshold メソッド

メモリに維持するローの最小数のスレッショルドを設定します。

構文

```
ConfigPersistent ConfigPersistent.setRowMinimumThreshold(
  int threshold
)
```

パラメータ

- **threshold**　スレッショルドの最小値。

備考

ローの最大数に達したら、ローはトランケートされ、setRowMinimumThreshold メソッドで定義されたローの最小数が維持されます。

参照

- 「setRowMaximumThreshold メソッド」 119 ページ

戻り値

最小スレッショルドが設定された ConfigPersistent。

setShadowPaging メソッド

シャドー・ページングをオンに設定します。

構文

```
ConfigPersistent ConfigPersistent.setShadowPaging(
  boolean shadow
) throws ULjException
```

パラメータ

- **shadow**　シャドー・ページングをオンにする場合は true、それ以外の場合は false に設定します。

備考

シャドー・ページングとは、永続ストアへの書き込みがすべて未使用のデータベース・ページに対して行われ、コミット操作が完了するまで永続的に格納されないことです。コミットされた変更は、アプリケーションが異常終了しても、永続的に保存されることが保証されます。

シャドー・ページングが false に設定されている場合、変更操作が行われてもコミットの完了前であればデータベースが破損する可能性があります。

シャドー・ページングで永続的にしなかった場合、データベース操作がより高速になり、返されるデータベースの結果が小さくなります。

データベース内のデータが重要ではないか、同期によってリカバリできる場合にのみ、シャドー・ページングなしでデータベースを処理するようにしてください。

戻り値

ShadowPaging が設定された ConfigPersistent。

setWriteAtEnd メソッド

シャットダウン時のインデックスの永続性をオンに設定します。

構文

ConfigPersistent **ConfigPersistent.setWriteAtEnd(**
  boolean *write_at_end*
) throws **ULjException**

パラメータ

- **write_at_end**　データベースを停止するまでメモリに維持する場合は true に設定します。

備考

このオプションを有効にすると、データベース操作が高速になりますが、アプリケーションが異常終了した場合はデータベースの変更内容がすべて失われます。

データベース内のデータが重要ではないか、同期によってリカバリできる場合にのみ、インデックスの永続性を有効にしてデータベースを処理するようにしてください。

戻り値

WriteAtEnd が設定された ConfigPersistent。

writeAtEnd メソッド

シャットダウン時のインデックスの永続性がオンになっているかどうかを確認します。

構文

boolean **ConfigPersistent.writeAtEnd()**

戻り値

シャットダウン時のインデックスの永続性がオンの場合は true、それ以外の場合は false。

ConfigRecordStore インタフェース (J2ME のみ)

J2ME レコード・ストアに保存される永続的なデータベース用の設定を確立します。

構文

public **ConfigRecordStore**

基本クラス

- 「Configuration インタフェース」 123 ページ
- 「ConfigPersistent インタフェース」 115 ページ

メンバ

ConfigRecordStore のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getAutoCheckpoint メソッド」 115 ページ
- 「getCacheSize メソッド」 116 ページ
- 「getDatabaseName メソッド」 123 ページ
- 「getLazyLoadIndexes メソッド」 116 ページ
- 「getPageSize メソッド」 123 ページ
- 「hasPersistentIndexes メソッド」 116 ページ
- 「setAutocheckpoint メソッド」 116 ページ
- 「setCacheSize メソッド」 117 ページ
- 「setEncryption メソッド」 117 ページ
- 「setIndexPersistence メソッド」 118 ページ
- 「setLazyLoadIndexes メソッド」 118 ページ
- 「setPageSize メソッド」 124 ページ
- 「setPassword メソッド」 124 ページ
- 「setRowMaximumThreshold メソッド」 119 ページ
- 「setRowMinimumThreshold メソッド」 119 ページ
- 「setShadowPaging メソッド」 120 ページ
- 「setWriteAtEnd メソッド」 121 ページ
- 「writeAtEnd メソッド」 121 ページ

Configuration インタフェース

データベース用の設定を確立します。

構文

public **Configuration**

派生クラス

- 「ConfigNonPersistent インタフェース」 113 ページ
- 「ConfigPersistent インタフェース」 115 ページ

備考

一部の属性は、データベースの作成時にのみ使用されます。その他の属性は、データベースへの最初の接続に適用されます。データベースの作成後、またはデータベースへの接続後に設定された属性は無視されます。

メンバ

Configuration のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getDatabaseName メソッド」 123 ページ
- 「getPageSize メソッド」 123 ページ
- 「setPageSize メソッド」 124 ページ
- 「setPassword メソッド」 124 ページ

getDatabaseName メソッド

データベース名を返します。

構文

String **Configuration.getDatabaseName()**

戻り値

データベースの名前。

getPageSize メソッド

データベースのページ・サイズ (バイト単位) を返します。

構文

int **Configuration.getPageSize()**

戻り値

ページ・サイズ。

setPageSize メソッド

データベースのページ・サイズ (バイト単位) を設定します。

構文

```
Configuration Configuration.setPageSize(
  int page_size
) throws ULjException
```

パラメータ

- **page_size**　ページ・サイズ。

備考

ページ・サイズの設定を使用して、永続的なデータベースに格納されるローの最大サイズが決定されます。また、このページ・サイズによって、インデックス・ページのサイズが確立され、各ページの子の数が決まります。

既存のデータベースを使用する場合は、データベースの作成時のページ・サイズにすでに設定されています。このメソッドを使用して、既存のデータベースのページ・サイズをリセットすることはできません。

ページ・サイズは 256 ～ 32736 バイトの範囲内です。デフォルトは 1024 バイトです。

戻り値

ページ・サイズが設定された Configuration オブジェクト。

setPassword メソッド

データベースのパスワードを設定します。

構文

```
Configuration Configuration.setPassword(
  String password
) throws ULjException
```

パラメータ

- **password**　新しいデータベースのパスワード、または既存のデータベースにアクセスするためのパスワード。

備考

このパスワードを使用してデータベースへのアクセスが許可されます。このパスワードは、データベースの作成時に指定されたパスワードと一致する必要があります。デフォルトは "dba" です。

戻り値

パスワードが設定された Configuration オブジェクト。

Connection インタフェース

データベース接続を表します。データベース操作を開始するには接続が必要です。

構文

public **Connection**

備考

接続は、DatabaseManager クラスの connect メソッドまたは createDatabase メソッドを使用して取得します。接続が不要になったら release メソッドを使用します。データベースのすべての接続を解放したら、データベースは終了します。

接続では次の操作が可能です。

- 新しいスキーマの作成 (テーブル、インデックス、パブリケーション)
- 新しい値とドメイン・オブジェクトの作成
- データベースへの変更の永続的なコミット
- 実行する SQL 文の準備
- コミットされていないデータベースへの変更のロールバック
- データベースへのチェックポイントの設定 (変更トランザクションを格納するだけではなく、コミットされた変更内容で基本となる永続ストアを更新)

次の例は、単純なデータベースのスキーマを作成する方法を示します。データベースにはテーブル T1 と T2 があります。T1 には num という整数のプライマリ・キー・カラムが 1 つあります。T2 には num という整数のプライマリ・キー・カラムと quantity という整数カラムがあります。T2 の quantity には追加インデックスがあります。T1 は PubA というパブリケーションに含まれます。

```
table_schema = conn.createTable("T1");
 table_schema.createColumn("num", Domain.INTEGER);
```

メンバ

Connection のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

CONNECTED 変数

接続した状態です。

構文

final byte **Connection.CONNECTED**

NOT_CONNECTED 変数

接続していない状態です。

構文

final byte **Connection.NOT_CONNECTED**

OPTION_DATABASE_ID 変数

データベース・オプション：データベース ID。

構文

final String **Connection.OPTION_DATABASE_ID**

OPTION_DATE_FORMAT 変数

データベース・オプション：日付形式。

構文

final String **Connection.OPTION_DATE_FORMAT**

OPTION_DATE_ORDER 変数

データベース・オプション：日付順。

構文

final String **Connection.OPTION_DATE_ORDER**

OPTION_ML_REMOTE_ID 変数

データベース・オプション：ML リモート ID。

構文

final String **Connection.OPTION_ML_REMOTE_ID**

備考

OPTION_NEAREST_CENTURY 変数

データベース・オプション：基準年。

構文

final String **Connection.OPTION_NEAREST_CENTURY**

OPTION_PRECISION 変数

データベース・オプション：精度。

構文

final String **Connection.OPTION_PRECISION**

OPTION_SCALE 変数

データベース・オプション：位取り。

構文

final String **Connection.OPTION_SCALE**

OPTION_TIMESTAMP_FORMAT 変数

データベース・オプション：タイムスタンプ形式。

構文

final String **Connection.OPTION_TIMESTAMP_FORMAT**

OPTION_TIMESTAMP_INCREMENT 変数

timestamp_increment データベース・オプションを示す定数。このオプションはタイムスタンプ値の精度を制限します。データベースにタイムスタンプが挿入されると、この増分に合わせてタイムスタンプはトランケートされます。使用できる値は 1 ～ 60000000 マイクロ秒です。デフォルトは 1 です (1000000 マイクロ秒は 1 秒に相当)。

構文

final String **Connection.OPTION_TIMESTAMP_INCREMENT**

OPTION_TIME_FORMAT 変数

データベース・オプション：時間形式。

構文

final String **Connection.OPTION_TIME_FORMAT**

PROPERTY_DATABASE_NAME 変数

データベース・プロパティ：データベース名。

構文

final String **Connection.PROPERTY_DATABASE_NAME**

PROPERTY_PAGE_SIZE 変数

データベース・プロパティ：ページ・サイズ。

構文

final String **Connection.PROPERTY_PAGE_SIZE**

SYNC_ALL 変数

データベース内の全テーブルの同期を要求するために使用するパブリケーションのリストです。どのパブリケーションにも含まれないテーブルも含まれます。

構文

final String **Connection.SYNC_ALL**

備考

NoSync と指定されているテーブルは同期されません。

この定数は、NULL 参照または空の文字列と同じです。

SYNC_ALL_DB_PUB_NAME 変数

SYNC_ALL_DB パブリケーションの予約名です。

構文

final String **Connection.SYNC_ALL_DB_PUB_NAME**

参照

- 「getLastDownloadTime メソッド」 140 ページ
- 「resetLastDownloadTime メソッド」 143 ページ

SYNC_ALL_PUBS 変数

データベース内の全パブリケーションの同期を要求するために使用するパブリケーションのリストです。

構文

final String **Connection.SYNC_ALL_PUBS**

備考

NoSync と指定されているテーブルは同期されません。

checkpoint メソッド

データベースの変更内容にチェックポイントを設定します。

構文

void **Connection.checkpoint()** throws **ULjException**

備考

この関数を呼び出すと、コミットされたすべてのトランザクションが、永続的なデータベースに適用されます。

commit メソッド

データベースの変更内容をコミットします。

構文

void **Connection.commit()** throws **ULjException**

備考

このメソッドを呼び出すと、最後のコミット後またはロールバック後に行われたデータベースへの変更がすべて永続的になります。

createDecimalNumber メソッド

DecimalNumber を作成します。

構文

```
DecimalNumber Connection.createDecimalNumber(
  int precision,
  int scale
) throws ULjException
```

パラメータ

- **precision**　数値の桁数。
- **scale**　数値の小数点以下の桁数。

戻り値

指定された型の DecimalNumber。

createDecimalNumber メソッド

DecimalNumber を作成します。

構文

```
DecimalNumber Connection.createDecimalNumber(
  int precision,
  int scale,
  String value
) throws ULjException
```

パラメータ

- **precision**　数値の桁数。
- **scale**　数値の小数点以下の桁数。
- **value**　設定する値。

戻り値

指定された型の DecimalNumber。

createDomain メソッド

固定サイズのドメインを作成します。

構文

```
Domain Connection.createDomain(
  int type
) throws ULjException
```

パラメータ

- **type**　ドメインの型。

戻り値

指定された型のドメイン。

createDomain メソッド

可変サイズのドメインを作成します。

構文

```
Domain Connection.createDomain(
  int type,
  int size
) throws ULjException
```

パラメータ

- **type**　ドメインの型。
- **size**　ドメインのサイズ。

戻り値

指定された型のドメイン。

createDomain メソッド

可変サイズのドメインを作成します。

構文

```
Domain Connection.createDomain(
  int type,
  int size,
  int scale
) throws ULjException
```

パラメータ

- **type**　ドメインの型。
- **size**　ドメインのサイズ。
- **scale**　ドメインの位取り。

戻り値

指定された型のドメイン。

createForeignKey メソッド

新しい外部キーを作成します。

構文

```
ForeignKeySchema Connection.createForeignKey(
  String table_name,
  String primary_table_name,
  String name
) throws ULjException
```

パラメータ

- **table_name**　外部キーを作成するテーブルの名前。プライマリ・テーブルへの有効な参照を持つよう制約されたテーブルです。
- **primary_table_name**　参照先カラムが含まれるテーブルの名前。
- **name**　外部キーの名前。指定する名前は有効な SQL 識別子である必要があります。

備考

注意
Ultra Light J では、テーブルに外部キー制約が適用されません。外部キーは、テーブルを同期する正しい順序を確認するために使用されます。クライアント・データベースの外部キーは、クライアントが同期する統合データベースの関係と一致している必要があります。

戻り値

名前とテーブルの関係が定義された ForeignKey。

createPublication メソッド

データベースに新しいパブリケーションを作成します。

構文

```
void Connection.createPublication(
  String pub_name,
  String[] tables
) throws ULjException
```

パラメータ

- **pub_name**　作成するパブリケーションの名前。
- **tables**　テーブル名の配列。

備考

データベース全体の同期には、特殊なパブリケーション・リスト Connection.SYNC_ALL を使用します。

参照

- 「dropPublication メソッド」 137 ページ
- 「setPublications メソッド」 235 ページ
- 「setTableOrder メソッド」 236 ページ

createSyncParms メソッド

HTTP 同期用の同期パラメータ・セットを作成します。

構文

```
SyncParms Connection.createSyncParms(
  String userName,
  String version
) throws ULjException
```

パラメータ

- **userName**　このクライアント・データベース用のユニークな Mobile Link ユーザ名。
- **version**　Mobile Link スクリプトのバージョン。

参照

- 「createSyncParms メソッド」 135 ページ
- 「setUserName メソッド」 237 ページ

戻り値

SyncParms オブジェクト。

createSyncParms メソッド

同期パラメータ・セットを作成します。

構文

```
SyncParms Connection.createSyncParms(
  int streamType,
  String userName,
  String version
) throws ULjException
```

パラメータ

- **streamType**　同期ストリームのタイプの指定に使用する、SyncParms クラス内で定義されている定数の 1 つ。
- **userName**　Mobile Link ユーザ名。
- **version**　Mobile Link スクリプトのバージョン。

参照

- 「createSyncParms メソッド」 134 ページ
- 「HTTP_STREAM 変数」 226 ページ
- 「HTTPS_STREAM 変数」 225 ページ

戻り値

SyncParms オブジェクト

createTable メソッド

データベースに新しいテーブルを作成します。

構文

```
TableSchema Connection.createTable(
  String table_name
) throws ULjException
```

パラメータ

- **table_name**　作成するテーブルの名前。

備考

このメソッドは、接続しているデータベースがスキーマ作成モードになっている場合にのみ実行できます。

戻り値

新しいテーブルの TableSchema オブジェクト。

参照

- 「schemaCreateBegin メソッド」 143 ページ
- 「schemaCreateComplete メソッド」 144 ページ

createUUIDValue メソッド

UUID 値を作成します。

構文

Value **Connection.createUUIDValue()** throws **ULjException**

戻り値

ドメインの Value。

createValue メソッド

ドメインから Value を作成します。

構文

Value **Connection.createValue(**
  Domain *dom*
) throws **ULjException**

パラメータ

- **dom**　指定されたドメイン。

戻り値

ドメインの Value。

disableSynchronization メソッド

テーブルの同期を無効にします。

構文

void **Connection.disableSynchronization(**
  String *table_name*
) throws **ULjException**

パラメータ

- **table_name**　テーブルの名前。

dropDatabase メソッド

データベースを削除します。

構文

void **Connection.dropDatabase()** throws **ULjException**

備考

接続によって参照されているデータベースを消去し、接続を解放します。削除するデータベースへの有効な接続が、この接続だけである必要があります。

dropForeignKey メソッド

外部キーを削除します。

構文

void **Connection.dropForeignKey(**
  String *table_name*,
  String *fkey_name*
) throws **ULjException**

パラメータ

- **table_name**　外部キーが含まれるテーブルの名前。
- **fkey_name**　削除する外部キーの名前。

dropPublication メソッド

データベースからパブリケーションを削除します。

構文

void **Connection.dropPublication(**
  String *pub_name*
) throws **ULjException**

パラメータ

- **pub_name**　削除するパブリケーションの名前。

備考

特殊なパブリケーション Connection.SYNC_ALL_DB_PUB_NAME は削除できません。

参照

- 「createPublication メソッド」 134 ページ

dropTable メソッド

データベースからテーブルを削除します。

構文

void **Connection.dropTable(**
  String *table_name*
) throws **ULjException**

パラメータ

- **table_name**　削除するテーブルの名前。

備考

現在の接続に、コミットされていない未処理のトランザクションがなく、テーブルがどのパブリケーションにも含まれていない場合にのみ、テーブルを削除できます。削除するテーブルにローがある場合、そのローは失われます。同期されていない操作がある場合はその操作も失われます。

emergencyShutdown メソッド

接続しているデータベースを緊急停止します。

構文

void **Connection.emergencyShutdown()** throws **ULjException**

備考

このメソッドは、重大なエラーが発生したときにのみ呼び出してください。物理的なハードウェアまたはデータが壊れた場合にのみ使用してください。

このメソッドは、開いている接続をすべて閉じ、接続しているデータベースを停止します。

enableSynchronization メソッド

テーブルの同期を有効にします。

構文

void **Connection.enableSynchronization(**
  String *table_name*
) throws **ULjException**

パラメータ

- **table_name**　テーブル名。

getDatabaseId メソッド

データベース ID の値を返します。

構文

int **Connection.getDatabaseId()** throws **ULjException**

戻り値

データベース ID。

getDatabaseInfo メソッド

データベース・プロパティに関する情報を含む DatabaseInfo オブジェクトを返します。

構文

DatabaseInfo **Connection.getDatabaseInfo()** throws **ULjException**

戻り値

DatabaseInfo オブジェクト。

getDatabasePartitionSize メソッド

データベース分割のサイズを返します。

構文

int **Connection.getDatabasePartitionSize()** throws **ULjException**

戻り値

データベース分割のサイズ。

参照

- 「getDatabaseId メソッド」 139 ページ

getDatabaseProperty メソッド

データベースのプロパティを返します。

構文

String **Connection.getDatabaseProperty(**
  String *name*
) throws ULjException

パラメータ

- **name**　データベース・プロパティの名前。

戻り値

指定された名前に対応するプロパティの値。

getLastDownloadTime メソッド

指定されたパブリケーションの最後のダウンロードの時刻を返します。

構文

Date **Connection.getLastDownloadTime(**
  String *pub_name*
) throws ULjException

パラメータ

- **pub_name**　チェックするパブリケーションの名前。

備考

パラメータ pub_name は、1 つのパブリケーションを参照するか、データベース全体を最後にダウンロードしたときの特殊なパブリケーション Connection.SYNC_ALL_DB_PUB_NAME である必要があります。

このメソッドは、schemaCreateComplete() の実行後でないと実行できません。

参照

- 「createPublication メソッド」 134 ページ
- 「resetLastDownloadTime メソッド」 143 ページ
- 「schemaCreateBegin メソッド」 143 ページ
- 「schemaCreateComplete メソッド」 144 ページ

戻り値

最後のダウンロードのタイムスタンプ。

getOption メソッド

データベース・オプションを返します。

構文

String **Connection.getOption(**
String *option_name*
) throws **ULjException**

パラメータ

- **option_name**　値を取得する設定オプションの値 (OPTION サフィックスの付いた変数)。

備考

データベース・オプションはデータベース内に格納され、オプションの設定後にデータベースに接続したときに取得できます。

データベースの作成時に一連の必須オプションが作成されます。

戻り値

データベース・オプションの値。

参照

- 「setOption メソッド」 144 ページ
- 「OPTION_DATABASE_ID 変数」 127 ページ
- 「OPTION_DATE_FORMAT 変数」 127 ページ
- 「OPTION_DATE_ORDER 変数」 128 ページ
- 「OPTION_ML_REMOTE_ID 変数」 128 ページ
- 「OPTION_NEAREST_CENTURY 変数」 128 ページ
- 「OPTION_PRECISION 変数」 128 ページ
- 「OPTION_SCALE 変数」 128 ページ
- 「OPTION_TIMESTAMP_FORMAT 変数」 129 ページ
- 「OPTION_TIMESTAMP_INCREMENT 変数」 129 ページ
- 「OPTION_TIME_FORMAT 変数」 129 ページ

getState メソッド

接続のステータスを返します。

構文

byte **Connection.getState()** throws **ULjException**

備考

有効な return 文は、Configuration インタフェースでサポートされている文です。

参照

- 「CONNECTED 変数」 127 ページ
- 「NOT_CONNECTED 変数」 127 ページ

prepareStatement メソッド

実行する文を準備します。

構文

PreparedStatement **Connection.prepareStatement(**
  String *sql*
) throws **ULjException**

パラメータ

- **sql**　準備する SQL 文。

参照

- 「PreparedStatement インタフェース」 180 ページ

戻り値

PreparedStatement オブジェクト。

release メソッド

接続を解放します。

構文

void **Connection.release()** throws **ULjException**

備考

一度解放した接続は、データベースへのアクセスに使用できなくなります。

コミットされていないトランザクションがある接続を解放しようとするとエラーが発生します。

renameTable メソッド

テーブルの名前を変更します。

構文

void **Connection.renameTable(**
  String *old_table_name*,
  String *new_table_name*
) throws **ULjException**

パラメータ

- **old_table_name**　既存のテーブルの名前。
- **new_table_name**　テーブルの新しい名前。

resetLastDownloadTime メソッド

指定されたパブリケーションのダウンロード時刻をリセットします。

構文

void **Connection.resetLastDownloadTime(**
String *pub_name*
) throws **ULjException**

パラメータ

- **pub_name**　チェックするパブリケーションの名前。

備考

データベース全体が同期されたダウンロード時刻をリセットするには、特殊なパブリケーション Connection.SYNC_ALL_DB_PUB_NAME を使用します。

このメソッドを使用するには、現在の接続に、コミットされていないトランザクションがないことが必要です。

参照

- 「createPublication メソッド」 134 ページ

rollback メソッド

データベースへの変更を取り消すロールバックをコミットします。

構文

void **Connection.rollback()** throws **ULjException**

備考

このメソッドを呼び出すと、この接続で、コミット後またはロールバック後に行われたデータベースへの変更がすべて取り消されます。

schemaCreateBegin メソッド

接続しているデータベースをスキーマ作成モードにします。

構文

void **Connection.schemaCreateBegin()** throws **ULjException**

備考

データベースがスキーマ作成モードのときは、データと同期の操作を行うことができません。データベースへの接続要求も拒否されます。

schemaCreateComplete メソッド

接続しているデータベースのスキーマ作成モードを終了します。

構文

void **Connection.schemaCreateComplete()** throws **ULjException**

setDatabaseId メソッド

グローバル・オートインクリメントのデータベース ID と分割サイズを設定します。

構文

void **Connection.setDatabaseId(**
  int *id*,
  int *size*
) throws **ULjException**

パラメータ

- **id**　データベース ID。
- **size**　分割のサイズ。

setOption メソッド

データベース・オプションを設定します。

構文

void **Connection.setOption(**
  String *option_name*,
  String *option_value*
) throws **ULjException**

パラメータ

- **option_name**　設定する設定オプションの値 (OPTION サフィックスの付いた変数)。
- **option_value**　オプションの新しい値。

備考

オプションが現在データベースに格納されていない場合は作成されます。

この接続にコミットされていないトランザクションがあるときに、このメソッドを呼び出すことはできません。

参照

- 「getOption メソッド」 140 ページ
- 「OPTION_DATABASE_ID 変数」 127 ページ
- 「OPTION_DATE_FORMAT 変数」 127 ページ
- 「OPTION_DATE_ORDER 変数」 128 ページ
- 「OPTION_ML_REMOTE_ID 変数」 128 ページ
- 「OPTION_NEAREST_CENTURY 変数」 128 ページ
- 「OPTION_PRECISION 変数」 128 ページ
- 「OPTION_SCALE 変数」 128 ページ
- 「OPTION_TIMESTAMP_FORMAT 変数」 129 ページ
- 「OPTION_TIMESTAMP_INCREMENT 変数」 129 ページ
- 「OPTION_TIME_FORMAT 変数」 129 ページ

startSynchronizationDelete メソッド

削除の同期を開始します。

構文

void **Connection.startSynchronizationDelete()** throws **ULjException**

備考

今後すべての削除が同期されるようになります。

stopSynchronizationDelete メソッド

削除の同期を停止します。

構文

void **Connection.stopSynchronizationDelete()** throws **ULjException**

備考

次に startSynchronizationDelete が実行されるまで、今後すべての削除が同期されなくなります。

synchronize メソッド

データベースを Mobile Link サーバと同期させます。

構文

void **Connection.synchronize(**
  SyncParms *config*
) throws **ULjException**

パラメータ

- **config**　同期に使用するパラメータ

備考

データベースにダウンロードが適用されるときにチェックポイントが設定されます。

参照

- 「checkpoint メソッド」 130 ページ

truncateTable メソッド

テーブル内のすべてのローを削除します。

構文

```
void Connection.truncateTable(
  String table_name
) throws ULjException
```

パラメータ

- **table_name**　トランケートするテーブルの名前。

備考

ローは同期されません。

DatabaseInfo インタフェース

Connection オブジェクトに関連付けられ、データベース情報を公開するメソッドを提供します。

構文

public **DatabaseInfo**

備考

このインタフェースは、Connection オブジェクトの getDatabaseInfo メソッドを使用して呼び出します。

メンバ

DatabaseInfo のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getCommitCount メソッド」 147 ページ
- 「getDbFormat メソッド」 147 ページ
- 「getLogSize メソッド」 148 ページ
- 「getNumberRowsToUpload メソッド」 148 ページ
- 「getPageReads メソッド」 148 ページ
- 「getPageSize メソッド」 148 ページ
- 「getPageWrites メソッド」 149 ページ
- 「getRelease メソッド」 149 ページ

getCommitCount メソッド

データベースに対して実行されたコミット操作の合計数を返します。

構文

int **DatabaseInfo.getCommitCount()**

戻り値

コミット操作の合計数。

getDbFormat メソッド

データベースのバージョン番号を返します。

構文

int **DatabaseInfo.getDbFormat()**

戻り値

バージョン番号。

getLogSize メソッド

トランザクション・ログの合計サイズ (バイト単位) を返します。

構文

int **DatabaseInfo.getLogSize()**

戻り値

トランザクション・ログのサイズ。

getNumberRowsToUpload メソッド

アップロードを待機しているローの数を返します。

構文

int **DatabaseInfo.getNumberRowsToUpload()**

戻り値

ローの数。

getPageReads メソッド

DatabaseInfo オブジェクトの作成時点のページ読み込み数を返します。

構文

int **DatabaseInfo.getPageReads()**

戻り値

ページ読み込み数。

getPageSize メソッド

データベースのページ・サイズ (バイト単位) を返します。

構文

int **DatabaseInfo.getPageSize()**

戻り値

ページ・サイズ。

getRelease メソッド

ソフトウェアのリリース番号を string として返します。

構文

String **DatabaseInfo.getRelease()**

備考

ソフトウェア・リリース値 7.1.3 は "7.1.3" として返されます。

戻り値

リリース番号。

getPageWrites メソッド

DatabaseInfo オブジェクトの作成時点のページ書き込み数を返します。

構文

int **DatabaseInfo.getPageWrites()**

戻り値

ページ書き込み数。

DatabaseManager クラス

基本設定を取得したり、新しいデータベースを作成したり、既存のデータベースに接続したりするための静的メソッドを提供します。

構文

public **DatabaseManager**

備考

次の例は、既存のデータベースを開く方法、またデータベースが存在しない場合は新規に作成する方法を示しています。

この例は J2ME デバイスで動作します。

```
Connection conn;
ConfigRecordStore config = DatabaseManager.createConfigurationRecordStore(
    "test.ulj"
    );
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

この例は J2ME BlackBerry デバイスで動作します。

```
Connection conn;
ConfigObjectStore config = DatabaseManager.createConfigurationObjectStore(
    "test.ulj"
);
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```

この例は J2SE デバイスで動作します。

```
Connection conn;
ConfigFile config = DatabaseManager.createConfigurationFile(
    "test.ulj"
);
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
```


メンバ

DatabaseManager のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

connect メソッド

設定に基づいて既存のデータベースに接続し、Connection を返します。

構文

```
Connection DatabaseManager.connect(
  Configuration config
) throws ULjException
```

パラメータ

- **config**　既存のデータベースの設定。

参照

- 「Configuration インタフェース」 123 ページ

戻り値

既存のデータベースへの Connection。

createConfigurationFile メソッド

ファイルを物理ストアとする設定を作成し、ConfigFile を返します。

構文

```
ConfigFile DatabaseManager.createConfigurationFile(
  String file_name
) throws ULjException
```

パラメータ

- **file_name**　使用または作成するファイルの名前。

参照

- 「ConfigFile インタフェース」 112 ページ

戻り値

データベースの設定に使用する ConfigFile。

createConfigurationNonPersistent メソッド

永続ストアなしの設定を作成し、ConfigNonPersist を返します。

構文

```
ConfigNonPersistent DatabaseManager.createConfigurationNonPersistent(
  String db_name
) throws ULjException
```

パラメータ

- **db_name**　データベースの名前。

参照

- 「ConfigNonPersistent インタフェース」 113 ページ

戻り値

データベースの設定に使用する ConfigNonPersistent。

createConfigurationObjectStore 関数 (J2ME BlackBerry のみ)

RIM オブジェクト・ストアを物理ストアとする設定を作成し、ConfigObjectStore を返します。

構文

```
ConfigObjectStore createConfigurationObjectStore(
Stringdb_name
)
```

パラメータ

- **db_name**　データベースの名前。

参照

- 「ConfigObjectStore インタフェース (J2ME BlackBerry のみ)」 114 ページ

戻り値

データベースの設定に使用する ConfigObjectStore。

createConfigurationRecordStore 関数 (J2ME のみ)

レコード・ストアを物理ストアとする設定を作成し、ConfigRecordStore を返します。

構文

```
ConfigRecordStore DatabaseManager.createConfigurationRecordStore(
  String db_name
)
```

パラメータ

- **db_name**　データベースの名前。

参照

- 「ConfigRecordStore インタフェース (J2ME のみ)」 122 ページ

戻り値

データベースの設定に使用する ConfigRecordStore。

createDatabase メソッド

設定に基づいて新しいデータベースを作成し、Connection を返します。

構文

```
Connection DatabaseManager.createDatabase(
  Configuration config
) throws ULjException
```

パラメータ

- **config**　新しいデータベースの設定。

備考

このメソッドは、同じ名前のデータベースを置換します。

参照

- 「Configuration インタフェース」 123 ページ

戻り値

新しいデータベースへの Connection。

createSISHTTPListener 関数 (J2ME BlackBerry のみ)

サーバ起動同期用の HTTP SISListener を作成します。

構文

SISListener **DatabaseManager.createSISHTTPListener(**
 SISRequestHandler *handler, int port*
 String *httpOptions*
)

パラメータ

- **handler**　SIS 要求を処理するために指定された SISRequestHandler。
- **port**　サーバ・メッセージを受信する HTTP ポート。
- **httpOptions**　サーバに接続するための HTTP オプション。

備考

推奨されるポート設定は 4400 です。

BlackBerry シミュレータに推奨される HTTP オプションは "deviceside=false" です。

参照

- 「SISListener インタフェース (J2ME BlackBerry のみ)」 188 ページ

戻り値

サーバ起動同期の SISListener。

release メソッド

DatabaseManager を閉じてすべての接続を解放し、すべてのデータベースを停止します。

構文

void **DatabaseManager.release()** throws **ULjException**

備考

コミットされていないトランザクションはロールバックされます。

setErrorLanguage メソッド

エラー・メッセージに使用する言語を設定します。

構文

void **DatabaseManager.setErrorLanguage(**
  String *lang*
)

パラメータ

- **lang**　2 文字の言語コード。

備考

認識される言語は EN、DE、FR、JA、ZH です。指定された言語を認識できなかった場合はデフォルトに戻ります。

J2SE と BlackBerry J2ME の各環境では、現在のロケールを使用してデフォルトの言語が決定されます。その他の J2ME 環境では、このメソッドが、言語を指定する唯一の方法です。デフォルトの言語は "EN" です。

DecimalNumber インタフェース

正確な decimal 値を表し、java.math.BigDecimal を使用できない Java プラットフォームに 10 進法計算のサポートを提供します。

構文

public **DecimalNumber**

メンバ

DecimalNumber のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「add メソッド」 156 ページ
- 「divide メソッド」 156 ページ
- 「getString メソッド」 157 ページ
- 「isNull メソッド」 157 ページ
- 「multiply メソッド」 157 ページ
- 「set メソッド」 158 ページ
- 「setNull メソッド」 158 ページ
- 「subtract メソッド」 158 ページ

add メソッド

2 つの DecimalNumber を加算し、その和を返します。

構文

```
DecimalNumber DecimalNumber.add(
  DecimalNumber num1,
  DecimalNumber num2
) throws ULjException
```

パラメータ

- **num1**　最初の数値。
- **num2**　2 番目の数値。

戻り値

num1 と num2 の和。

divide メソッド

最初の DecimalNumber を 2 番目の DecimalNumber で割ってその商を返します。

構文

DecimalNumber **DecimalNumber.divide(**
  DecimalNumber *num1*,
  DecimalNumber *num2*
) throws **ULjException**

パラメータ

- **num1**　被除数。
- **num2**　除数。

戻り値

num1 を num2 で割った商。

getString メソッド

DecimalNumber の String 表現を返します。

構文

String **DecimalNumber.getString()** throws **ULjException**

戻り値

String 値。

isNull メソッド

DecimalNumber が NULL かどうかを確認します。

構文

boolean **DecimalNumber.isNull()**

戻り値

オブジェクトが NULL の場合は true、NULL 以外の場合は false。

multiply メソッド

2 つの DecimalNumber を乗算し、その積を返します。

構文

DecimalNumber **DecimalNumber.multiply(**
  DecimalNumber *num1*,
  DecimalNumber *num2*
) throws **ULjException**

パラメータ

- **num1**　被乗数。
- **num2**　乗数。

戻り値

num1 と num2 の積。

set メソッド

DecimalNumber に String 値を設定します。

構文

void **DecimalNumber.set(**
  String *value*
) throws **ULjException**

パラメータ

- **value**　String として表した数値。

setNull メソッド

DecimalNumber を NULL に設定します。

構文

void **DecimalNumber.setNull()** throws **ULjException**

subtract メソッド

2 番目の DecimalNumber を最初の DecimalNumber から減算し、その差を返します。

構文

DecimalNumber **DecimalNumber.subtract(**
  DecimalNumber *num1*,
  DecimalNumber *num2*
) throws **ULjException**

パラメータ

- **num1**　被減数。
- **num2**　減数。

戻り値

num1 と num2 の差。

Domain インタフェース

テーブル内のカラムのドメインの型情報を表します。

構文

public **Domain**

備考

このインタフェースには、さまざまなドメインを表す定数と、Domain オブジェクトから情報を抽出するためのメソッドがあります。

次の例は、単純なデータベースのスキーマを作成する方法を示します。整数カラムと可変長の文字列カラム (長さは最大で 32 バイト) のある T2 テーブルが作成されます。

```
// Assumes a valid Connection object conn
TableSchema table_schema;
IndexSchema index_schema;

table_schema = conn.createTable("T2");
table_schema.createColumn("num", Domain.INTEGER);
table_schema.createColumn("name", Domain.VARCHAR, 32);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("num", IndexSchema.ASCENDING);
```

整数型

ドメイン定数	SQL 型	値の範囲
BIT	BIT	0 または 1
TINY	TINYINT	0 ～ 255 (1 バイトの記憶領域を使用する符号なし整数)
SHORT	SMALLINT	-32768 ～ 32767 (2 バイトの記憶領域を使用する符号付き整数)
UNSIGNED_SHORT	UNSIGNED SMALLINT	0 ～ 65535 (2 バイトの記憶領域を使用する符号なし整数)
INTEGER	INTEGER	-231 ～ 231 - 1、または -2147483648 ～ 2147483647 (4 バイトの記憶領域を使用する符号付き整数)
UNSIGNED_INTEGER	UNSIGNED INTEGER	0 ～ 232 - 1、または 0 ～ 4294967295 (4 バイトの記憶領域を使用する符号なし整数)
BIG	BIGINT	-263 ～ 263 - 1、または -9223372036854775808 ～ 9223372036854775807 (8 バイトの記憶領域を使用する符号付き整数)

ドメイン定数	SQL 型	値の範囲
UNSIGNED_BIG	UNSIGNED BIGINT	0 ～ 264 - 1、または 0 ～ 18446744073709551615 (8 バイトの記憶領域を使用する符号なし整数)

整数以外の数値型

ドメイン定数	SQL 型	値の範囲
REAL	REAL	-3.402823e+38 ～ 3.402823e+38、0 に最も近い最小の数値は 1.175495e-38 (4 バイトの記憶領域を使用する単精度の浮動小数点数、6 桁目の後に丸め誤差が生じる可能性があります)
DOUBLE	DOUBLE	-1.79769313486231e+308 ～ 1.79769313486231e+308、0 に最も近い最小の数値は 2.22507385850721e-308 (8 バイトの記憶領域を使用する単精度の浮動小数点数、15 桁目の後に丸め誤差が生じる可能性があります)
NUMERIC	NUMERIC (precision, scale)	合計桁数が precision (サイズ)、小数点以下の桁数が scale 桁の任意の 10 進数 (precision 内の丸めなし)

文字型とバイナリ型

ドメイン定数	SQL 型	サイズの範囲
VARCHAR	VARCHAR(size)	1 ～ 32767 文字 (1 ～ 3 バイトの UTF-8 文字として格納)。式を評価するときのテンポラリ文字値の最大長は 2048 文字です。
LONGVARCHAR	LONG VARCHAR	任意の長さ (メモリで許容される範囲内)。LONG VARCHAR カラムで実行可能な演算は、これらの挿入、更新、削除、またはクエリの select リストへのこれらの指定のみです。
BINARY	BINARY(size)	1 ～ 32767 バイト。式を評価するときのテンポラリ文字値の最大長は 2048 バイトです。
LONGBINARY	LONG BINARY	任意の長さ (メモリで許容される範囲内)。LONG BINARY カラムで実行可能な演算は、これらの挿入、更新、削除、またはクエリの select リストへのこれらの指定のみです。
UUID	UNIQUEIDENTIFIER	常に 16 バイトの解釈が特殊なバイナリ

日付型と時間型

ドメイン定数	SQL 型	値
DATE	DATE	年、月、日。
TIME	TIME	時、分、秒 (小数位あり) で構成される時刻。
TIMESTAMP	TIMESTAMP	DATE と TIME。

BIT カラムはデフォルトでは NULL 入力不可です。その他の型はデフォルトで NULL 入力可です。

メンバ

Domain のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

BIG 変数

64 ビット整数 (SQL 型 BIGINT) のドメイン ID 定数です。

構文

final short **Domain.BIG**

参照

● 「Domain インタフェース」 159 ページ

BINARY 変数

最大 **size** バイトの可変長のバイナリ・オブジェクト (SQL 型 BINARY(**size**)) のドメイン ID 定数です。

構文

final short **Domain.BINARY**

参照

● 「Domain インタフェース」 159 ページ

BINARY_DEFAULT 変数

バイナリ型のデフォルト・サイズです。

構文

final short **Domain.BINARY_DEFAULT**

BINARY_MAX 変数

バイナリ型の最大サイズです。

構文

final short **Domain.BINARY_MAX**

BINARY_MIN 変数

バイナリ型の最小サイズです。

構文

final short **Domain.BINARY_MIN**

BIT 変数

ビット (SQL 型 BIT) のドメイン ID 定数です。

構文

final short **Domain.BIT**

備考

BIT カラムはデフォルトでは NULL 入力不可です。

参照

- 「Domain インタフェース」 159 ページ

CHARACTER_MAX 変数

文字型の最大サイズです。

構文

final short **Domain.CHARACTER_MAX**

DATE 変数

日付 (SQL 型 DATE) のドメイン ID 定数です。

構文

final short **Domain.DATE**

参照

- 「Domain インタフェース」 159 ページ

DOMAIN_MAX 変数

Domain 型の最大数です。

構文

final short **Domain.DOMAIN_MAX**

DOUBLE 変数

8 バイトの浮動小数点数 (SQL 型 DOUBLE) のドメイン ID 定数です。

構文

final short **Domain.DOUBLE**

参照

- 「Domain インタフェース」 159 ページ

INTEGER 変数

32 ビット整数 (SQL 型 INTEGER) のドメイン ID 定数です。

構文

final short **Domain.INTEGER**

参照

- 「Domain インタフェース」 159 ページ

LONGBINARY 変数

任意の長いブロックのバイナリ・データ (BLOB) (SQL 型 LONG BINARY) のドメイン ID 定数です。

構文

final short **Domain.LONGBINARY**

参照

- 「Domain インタフェース」 159 ページ

LONGBINARY_DEFAULT 変数

BLOB 型のデフォルト・サイズです。

構文

final short **Domain.LONGBINARY_DEFAULT**

LONGBINARY_MIN 変数

BLOB 型の最小サイズです。

構文

final short **Domain.LONGBINARY_MIN**

LONGVARCHAR 変数

任意の長いブロックの文字データ (CLOB) (SQL 型 LONG VARCHAR) のドメイン ID 定数です。

構文

final short **Domain.LONGVARCHAR**

参照

- 「Domain インタフェース」 159 ページ

LONGVARCHAR_DEFAULT 変数

CLOB 型のデフォルト・サイズです。

構文

final short **Domain.LONGVARCHAR_DEFAULT**

LONGVARCHAR_MIN 変数

CLOB 型の最小サイズです。

構文

final short **Domain.LONGVARCHAR_MIN**

NUMERIC 変数

合計桁数が固定 precision (サイズ)、小数点以下の桁数が **scale** 桁の数値 (SQL 型 NUMERIC(**precision,scale**)) のドメイン ID 定数です。

構文

final short **Domain.NUMERIC**

参照

- 「Domain インタフェース」 159 ページ

PRECISION_DEFAULT 変数

数値の精度のデフォルト・サイズです。

構文

final short **Domain.PRECISION_DEFAULT**

PRECISION_MAX 変数

数値の精度の最大サイズです。

構文

final short **Domain.PRECISION_MAX**

PRECISION_MIN 変数

数値の精度の最小サイズです。

構文

final short **Domain.PRECISION_MIN**

REAL 変数

4 バイトの浮動小数点数 (SQL 型 REAL) のドメイン ID 定数です。

構文

final short **Domain.REAL**

参照

- 「Domain インタフェース」 159 ページ

SCALE_DEFAULT 変数

数値の位取りのデフォルト・サイズです。

構文

final short **Domain.SCALE_DEFAULT**

SCALE_MAX 変数

数値の位取りの最大サイズです。

構文

final short **Domain.SCALE_MAX**

SCALE_MIN 変数

数値の位取りの最小サイズです。

構文

final short **Domain.SCALE_MIN**

SHORT 変数

16 ビット整数 (SQL 型 SMALLINT) のドメイン ID 定数です。

構文

final short **Domain.SHORT**

参照

- 「Domain インタフェース」 159 ページ

TIME 変数

時間 (SQL 型 TIME) のドメイン ID 定数です。

構文

final short **Domain.TIME**

参照

- 「Domain インタフェース」 159 ページ

TIMESTAMP 変数

タイムスタンプ (SQL 型 TIMESTAMP) のドメイン ID 定数です。

構文

final short **Domain.TIMESTAMP**

参照

- 「Domain インタフェース」 159 ページ

TINY 変数

符号なし 8 ビット整数 (SQL 型 TINYINT) のドメイン ID 定数です。

構文

final short **Domain.TINY**

参照

- 「Domain インタフェース」 159 ページ

UINT16_MAX 変数

符号なし 16 ビット整数の最大サイズです。

構文

final int **Domain.UINT16_MAX**

UNSIGNED_BIG 変数

符号なし 64 ビット整数 (SQL 型 UNSIGNED BIGINT) のドメイン ID 定数です。

構文

final short **Domain.UNSIGNED_BIG**

参照

- 「Domain インタフェース」 159 ページ

UNSIGNED_INTEGER 変数

符号なし 32 ビット整数 (SQL 型 UNSIGNED INTEGER) のドメイン ID 定数です。

構文

final short **Domain.UNSIGNED_INTEGER**

参照

- 「Domain インタフェース」 159 ページ

UNSIGNED_SHORT 変数

符号なし 16 ビット整数 (SQL 型 UNSIGNED SMALLINT) のドメイン ID 定数です。

構文

final short **Domain.UNSIGNED_SHORT**

参照

- 「Domain インタフェース」 159 ページ

UUID 変数

UniqueIdentifier (SQL 型 UNIQUEIDENTIFIER) のドメイン ID 定数です。

構文

final short **Domain.UUID**

参照

- 「Domain インタフェース」 159 ページ

VARCHAR 変数

最大 **size** バイトの可変長の文字列 (SQL 型 VARCHAR(**size**)) のドメイン ID 定数です。

構文

final short **Domain.VARCHAR**

参照

- 「Domain インタフェース」 159 ページ

VARCHAR_DEFAULT 変数

文字型のデフォルト・サイズです。

構文

final short **Domain.VARCHAR_DEFAULT**

VARCHAR_MIN 変数

文字型の最小サイズです。

構文

final short **Domain.VARCHAR_MIN**

getName メソッド

ドメインの名前を返します。

構文

String **Domain.getName()**

戻り値

ドメイン名。

getPrecision メソッド

ドメイン値の精度を返します。

構文

int **Domain.getPrecision()**

戻り値

値の精度。

getScale メソッド

ドメイン値の位取りを返します。

構文

int **Domain.getScale()**

戻り値

値の位取り。

getSize メソッド

ドメイン値のサイズを返します。

構文

short **Domain.getSize()**

戻り値

値のサイズ。

getType メソッド

ドメインの型を返します。

構文

short **Domain.getType()**

戻り値

整数で表したドメインの型。

EncryptionControl インタフェース

データベースの暗号化を制御します。

構文

public **EncryptionControl**

備考

このインタフェースは、独自の暗号化または難読化の手法を実装するために使用します。データベースを暗号化するには、EncryptionControl を実装する新しいクラスを作成し、このクラスで独自の暗号化手法を指定し、ConfigPersistent インタフェースの setEncryption メソッドを使用して EncryptionControl クラスの新しいインスタンスを開始します。

参照

- 「setEncryption メソッド」 117 ページ

メンバ

EncryptionControl のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「decrypt メソッド」 173 ページ
- 「encrypt メソッド」 174 ページ
- 「initialize メソッド」 174 ページ

decrypt メソッド

データベース内の byte 配列を復号化します。

構文

```
void EncryptionControl.decrypt(
  int page_no,
  byte[] src,
  byte[] tgt
) throws ULjException
```

パラメータ

- **page_no**　配列データのページ番号。
- **src**　暗号化されているソース・ページ。
- **tgt**　メソッドで復号化されたページ。

備考

このメソッドには、暗号化された byte 配列、ソース、関連するページ番号を指定します。メソッドでは、src を復号化し、その結果を tgt byte 配列に格納します。その後、アプリケーション内のデータ操作に tgt を使用します。

encrypt メソッド

データベース内の byte 配列を暗号化します。

構文

```
void EncryptionControl.encrypt(
  int page_no,
  byte[] src,
  byte[] tgt
) throws ULjException
```

パラメータ

- **page_no**　配列データのページ番号。
- **src**　復号化されているソース・ページ。
- **tgt**　メソッドで暗号化されたページ。

備考

このメソッドには、暗号化されていない byte 配列、ソース、関連するページ番号を指定します。メソッドでは、src を暗号化または難読化し、その結果を tgt byte 配列に格納します。その後 tgt をデータベースに格納します。

initialize メソッド

パスワードを使用して暗号化制御を初期化します。

構文

```
void EncryptionControl.initialize(
  String password
) throws ULjException
```

パラメータ

- **password**　暗号化と復号化に使用するパスワード。

ForeignKeySchema インタフェース

外部キーのスキーマを指定します。

構文

public **ForeignKeySchema**

備考

このインタフェースをサポートするオブジェクトは Connection.createForeignKey(String) メソッドから返されます。

外部キーには 1 つ以上のカラム参照が必要です。参照先カラムのセットはプライマリ・テーブルのカラムである必要があり、またこのセットはプライマリ・テーブルに対するプライマリ・キーまたはユニーク・キーの制約の対象である必要があります。

次の例は、単純なデータベースのスキーマを作成する方法を示しています。Invoices テーブルには Products テーブルへの外部キーがあり、すべての送り状で有効な製品 ID を参照する必要があることを指定しています。

```
TableSchema table_schema;
IndexSchema index_schema;
ForeignKeySchema fkey_schema;

table_schema = conn.createTable("Invoices");
table_schema.createColumn("inv_id", Domain.INTEGER);
table_schema.createColumn("quantity", Domain.INTEGER);
table_schema.createColumn("sold_prod_id", Domain.INTEGER);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("inv_id", IndexSchema.ASCENDING);

table_schema = conn.createTable("Products");
table_schema.createColumn("prod_id", Domain.INTEGER);
table_schema.createColumn("prod_name", Domain.VARCHAR, 40);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("prod_id", IndexSchema.ASCENDING);

fkey_schema = conn.createForeignKey(
   "Invoices", "Products", "InvoiceToProduct" );
fkey_schema.addColumnReference("sold_prod_is", "prod_id");

conn.schemaCreateComplete();
```

メンバ

ForeignKeySchema のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「addColumnReference メソッド」 175 ページ

addColumnReference メソッド

外部キーにカラム参照を追加します。

構文

ForeignKeySchema **ForeignKeySchema.addColumnReference(**
  String *foreign_column*,
  String *primary_column*
) throws **ULjException**

パラメータ

- **foreign_column**　外部キーが含まれるカラムの名前。このカラムの値を使用して、プライマリ・テーブルの primary_column_name カラムの値が参照されます。
- **primary_column**　参照先テーブル内のカラムの名前。すべてのプライマリ・カラムが、プライマリ・テーブルのプライマリ・キーまたはユニーク・キーの制約にセットとして含まれる必要があります。

戻り値

外部キーが外部カラムに割り当てられた ForeignKeySchema。

IndexSchema インタフェース

インデックスのスキーマを指定し、システム・テーブルの問い合わせに便利な定数を提供します。

構文

public **IndexSchema**

備考

このインタフェースをサポートするオブジェクトは、TableSchema.createIndex(String)、TableSchema.createPrimaryIndex(String)、TableSchema.createUniqueIndex(String)、TableSchema.createUniqueKey(String) の各メソッドから返されます。各種のインデックス・タイプの説明については、「TableSchema インタフェース」 245 ページを参照してください。

インデックスには 1 つ以上のカラムが必要です。

インデックスは、インデックスに追加された最初のカラムでソートされ、次に 2 番目のカラム (指定されている場合) でソートされ、その後同様に続きます。

インデックスに LONGBINARY 型または LONGVARCHAR 型のカラムを含めることはできません。

次の例は、2 カラムのインデックスを作成する方法を示しています。

```
// Assumes a valid TableSchema object table_schema on
// a table with columns A and B.
IndexSchema index_schema;
index_schema = table_schema.createIndex("AthenBreversed");
index_schema.addColumn("A", IndexSchema.ASCENDING);
index_schema.addColumn("B", IndexSchema.DESCENDING);
```

メンバ

IndexSchema のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「addColumn メソッド」 179 ページ
- 「ASCENDING 変数」 177 ページ
- 「DESCENDING 変数」 178 ページ
- 「PERSISTENT 変数」 178 ページ
- 「PRIMARY_INDEX 変数」 178 ページ
- 「UNIQUE_INDEX 変数」 178 ページ
- 「UNIQUE_KEY 変数」 179 ページ

ASCENDING 変数

カラムのインデックスが昇順でソートされています。

構文

final byte **IndexSchema.ASCENDING**

DESCENDING 変数

カラムのインデックスが降順でソートされています。

構文

final byte **IndexSchema.DESCENDING**

PERSISTENT 変数

インデックスが永続的であることを示すビット・フラグです。

構文

final byte **IndexSchema.PERSISTENT**

備考

この値は、テーブル SYS_INDEXES の index_flags カラムで他のフラグと論理的に組み合わせることができます。

PRIMARY_INDEX 変数

インデックスがプライマリ・キーであることを示すビット・フラグです。

構文

final byte **IndexSchema.PRIMARY_INDEX**

備考

この値は、テーブル SYS_INDEXES の index_flags カラムで他のフラグと論理的に組み合わせることができます。

UNIQUE_INDEX 変数

インデックスがユニーク・インデックスであることを示すビット・フラグです。

構文

final byte **IndexSchema.UNIQUE_INDEX**

備考

この値は、テーブル SYS_INDEXES の index_flags カラムで他のフラグと論理的に組み合わせることができます。

UNIQUE_KEY 変数

インデックスがユニーク・キーであることを示すビット・フラグです。

構文

final byte **IndexSchema.UNIQUE_KEY**

備考

この値は、テーブル SYS_INDEXES の index_flags カラムで他のフラグと論理的に組み合わせることができます。

addColumn メソッド

インデックスにカラムを追加します。

構文

```
IndexSchema IndexSchema.addColumn(
  String column_name,
  byte sort_order
) throws ULjException
```

パラメータ

- **column_name**　追加するカラムの名前。指定するカラムは、このインデックスを作成しているテーブルのカラムである必要があります。
- **sort_order**　ソート順を指定する定数。IndexSchema.ASCENDING または IndexSchema.DESCENDING にします。

備考

カラムがインデックスに追加される順序によってソートの優先順位が決まります。最初のカラムの優先順位が最も高くなります。

戻り値

カラムが追加された IndexSchema。

PreparedStatement インタフェース

SQL クエリを実行して ResultSet を生成するか、準備された SQL 文を Ultra Light データベースに対して実行するメソッドを提供します。

構文

public **PreparedStatement**

基本クラス

- 「CollectionOfValueWriters インタフェース」 100 ページ

備考

次の例は、PreparedStatement を実行し、ResultSet が作成されたかどうかを確認し、ResultSet をローカル変数に格納し、PreparedStatement を閉じる方法を示しています。

```
// Create a new PreparedStatement object from an existing connection.
String sql_string = "SELECT * FROM SampleTable";
PreparedStatement ps = conn.prepareStatement(sql_string);

// result returns true if the execute statement runs successfully.
boolean result = ps.execute();

// Check if the PreparedStatement contains a ResultSet.
if (ps.hasResultSet()) {
    // Store the ResultSet in the rs variable.
    ResultSet rs = ps.getResultSet();
}

// Close the PreparedStatement to release resources.
ps.close();
```


メンバ

PreparedStatement のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

close メソッド

PreparedStatement を閉じて、関連付けられているメモリ・リソースを解放します。

構文

void **PreparedStatement.close()** throws **ULjException**

備考

このオブジェクトに対してこれ以上メソッドを使用できなくなります。PreparedStatement に ResultSet が含まれる場合は、ResultSet も閉じられます。

execute メソッド

準備された SQL 文を実行します。

構文

boolean **PreparedStatement.execute()** throws **ULjException**

参照

- 「ResultSet インタフェース」 184 ページ

戻り値

文が正常に実行された場合は true、それ以外の場合は false。

executeQuery メソッド

準備された SQL SELECT 文を実行し、ResultSet を返します。

構文

ResultSet **PreparedStatement.executeQuery()** throws **ULjException**

参照

- 「ResultSet インタフェース」 184 ページ

戻り値

準備された SQL SELECT 文のクエリの結果を含む ResultSet。

getPlan メソッド

SQL クエリ実行プランのテキストベースの記述を返します。

構文

String **PreparedStatement.getPlan()** throws **ULjException**

備考

プランがない場合は、空の文字列を返します。

戻り値

プランの String 表現。

getResultSet メソッド

準備された SQL 文の ResultSet を返します。

構文

ResultSet **PreparedStatement.getResultSet()** throws **ULjException**

参照

- 「ResultSet インタフェース」 184 ページ

戻り値

準備された SQL 文のクエリの結果を含む ResultSet。

getUpdateCount メソッド

最後のコミット文の後に挿入、更新、または削除されたロー数を返します。

構文

int **PreparedStatement.getUpdateCount()** throws **ULjException**

戻り値

変更がなかった場合は -1、それ以外の場合は更新されたローの数。

hasResultSet メソッド

PreparedStatement に ResultSet が含まれるかどうかを確認します。

構文

boolean **PreparedStatement.hasResultSet()** throws **ULjException**

参照

- 「ResultSet インタフェース」 184 ページ

戻り値

ResultSet が見つかった場合は true、それ以外の場合は false。

ResultSet インタフェース

テーブルをローごとにトラバースし、カラム・データにアクセスするメソッドを提供します。

構文

public **ResultSet**

基本クラス

- 「CollectionOfValueReaders インタフェース」 93 ページ

備考

execute メソッドまたは executeQuery メソッドを使用して、SQL SELECT 文を含む PreparedStatement を実行することで ResultSet を生成します。

次の例は、新しい PreparedStatement を実行し、ResultSet でローをフェッチし、指定されたカラムのデータにアクセスする方法を示しています。

```
// Define a new SQL SELECT statement.
String sql_string = "SELECT column1, column2 FROM SampleTable";

// Create a new PreparedStatement from an existing connection.
PreparedStatement ps = conn.prepareStatement(sql_string);

// Create a new ResultSet to contain the query results of the SQL statement.
ResultSet rs = ps.executeQuery();

// Check if the PreparedStatement contains a ResultSet.
if (ps.hasResultSet()) {
    // Retrieve the column values from the first row using getString.
    while (rs.next()) {
        c1 = rs.getString(1);
        c2 = rs.getString(2);
        ...
    }
}
```


メンバ

ResultSet のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「close メソッド」 185 ページ
- 「getBlobInputStream メソッド」 94 ページ
- 「getBoolean メソッド」 94 ページ
- 「getBytes メソッド」 95 ページ
- 「getClobReader メソッド」 95 ページ
- 「getDate メソッド」 95 ページ
- 「getDecimalNumber メソッド」 96 ページ
- 「getDouble メソッド」 96 ページ
- 「getFloat メソッド」 96 ページ
- 「getInt メソッド」 97 ページ
- 「getLong メソッド」 97 ページ
- 「getOrdinal メソッド」 98 ページ
- 「getResultSetMetadata メソッド」 185 ページ
- 「getString メソッド」 98 ページ
- 「getValue メソッド」 98 ページ
- 「isNull メソッド」 99 ページ
- 「next メソッド」 186 ページ
- 「previous メソッド」 186 ページ

close メソッド

ResultSet を閉じて、関連付けられているメモリ・リソースを解放します。

構文

void **ResultSet.close()** throws **ULjException**

備考

この後にこの ResultSet のローをフェッチしようとするとエラーが発生します。

getResultSetMetadata メソッド

ResultSet のメタデータを含む ResultSetMetadata を返します。

構文

ResultSetMetadata **ResultSet.getResultSetMetadata()** throws **ULjException**

戻り値

ResultSetMetadata オブジェクト。

next メソッド

ResultSet 内の次のデータ・ローをフェッチします。

構文

boolean **ResultSet.next()** throws **ULjException**

参照

- 「ResultSetMetadata インタフェース」 187 ページ

戻り値

次のローが正常にフェッチされた場合は true、それ以外の場合は false。

previous メソッド

ResultSet 内の前のデータ・ローをフェッチします。

構文

boolean **ResultSet.previous()** throws **ULjException**

戻り値

前のローが正常にフェッチされた場合は true、それ以外の場合は false。

ResultSetMetadata インタフェース

ResultSet オブジェクトに関連付けられ、カラム情報を提供するメソッドが含まれます。

構文

public **ResultSetMetadata**

備考

このインタフェースは、ResultSet オブジェクトの getResultSetMetadata メソッドを使用して呼び出します。

メンバ

ResultSetMetadata のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getColumnCount メソッド」 187 ページ

getColumnCount メソッド

ResultSet 内のカラムの合計数を返します。

構文

int **ResultSetMetadata.getColumnCount()** throws **ULjException**

戻り値

カラムの数。

SISListener インタフェース (J2ME BlackBerry のみ)

サーバ起動同期のメッセージを受信します。

構文

public **SISListener**

備考

アプリケーションでは、DatabaseManager インタフェースの適切な createSISHTTPListener メソッドを使用して SISListener のインスタンスを作成します。

メンバ

SISListener のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「startListening メソッド」 188 ページ
- 「stopListening メソッド」 188 ページ
- 「createSISHTTPListener 関数 (J2ME BlackBerry のみ)」 153 ページ

startListening メソッド

受信スレッドを作成し、開始します。

構文

void **SISListener.startListening()**

stopListening メソッド

受信スレッドを停止します。

構文

void **SISListener.stopListening()**

SISRequestHandler インタフェース (J2ME BlackBerry のみ)

サーバ起動同期の要求を処理します。

構文

public **SISRequestHandler**

メンバ

SISRequestHandler のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「onError メソッド」 189 ページ
- 「onRequest メソッド」 189 ページ

onError メソッド

ワーカ・スレッドに対する SIS 要求を処理します。

構文

void **SISListener.onError(**
  String *text*
)

パラメータ

- **text**　要求によって送信された文字列。

onRequest メソッド

SIS 受信中に発生する SIS 関連のエラーを処理します。

構文

void **SISListener.onRequest(**
  String *text*
)

パラメータ

- **text**　例外の string 表現。

備考

受信を停止するには、SISListener インタフェースの stopListening メソッドを明示的に呼び出します。

SQLCode インタフェース

Ultra Light J によってレポートされる可能性のある SQL コードを列挙します。

構文

public **SQLCode**

派生クラス

- 「ULjException クラス」 253 ページ

備考

各エラーの詳細については、SQL Anywhere のマニュアル・セットの「SQL Anywhere のエラー・メッセージ (SQLCODE 順)」を参照してください。

メンバ

SQLCode のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

SQLE_AGGREGATES_NOT_ALLOWED 変数

SQLE_AGGREGATES_NOT_ALLOWED(-150)。

構文

final int **SQLCode.SQLE_AGGREGATES_NOT_ALLOWED**

SQLE_ALIAS_NOT_UNIQUE 変数

SQLE_ALIAS_NOT_UNIQUE(-830)。

構文

final int **SQLCode.SQLE_ALIAS_NOT_UNIQUE**

SQLE_ALIAS_NOT_YET_DEFINED 変数

SQLE_ALIAS_NOT_YET_DEFINED(-831)。

構文

final int **SQLCode.SQLE_ALIAS_NOT_YET_DEFINED**

SQLE_AUTHENTICATION_FAILED 変数

SQLE_AUTHENTICATION_FAILED(-218)。

構文

final int **SQLCode.SQLE_AUTHENTICATION_FAILED**

SQLE_CANNOT_EXECUTE_STMT 変数

SQLE_CANNOT_EXECUTE_STMT(111)。

構文

final int **SQLCode.SQLE_CANNOT_EXECUTE_STMT**

SQLE_CLIENT_OUT_OF_MEMORY 変数

SQLE_CLIENT_OUT_OF_MEMORY(-876)。

構文

final int **SQLCode.SQLE_CLIENT_OUT_OF_MEMORY**

SQLE_COLUMN_AMBIGUOUS 変数

SQLE_COLUMN_AMBIGUOUS(-144)。

構文

final int **SQLCode.SQLE_COLUMN_AMBIGUOUS**

SQLE_COLUMN_CANNOT_BE_NULL 変数

SQLE_COLUMN_CANNOT_BE_NULL(-195)。

構文

final int **SQLCode.SQLE_COLUMN_CANNOT_BE_NULL**

SQLE_COLUMN_NOT_FOUND 変数

SQLE_COLUMN_NOT_FOUND(-143)。

構文

final int **SQLCode.SQLE_COLUMN_NOT_FOUND**

SQLE_COLUMN_NOT_STREAMABLE 変数

SQLE_COLUMN_NOT_STREAMABLE(-1100)。

構文

final int **SQLCode.SQLE_COLUMN_NOT_STREAMABLE**

SQLE_COMMUNICATIONS_ERROR 変数

SQLE_COMMUNICATIONS_ERROR(-85)。

構文

final int **SQLCode.SQLE_COMMUNICATIONS_ERROR**

SQLE_CONFIG_IN_USE 変数

SQLE_CONFIG_IN_USE(-1276)。

構文

final int **SQLCode.SQLE_CONFIG_IN_USE**

SQLE_CONVERSION_ERROR 変数

SQLE_CONVERSION_ERROR(-157)。

構文

final int **SQLCode.SQLE_CONVERSION_ERROR**

SQLE_CURSOR_ALREADY_OPEN 変数

SQLE_CURSOR_ALREADY_OPEN(-172)。

構文

final int **SQLCode.SQLE_CURSOR_ALREADY_OPEN**

SQLE_DATABASE_ACTIVE 変数

SQLE_DATABASE_ACTIVE(-664)。

構文

final int **SQLCode.SQLE_DATABASE_ACTIVE**

SQLE_DEVICE_IO_FAILED 変数

SQLE_DEVICE_IO_FAILED(-974)。

構文

final int **SQLCode.SQLE_DEVICE_IO_FAILED**

SQLE_DIV_ZERO_ERROR 変数

SQLE_DIV_ZERO_ERROR(-628)。

構文

final int **SQLCode.SQLE_DIV_ZERO_ERROR**

SQLE_DOWNLOAD_CONFLICT 変数

SQLE_DOWNLOAD_CONFLICT(-839)。

構文

final int **SQLCode.SQLE_DOWNLOAD_CONFLICT**

SQLE_ERROR 変数

SQLE_ERROR(-300)。

構文

final int **SQLCode.SQLE_ERROR**

SQLE_EXISTING_PRIMARY_KEY 変数

SQLE_EXISTING_PRIMARY_KEY(-112)。

構文

final int **SQLCode.SQLE_EXISTING_PRIMARY_KEY**

SQLE_EXPRESSION_ERROR 変数

SQLE_EXPRESSION_ERROR(-156)。

構文

final int **SQLCode.SQLE_EXPRESSION_ERROR**

SQLE_FILE_BAD_DB 変数

SQLE_FILE_BAD_DB(-1006)。

構文

final int **SQLCode.SQLE_FILE_BAD_DB**

SQLE_FILE_WRONG_VERSION 変数

SQLE_FILE_WRONG_VERSION(-1005)。

構文

final int **SQLCode.SQLE_FILE_WRONG_VERSION**

SQLE_FOREIGN_KEY_NAME_NOT_FOUND 変数

SQLE_FOREIGN_KEY_NAME_NOT_FOUND(-145)。

構文

final int **SQLCode.SQLE_FOREIGN_KEY_NAME_NOT_FOUND**

SQLE_IDENTIFIER_TOO_LONG 変数

SQLE_IDENTIFIER_TOO_LONG(-250)。

構文

final int **SQLCode.SQLE_IDENTIFIER_TOO_LONG**

SQLE_INCOMPLETE_SYNCHRONIZATION 変数

SQLE_INCOMPLETE_SYNCHRONIZATION(-1271)。

構文

final int **SQLCode.SQLE_INCOMPLETE_SYNCHRONIZATION**

SQLE_INDEX_HAS_NO_COLUMNS 変数

SQLE_INDEX_HAS_NO_COLUMNS(-1274)。

構文

final int **SQLCode.SQLE_INDEX_HAS_NO_COLUMNS**

SQLE_INDEX_NOT_FOUND 変数

SQLE_INDEX_NOT_FOUND(-183)。

構文

final int **SQLCode.SQLE_INDEX_NOT_FOUND**

SQLE_INDEX_NOT_UNIQUE 変数

SQLE_INDEX_NOT_UNIQUE(-196)。

構文

final int **SQLCode.SQLE_INDEX_NOT_UNIQUE**

SQLE_INTERRUPTED 変数

SQLE_INTERRUPTED(-299)。

構文

final int **SQLCode.SQLE_INTERRUPTED**

SQLE_INVALID_COMPARISON 変数

SQLE_INVALID_COMPARISON(-710)。

構文

final int **SQLCode.SQLE_INVALID_COMPARISON**

SQLE_INVALID_DISTINCT_AGGREGATE 変数

SQLE_INVALID_DISTINCT_AGGREGATE(-863)。

構文

final int **SQLCode.SQLE_INVALID_DISTINCT_AGGREGATE**

SQLE_INVALID_DOMAIN 変数

SQLE_INVALID_DOMAIN(-1275)。

構文

final int **SQLCode.SQLE_INVALID_DOMAIN**

SQLE_INVALID_FOREIGN_KEY_DEF 変数

SQLE_INVALID_FOREIGN_KEY_DEF(-113)。

構文

final int **SQLCode.SQLE_INVALID_FOREIGN_KEY_DEF**

SQLE_INVALID_GROUP_SELECT 変数

SQLE_INVALID_GROUP_SELECT(-149)。

構文

final int **SQLCode.SQLE_INVALID_GROUP_SELECT**

SQLE_INVALID_INDEX_TYPE 変数

SQLE_INVALID_INDEX_TYPE(-650)。

構文

final int **SQLCode.SQLE_INVALID_INDEX_TYPE**

SQLE_INVALID_LOGON 変数

SQLE_INVALID_LOGON(-103)。

構文

final int **SQLCode.SQLE_INVALID_LOGON**

SQLE_INVALID_OPTION 変数

SQLE_INVALID_OPTION(-200)。

構文

final int **SQLCode.SQLE_INVALID_OPTION**

SQLE_INVALID_OPTION_SETTING 変数

SQLE_INVALID_OPTION_SETTING(-201)。

構文

final int **SQLCode.SQLE_INVALID_OPTION_SETTING**

SQLE_INVALID_ORDER 変数

SQLE_INVALID_ORDER(-152)。

構文

final int **SQLCode.SQLE_INVALID_ORDER**

SQLE_INVALID_PARAMETER 変数

SQLE_INVALID_PARAMETER(-735)。

構文

final int **SQLCode.SQLE_INVALID_PARAMETER**

SQLE_INVALID_UNION 変数

SQLE_INVALID_UNION(-153)。

構文

final int **SQLCode.SQLE_INVALID_UNION**

SQLE_LOCKED 変数

SQLE_LOCKED(-210)。

構文

final int **SQLCode.SQLE_LOCKED**

SQLE_MAX_ROW_SIZE_EXCEEDED 変数

SQLE_MAX_ROW_SIZE_EXCEEDED(-1132)。

構文

final int **SQLCode.SQLE_MAX_ROW_SIZE_EXCEEDED**

SQLE_MUST_BE_ONLY_CONNECTION 変数

SQLE_MUST_BE_ONLY_CONNECTION(-211)。

構文

final int **SQLCode.SQLE_MUST_BE_ONLY_CONNECTION**

SQLE_NAME_NOT_UNIQUE 変数

SQLE_NAME_NOT_UNIQUE(-110)。

構文

final int **SQLCode.SQLE_NAME_NOT_UNIQUE**

SQLE_NOERROR 変数

SQLE_NOERROR(0)。

構文

final int **SQLCode.SQLE_NOERROR**

SQLE_NOT_IMPLEMENTED 変数

SQLE_NOT_IMPLEMENTED(-134)。

構文

final int **SQLCode.SQLE_NOT_IMPLEMENTED**

SQLE_NO_COLUMN_NAME 変数

SQLE_NO_COLUMN_NAME(-163)。

構文

final int **SQLCode.SQLE_NO_COLUMN_NAME**

SQLE_NO_CURRENT_ROW 変数

SQLE_NO_CURRENT_ROW(-197)。

構文

final int **SQLCode.SQLE_NO_CURRENT_ROW**

SQLE_NO_MATCHING_SELECT_ITEM 変数

SQLE_NO_MATCHING_SELECT_ITEM(-812)。

構文

final int **SQLCode.SQLE_NO_MATCHING_SELECT_ITEM**

SQLE_NO_PRIMARY_KEY 変数

SQLE_NO_PRIMARY_KEY(-118)。

構文

final int **SQLCode.SQLE_NO_PRIMARY_KEY**

SQLE_OVERFLOW_ERROR 変数

SQLE_OVERFLOW_ERROR(-158)。

構文

final int **SQLCode.SQLE_OVERFLOW_ERROR**

SQLE_PAGE_SIZE_TOO_BIG 変数

SQLE_PAGE_SIZE_TOO_BIG(-97)。

構文

final int **SQLCode.SQLE_PAGE_SIZE_TOO_BIG**

SQLE_PAGE_SIZE_TOO_SMALL 変数

SQLE_PAGE_SIZE_TOO_SMALL(-972)。

構文

final int **SQLCode.SQLE_PAGE_SIZE_TOO_SMALL**

SQLE_PARAMETER_CANNOT_BE_NULL 変数

SQLE_PARAMETER_CANNOT_BE_NULL(-1277)。

構文

final int **SQLCode.SQLE_PARAMETER_CANNOT_BE_NULL**

SQLE_PERMISSION_DENIED 変数

SQLE_PERMISSION_DENIED(-121)。

構文

final int **SQLCode.SQLE_PERMISSION_DENIED**

SQLE_PRIMARY_KEY_NOT_UNIQUE 変数

SQLE_PRIMARY_KEY_NOT_UNIQUE(-193)。

構文

final int **SQLCode.SQLE_PRIMARY_KEY_NOT_UNIQUE**

SQLE_PUBLICATION_NOT_FOUND 変数

SQLE_PUBLICATION_NOT_FOUND(-280)。

構文

final int **SQLCode.SQLE_PUBLICATION_NOT_FOUND**

SQLE_RESOURCE_GOVERNOR_EXCEEDED 変数

SQLE_RESOURCE_GOVERNOR_EXCEEDED(-685)。

構文

final int **SQLCode.SQLE_RESOURCE_GOVERNOR_EXCEEDED**

SQLE_ROW_LOCKED 変数

SQLE_ROW_LOCKED(-1281)。

構文

final int **SQLCode.SQLE_ROW_LOCKED**

SQLE_ROW_UPDATED_SINCE_READ 変数

SQLE_ROW_UPDATED_SINCE_READ(-208)。

構文

final int **SQLCode.SQLE_ROW_UPDATED_SINCE_READ**

SQLE_SCHEMA_UPGRADE_NOT_ALLOWED 変数

SQLE_SCHEMA_UPGRADE_NOT_ALLOWED(-953)。

構文

final int **SQLCode.SQLE_SCHEMA_UPGRADE_NOT_ALLOWED**

SQLE_SERVER_SYNCHRONIZATION_ERROR 変数

SQLE_SERVER_SYNCHRONIZATION_ERROR(-857)。

構文

final int **SQLCode.SQLE_SERVER_SYNCHRONIZATION_ERROR**

SQLE_SUBQUERY_RESULT_NOT_UNIQUE 変数

SQLE_SUBQUERY_RESULT_NOT_UNIQUE(-186)。

構文

final int **SQLCode.SQLE_SUBQUERY_RESULT_NOT_UNIQUE**

SQLE_SUBQUERY_SELECT_LIST 変数

SQLE_SUBQUERY_SELECT_LIST(-151)。

構文

final int **SQLCode.SQLE_SUBQUERY_SELECT_LIST**

SQLE_SYNCHRONIZATION_IN_PROGRESS 変数

SQLE_SYNCHRONIZATION_IN_PROGRESS(-1272)。

構文

final int **SQLCode.SQLE_SYNCHRONIZATION_IN_PROGRESS**

SQLE_SYNC_INFO_INVALID 変数

SQLE_SYNC_INFO_INVALID(-956)。

構文

final int **SQLCode.SQLE_SYNC_INFO_INVALID**

SQLE_SYNTAX_ERROR 変数

SQLE_SYNTAX_ERROR(-131)。

構文

final int **SQLCode.SQLE_SYNTAX_ERROR**

SQLE_TABLE_HAS_NO_COLUMNS 変数

SQLE_TABLE_HAS_NO_COLUMNS(-1273)。

構文

final int **SQLCode.SQLE_TABLE_HAS_NO_COLUMNS**

SQLE_TABLE_IN_USE 変数

SQLE_TABLE_IN_USE(-214)。

構文

final int **SQLCode.SQLE_TABLE_IN_USE**

SQLE_TABLE_NOT_FOUND 変数

SQLE_TABLE_NOT_FOUND(-141)。

構文

final int **SQLCode.SQLE_TABLE_NOT_FOUND**

SQLE_TOO_MANY_PUBLICATIONS 変数

SQLE_TOO_MANY_PUBLICATIONS(-1106)。

構文

final int **SQLCode.SQLE_TOO_MANY_PUBLICATIONS**

SQLE_ULTRALITEJ_OPERATION_FAILED 変数

SQLE_ULTRALITEJ_OPERATION_FAILED(-1279)。

構文

final int **SQLCode.SQLE_ULTRALITEJ_OPERATION_FAILED**

SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED 変数

SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED(-1278)。

構文

final int **SQLCode.SQLE_ULTRALITEJ_OPERATION_NOT_ALLOWED**

SQLE_ULTRALITE_DATABASE_NOT_FOUND 変数

SQLE_ULTRALITE_DATABASE_NOT_FOUND(-954)。

構文

final int **SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND**

SQLE_ULTRALITE_OBJ_CLOSED 変数

SQLE_ULTRALITE_OBJ_CLOSED(-908)。

構文

final int **SQLCode.SQLE_ULTRALITE_OBJ_CLOSED**

SQLE_UNABLE_TO_CONNECT 変数

SQLE_UNABLE_TO_CONNECT(-105)。

構文

final int **SQLCode.SQLE_UNABLE_TO_CONNECT**

SQLE_UNCOMMITTED_TRANSACTIONS 変数

SQLE_UNCOMMITTED_TRANSACTIONS(-755)。

構文

final int **SQLCode.SQLE_UNCOMMITTED_TRANSACTIONS**

SQLE_UNDERFLOW 変数

SQLE_UNDERFLOW(-1280)。

構文

final int **SQLCode.SQLE_UNDERFLOW**

SQLE_UNKNOWN_FUNC 変数

SQLE_UNKNOWN_FUNC(-148)。

構文

final int **SQLCode.SQLE_UNKNOWN_FUNC**

SQLE_UPLOAD_FAILED_AT_SERVER 変数

SQLE_UPLOAD_FAILED_AT_SERVER(-794)。

構文

final int **SQLCode.SQLE_UPLOAD_FAILED_AT_SERVER**

SQLE_VALUE_IS_NULL 変数

SQLE_VALUE_IS_NULL(-1050)。

構文

final int **SQLCode.SQLE_VALUE_IS_NULL**

SQLE_VARIABLE_INVALID 変数

SQLE_VARIABLE_INVALID(-155)。

構文

final int **SQLCode.SQLE_VARIABLE_INVALID**

SQLE_WRONG_NUM_OF_INSERT_COLS 変数

SQLE_WRONG_NUM_OF_INSERT_COLS(-207)。

構文

final int **SQLCode.SQLE_WRONG_NUM_OF_INSERT_COLS**

SQLE_WRONG_PARAMETER_COUNT 変数

SQLE_WRONG_PARAMETER_COUNT(-154)。

構文

final int **SQLCode.SQLE_WRONG_PARAMETER_COUNT**

StreamHTTPParms インタフェース

HTTP を使用して Mobile Link サーバと通信する方法を定義する HTTP ストリーム・パラメータを表します。

構文

public **StreamHTTPParms**

派生クラス

- 「StreamHTTPSParms インタフェース」 214 ページ

備考

次の例では、ホスト名 "MyMLHost" にある Mobile Link 11 のサーバと通信するようにストリーム・パラメータを設定しています。サーバは、パラメータ "-xo http(port=1234)" を指定して起動されています。

```
SyncParms syncParms = myConnection.createSyncParms(
    SyncParms.HTTP_STREAM,
    "MyUniqueMLUserID",
    "MyMLScriptVersion"
);
StreamHTTPParms httpParms = syncParms.getStreamParms();
httpParms.setHost("MyMLHost");
httpParms.setPort(1234);
```

このインタフェースを実装するインスタンスは、getStreamParms 関数から返されます。

メンバ

StreamHTTPParms のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getHost メソッド」 209 ページ
- 「getOutputBufferSize メソッド」 210 ページ
- 「getPort メソッド」 210 ページ
- 「getURLSuffix メソッド」 211 ページ
- 「setHost メソッド」 211 ページ
- 「setOutputBufferSize メソッド」 211 ページ
- 「setPort メソッド」 212 ページ
- 「setURLSuffix メソッド」 212 ページ

getHost メソッド

Mobile Link サーバのホスト名を返します。

構文

String **StreamHTTPParms.getHost()**

参照

- 「setHost メソッド」 211 ページ
- 「getPort メソッド」 210 ページ
- 「setPort メソッド」 212 ページ

戻り値

ホスト名。

getOutputBufferSize メソッド

データが Mobile Link サーバに送信される前に格納される出力バッファのサイズをバイト単位で返します。

構文

int **StreamHTTPParms.getOutputBufferSize()**

備考

この値を大きくすると、サイズの大きいアップロードの送信に必要なネットワーク・フラッシュの回数が減る可能性がありますが、メモリの使用量が増えます。HTTP では、フラッシュごとに大容量 (約 250 バイト) の HTTP ヘッダが送信されるので、フラッシュの回数が減ると帯域幅の使用量が削減されます。

参照

- 「setOutputBufferSize メソッド」 211 ページ

戻り値

バッファのサイズを含む整数。

getPort メソッド

Mobile Link サーバへの接続に使用されているポート番号を返します。

構文

int **StreamHTTPParms.getPort()**

参照

- 「setPort メソッド」 212 ページ

戻り値

Mobile Link サーバのポート番号。

getURLSuffix メソッド

URL サフィックスを含む String を返します。

構文

String **StreamHTTPParms.getURLSuffix()**

参照

- 「setURLSuffix メソッド」 212 ページ

戻り値

URL サフィックスを含む String。

setHost メソッド

Mobile Link サーバのホスト名を設定します。

構文

void **StreamHTTPParms.setHost(**
  String *v*
)

パラメータ

- **v**　ホスト名。

備考

デフォルトは NULL で、これは localhost を示します。

参照

- 「getHost メソッド」 209 ページ
- 「getPort メソッド」 210 ページ
- 「setPort メソッド」 212 ページ

setOutputBufferSize メソッド

データが Mobile Link サーバに送信される前に格納される出力バッファのサイズをバイト単位で設定します。

構文

void **StreamHTTPParms.setOutputBufferSize(**
  int *size*
)

パラメータ

- **size**　新しいバッファのサイズ。

備考

デフォルトは Blackberry J2ME 以外では 512 で、Blackberry J2ME では 4096 です。有効な値の範囲は 512 ～ 32768 です。この値を大きくすると Java ランタイムから、Mobile Link サーバでは処理できないチャンク形式の HTTP が送信される可能性があります。Mobile Link サーバから「未知の転送エンコードです」というエラーが出力された場合は、この値を小さくしてみてください。

参照

- 「getOutputBufferSize メソッド」 210 ページ

setPort メソッド

Mobile Link サーバへの接続に使用するポート番号を設定します。

構文

```
void StreamHTTPParms.setPort(
  int v
)
```

パラメータ

- **v**　1 ～ 65535 のポート番号。範囲外の値はデフォルト値に変更されます。

備考

デフォルトのポートは HTTP 同期の場合は 80、HTTPS 同期の場合は 443 です。

参照

- 「getPort メソッド」 210 ページ

setURLSuffix メソッド

Mobile Link サーバの URL サフィックスを設定します。

構文

```
void StreamHTTPParms.setURLSuffix(
  String v
)
```

パラメータ

- **v**　URL サフィックスの文字列。

備考

デフォルトは NULL で、これは "Mobilink/" を示します。

参照

- 「getURLSuffix メソッド」 211 ページ

StreamHTTPSParms インタフェース

セキュア HTTP を使用して Mobile Link サーバと通信する方法を定義する HTTPS ストリーム・パラメータを表します。

構文

public **StreamHTTPSParms**

基本クラス

● 「StreamHTTPParms インタフェース」 209 ページ

備考

次の例では、ホスト名 "MyMLHost" にある Mobile Link 11 のサーバと通信するようにストリーム・パラメータを設定しています。サーバは、パラメータ "-xo https(port=1234;certificate=RSAServer.crt;certificate_password=x)" を指定して起動されています。

```
SyncParms syncParms = myConnection.createSyncParms(
   SyncParms.HTTPS,
   "MyUniqueMLUserID",
   "MyMLScriptVersion"
);
StreamHTTPSParms httpsParms =
   (StreamHTTPSParms) syncParms.getStreamParms();
httpsParms.setHost("MyMLHost");
httpsParms.setPort(1234);
```

上記の例では、RSAServer.crt 内の証明書が、クライアント・ホストまたはデバイスにインストール済みの信頼できるルート証明書のチェーンに追加されていることを前提としています。

J2SE の場合、次のいずれかの方法で、必要な信頼できるルート証明書を配備できます。

1. 信頼できるルート証明書を JRE の lib/security/cacerts キー・ストアにインストールする。

2. Java の keytool ユーティリティを使用して独自のキー・ストアを構築し、Java システム・プロパティ javax.net.ssl.trustStore をその場所に設定する (javax.net.ssl.trustStorePassword を適切な値に設定する)。

3. setTrustedCertificates 関数パラメータを使用して、配備された証明書ファイルを参照する。

セキュリティを強化するには、setCertificateName、setCertificateCompany、setCertificateUnit の各メソッドを使用して Mobile Link サーバの証明書の検証を有効にします。

このインタフェースを実装するインスタンスは、HTTPS 同期用に SyncParms クラス・オブジェクトが作成されたときに、getStreamParms 関数によって返されます。

メンバ

StreamHTTPSParms のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getCertificateCompany メソッド」 215 ページ
- 「getCertificateName メソッド」 215 ページ
- 「getCertificateUnit メソッド」 216 ページ
- 「getHost メソッド」 209 ページ
- 「getOutputBufferSize メソッド」 210 ページ
- 「getPort メソッド」 210 ページ
- 「getTrustedCertificates メソッド」 216 ページ
- 「getURLSuffix メソッド」 211 ページ
- 「setCertificateCompany メソッド」 216 ページ
- 「setCertificateName メソッド」 216 ページ
- 「setCertificateUnit メソッド」 217 ページ
- 「setHost メソッド」 211 ページ
- 「setOutputBufferSize メソッド」 211 ページ
- 「setPort メソッド」 212 ページ
- 「setTrustedCertificates メソッド」 217 ページ
- 「setURLSuffix メソッド」 212 ページ

getCertificateCompany メソッド

セキュア接続の検証に使用する証明書の会社名を返します。

構文

String **StreamHTTPSParms.getCertificateCompany()**

戻り値

証明書の会社名。

getCertificateName メソッド

セキュア接続の検証に使用する証明書の通称を返します。

構文

String **StreamHTTPSParms.getCertificateName()**

戻り値

証明書の名前。

getCertificateUnit メソッド

セキュア接続の検証に使用する証明書に記載される部署名を返します。

構文

String **StreamHTTPSParms.getCertificateUnit()**

戻り値

組織単位名。

getTrustedCertificates メソッド

安全な同期に使用される信頼できるルート証明書のリストを含むファイルの名前を返します。

構文

String **StreamHTTPSParms.getTrustedCertificates()**

戻り値

信頼できるルート証明書ファイルのファイル名。

setCertificateCompany メソッド

セキュア接続の検証に使用する証明書の会社名を設定します。

構文

void **StreamHTTPSParms.setCertificateCompany(**
  String *val*
)

パラメータ

- **val**　会社名。

備考

デフォルトは NULL で、この場合、証明書で会社名は検証されません。

setCertificateName メソッド

セキュア接続の検証に使用する証明書の通称を設定します。

構文

```
void StreamHTTPSParms.setCertificateName(
  String val
)
```

パラメータ

- **val**　証明書の通称。

備考

デフォルトは NULL で、この場合、証明書で通称は検証されません。

setCertificateUnit メソッド

セキュア接続の検証に使用する証明書に記載される部署名を設定します。

構文

```
void StreamHTTPSParms.setCertificateUnit(
  String val
)
```

パラメータ

- **val**　会社の組織単位名。

備考

デフォルトは NULL で、この場合、証明書で組織単位名は検証されません。

setTrustedCertificates メソッド

安全な同期に使用される信頼できるルート証明書のリストを含むファイルを設定します。

構文

```
void StreamHTTPSParms.setTrustedCertificates(
  String filename
) throws ULjException
```

パラメータ

- **filename**　信頼できるルート証明書のファイル名。

備考

このパラメータは J2SE システムだけで使用します。

デフォルトは NULL で、この場合、システムのデフォルトの証明書ストアを使用して、Mobile Link サーバからの証明書チェーンが検証されます。

SyncObserver インタフェース

同期の進行状況の情報を受け取ります。

構文

public **SyncObserver**

備考

同期中に進行状況のレポートを受け取るには、そのタスクを実行する新しいクラスを作成し、setSyncObserver 関数を使用して実装します。

次の例は、単純な SyncObserver インタフェースを示しています。

```
class MyObserver implements SyncObserver {
    public boolean syncProgress(int state, SyncResult result) {
        System.out.println(
            "sync progress state = " + state
            + " bytes sent = " + result.getSentByteCount()
            + " bytes received = " + result.getReceivedByteCount()
        );
        return false;   // Always continue synchronization.
    }
    public MyObserver() {} // The default constructor.
}
```

上記の observer クラスは次のようにして有効にします。

```
// J2ME Sample
Connection conn;
ConfigRecordStore config = DatabaseManager.createConfigurationRecordStore(
    "test.ulj"
    );
try {
    conn = DatabaseManager.connect(config);
} catch(ULjException ex) {
    conn = DatabaseManager.createDatabase(config);
    // Create the schema here.
}
SyncParms.setSyncObserver(new MyObserver());
```

メンバ

SyncObserver のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「syncProgress メソッド」 218 ページ

syncProgress メソッド

同期中に呼び出され、進行状況をユーザに通知します。

構文

boolean **SyncObserver.syncProgress(**
  int *state*,


```
  SyncResult data
)
```

パラメータ

- **state**　同期の現在のステータスを表す SyncOberver.States 定数の 1 つ。
- **data**　同期の最新の結果を含む SyncResult。

備考

パケットとして通知されるさまざまなステータスが送受信されます。1 つのパケットで複数のテーブルがアップロードまたはダウンロードされる場合もあるので、特定の同期に対する syncProgress の呼び出しで、いくつかのステータスが省略される場合があります。

> **注意**
> SyncResult クラス・メソッドを除き、syncProgress 呼び出し中に他の Ultra Light J API メソッドを呼び出すことはできません。

参照

- 「SyncObserver.States インタフェース」 220 ページ
- 「setSyncObserver メソッド」 236 ページ

戻り値

同期をキャンセルする場合は true、続行する場合は false を返します。

SyncObserver.States インタフェース

observer に通知できる同期ステータスを定義します。

構文

public **SyncObserver.States**

参照

- 「setSyncObserver メソッド」 236 ページ
- 「SyncObserver インタフェース」 218 ページ

メンバ

SyncObserver.States のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「CHECKING_LAST_UPLOAD 変数」 220 ページ
- 「COMMITTING_DOWNLOAD 変数」 220 ページ
- 「DISCONNECTING 変数」 221 ページ
- 「DONE 変数」 221 ページ
- 「ERROR 変数」 221 ページ
- 「FINISHING_UPLOAD 変数」 221 ページ
- 「RECEIVING_TABLE 変数」 221 ページ
- 「RECEIVING_UPLOAD_ACK 変数」 222 ページ
- 「ROLLING_BACK_DOWNLOAD 変数」 222 ページ
- 「SENDING_DOWNLOAD_ACK 変数」 222 ページ
- 「SENDING_HEADER 変数」 222 ページ
- 「SENDING_TABLE 変数」 222 ページ
- 「STARTING 変数」 222 ページ

CHECKING_LAST_UPLOAD 変数

前のアップロードのステータスをチェックしています。

構文

final int **SyncObserver.States.CHECKING_LAST_UPLOAD**

COMMITTING_DOWNLOAD 変数

構文

final int **SyncObserver.States.COMMITTING_DOWNLOAD**

備考

ダウンロードされたローがデータベースにコミットされています。

DISCONNECTING 変数

同期ストリームを切断しています。

構文

final int **SyncObserver.States.DISCONNECTING**

DONE 変数

同期が完了しました。

構文

final int **SyncObserver.States.DONE**

備考

その他のステータスはレポートされません。

ERROR 変数

同期は完了しましたが、エラーが発生しました。

構文

final int **SyncObserver.States.ERROR**

FINISHING_UPLOAD 変数

アップロードの完了処理中です。

構文

final int **SyncObserver.States.FINISHING_UPLOAD**

RECEIVING_TABLE 変数

新しいテーブルがダウンロードされています。

構文

final int **SyncObserver.States.RECEIVING_TABLE**

RECEIVING_UPLOAD_ACK 変数

アップロードの確認がダウンロードされています。

構文

final int **SyncObserver.States.RECEIVING_UPLOAD_ACK**

ROLLING_BACK_DOWNLOAD 変数

ダウンロード中にエラーが発生したため、同期によってダウンロードがロールバックされています。

構文

final int **SyncObserver.States.ROLLING_BACK_DOWNLOAD**

SENDING_DOWNLOAD_ACK 変数

ダウンロード完了の確認が送信されています。

構文

final int **SyncObserver.States.SENDING_DOWNLOAD_ACK**

SENDING_HEADER 変数

同期ストリームが開かれ、ヘッダが送信されようとしています。

構文

final int **SyncObserver.States.SENDING_HEADER**

SENDING_TABLE 変数

新しいテーブルがアップロードされています。

構文

final int **SyncObserver.States.SENDING_TABLE**

STARTING 変数

同期を開始しています。処理は行われていません。

構文

final int **SyncObserver.States.STARTING**

SyncParms クラス

データベース同期処理中に使用されたパラメータを保持します。

構文

public **SyncParms**

備考

このインタフェースは、Connection オブジェクトの createSyncParms メソッドを使用して呼び出します。

同期コマンドは一度に 1 つだけ設定できます。コマンドは、setDownloadOnly、setPingOnly、setUploadOnly の各メソッドを使用して指定します。このいずれかのメソッドを true に設定すると、他のメソッドが false に設定されます。

UserName パラメータと Version パラメータは設定する必要があります。UserName はクライアント・データベースごとにユニークである必要があります。

通信ストリームは、getStreamParms メソッドを使用して、SyncParms クラス・オブジェクトのタイプに基づいて設定します。たとえば、次のコードでは、HTTP 同期の準備と実行を行っています。

```
SyncParms syncParms = myConnection.createSyncParms(
   SyncParms.HTTP_STREAM,
   "MyUniqueMLUserID",
   "MyMLScriptVersion"
);
syncParms.setPassword("ThePWDforMyUniqueMLUserID");
syncParms.getStreamParms().setHost("MyMLHost");
myConnection.synchronize(syncParms);
```

Comma Separated Lists AuthenticationParms、Publications、TableOrder の各パラメータはすべて、値のカンマ区切りのリストを含む文字列値を使用して指定します。リスト内の値は一重引用符または二重引用符で囲むことができますが、エスケープ文字はありません。引用符がなかった場合、値の前後のスペースは無視されます。次に例を示します。

```
syncParms.setTableOrder("'Table A',¥"Table B,D¥",Table C" );
```

このコードは、"Table A"、"Table B,D"、"Table C" を指定することになります。

メンバ

SyncParms のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

HTTPS_STREAM 変数

セキュア HTTPS 同期用の SyncParms クラス・オブジェクトを作成します。

構文

final int **SyncParms.HTTPS_STREAM**

参照

- 「createSyncParms メソッド」 135 ページ

HTTP_STREAM 変数

HTTP 同期用の SyncParms クラス・オブジェクトを作成します。

構文

final int **SyncParms.HTTP_STREAM**

参照

- 「createSyncParms メソッド」 135 ページ

SyncParms メソッド

SyncParms クラス・オブジェクトを作成するには、createSyncParms 関数を使用します。

構文

SyncParms.SyncParms()

参照

- 「createSyncParms メソッド」 135 ページ

getAcknowledgeDownload メソッド

リモートからダウンロードの確認が送信されるかどうかを確認します。

構文

abstract boolean **SyncParms.getAcknowledgeDownload()**

参照

- 「setAcknowledgeDownload メソッド」 231 ページ

戻り値

リモートからダウンロードの確認が送信される場合は true、それ以外の場合は false。

getAuthenticationParms メソッド

カスタムのユーザ認証スクリプトに渡されるパラメータを返します。

構文

abstract String **SyncParms.getAuthenticationParms()**

参照

- 「setAuthenticationParms メソッド」 232 ページ

戻り値

認証パラメータのリスト、またはパラメータが指定されていない場合は NULL。

getLivenessTimeout メソッド

活性タイムアウトの長さを秒単位で返します。

構文

abstract int **SyncParms.getLivenessTimeout()**

参照

- 「setLivenessTimeout メソッド」 233 ページ

戻り値

タイムアウト。

getNewPassword メソッド

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

構文

abstract String **SyncParms.getNewPassword()**

参照

- 「setUserName メソッド」 237 ページ
- 「setNewPassword メソッド」 233 ページ

戻り値

次の同期後に設定される新しいパスワード。

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

abstract String **SyncParms.getPassword()**

参照

- 「setPassword メソッド」 234 ページ

戻り値

Mobile Link ユーザのパスワード。

getPublications メソッド

同期させるパブリケーションを返します。

構文

abstract String **SyncParms.getPublications()**

参照

- 「setPublications メソッド」 235 ページ

戻り値

同期するパブリケーションのセット。

getSendColumnNames メソッド

カラム名が Mobile Link サーバに送信されている場合は true を返します。

構文

abstract boolean **SyncParms.getSendColumnNames()**

参照

- 「setSendColumnNames メソッド」 235 ページ

戻り値

カラム名が送信されている場合は true。

getStreamParms メソッド

同期ストリームを設定するパラメータを返します。

構文

abstract StreamHTTPParms **SyncParms.getStreamParms()**

備考

同期ストリームのタイプは、SyncParms クラス・オブジェクトの作成時に指定します。

参照

- 「createSyncParms メソッド」 135 ページ
- 「StreamHTTPParms インタフェース」 209 ページ
- 「StreamHTTPSParms インタフェース」 214 ページ

戻り値

HTTP または HTTPS の同期ストリームのパラメータを指定する StreamHTTPParms インタフェースまたは StreamHTTPSParms オブジェクト。オブジェクトは参照で返されます。

getSyncObserver メソッド

現在の SyncObserver インタフェースを返します。

構文

abstract SyncObserver **SyncParms.getSyncObserver()**

戻り値

SyncObserver インタフェース、または observer が存在しない場合は NULL。

getSyncResult メソッド

同期のステータスを含む SyncResult クラス・オブジェクトを返します。

構文

abstract SyncResult **SyncParms.getSyncResult()**

参照

- 「SyncResult クラス」 239 ページ

戻り値

SyncResult クラス・オブジェクト。

getTableOrder メソッド

統合データベースにテーブルがアップロードされる順序を返します。

構文

abstract String **SyncParms.getTableOrder()**

参照

- 「setTableOrder メソッド」 236 ページ

戻り値

テーブル名のカンマ区切りのリスト、またはテーブルの順序が指定されていない場合は NULL。カンマ区切りのリストの詳細については、クラスの説明を参照してください。

getUserName メソッド

Mobile Link サーバがクライアントをユニークに識別する Mobile Link ユーザ名を返します。

構文

abstract String **SyncParms.getUserName()**

参照

- 「setUserName メソッド」 237 ページ

戻り値

Mobile Link ユーザ名。

getVersion メソッド

使用する同期スクリプトを返します。

構文

abstract String **SyncParms.getVersion()**

参照

- 「setVersion メソッド」 238 ページ

戻り値

スクリプト・バージョン。

isDownloadOnly メソッド

同期がダウンロード専用かどうかを確認します。

構文

abstract boolean **SyncParms.isDownloadOnly()**

参照

- 「setDownloadOnly メソッド」 232 ページ

戻り値

アップロードが無効になっている場合は true、それ以外の場合は false。

isPingOnly メソッド

同期を実行しないで Mobile Link サーバに対して ping が実行されるかどうかを確認します。

構文

abstract boolean **SyncParms.isPingOnly()**

参照

- 「setPingOnly メソッド」 234 ページ

戻り値

クライアントからサーバに対して ping が実行されるだけの場合は true、それ以外の場合は false。

isUploadOnly メソッド

同期がアップロード専用かどうかを確認します。

構文

abstract boolean **SyncParms.isUploadOnly()**

参照

- 「setUploadOnly メソッド」 237 ページ

戻り値

ダウンロードが無効になっている場合は true、それ以外の場合は false。

setAcknowledgeDownload メソッド

リモートが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

```
abstract void SyncParms.setAcknowledgeDownload(
  boolean ack
)
```

パラメータ

- **ack**　クライアントからダウンロード確認を送信する場合は true、それ以外の場合は false に設定します。

備考

デフォルトは false です。

参照

- 「getAcknowledgeDownload メソッド」 226 ページ

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

```
abstract void SyncParms.setAuthenticationParms(
  String v
) throws ULjException
```

パラメータ

- **v**　認証パラメータのカンマ区切りのリスト、または NULL 参照。カンマ区切りのリストの詳細については、クラスの説明を参照してください。

備考

最初の 255 文字列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)。

参照

- 「getAuthenticationParms メソッド」 226 ページ

setDownloadOnly メソッド

同期をダウンロード専用に設定します。

構文

```
abstract void SyncParms.setDownloadOnly(
  boolean v
)
```

パラメータ

- **v**　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。

備考

デフォルトは false です。true を指定すると、setPingOnly と setUploadOnly が false に変更されます。

参照

- 「isDownloadOnly メソッド」 230 ページ
- 「setPingOnly メソッド」 234 ページ
- 「setUploadOnly メソッド」 237 ページ

setLivenessTimeout メソッド

活性タイムアウトの長さを秒単位で設定します。デフォルト値は 100 秒です。

構文

```
abstract void SyncParms.setLivenessTimeout(
  int l
) throws ULjException
```

パラメータ

- **l** 新しい活性タイムアウト値。

備考

活性タイムアウトは、サーバで許容される、リモートのアイドル時間の長さです。リモートが l 秒間サーバと通信しなかった場合、サーバはリモートとの接続が失われたとみなし、同期を終了します。リモートは、接続を継続するために、自動的に定期メッセージをサーバに送信します。

負の値を設定すると、例外がスローされます。値は Mobile Link サーバによって予告なく変更される場合があります。変更は、値が低すぎるか高すぎる場合に行われます。

参照

- 「getLivenessTimeout メソッド」 227 ページ

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

```
abstract void SyncParms.setNewPassword(
  String v
)
```

パラメータ

- **v** 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。

デフォルトは NULL で、この場合、パスワードは置換されません。

参照

- 「getNewPassword メソッド」 227 ページ
- 「setPassword メソッド」 234 ページ
- 「setUserName メソッド」 237 ページ

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

```
abstract void SyncParms.setPassword(
  String v
) throws ULjException
```

パラメータ

- **v** 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、データベースの他のユーザ ID とパスワードと異なります。このメソッドで、Mobile Link サーバに対してアプリケーションが認証されます。

デフォルトは空の文字列で、これはパスワードなしを示します。

参照

- 「getPassword メソッド」 227 ページ
- 「setNewPassword メソッド」 233 ページ
- 「setUserName メソッド」 237 ページ

setPingOnly メソッド

同期を実行しないで Mobile Link サーバに対して ping を実行することを設定します。

構文

```
abstract void SyncParms.setPingOnly(
  boolean v
)
```

パラメータ

- **v** サーバに ping を実行するだけの場合は true、同期を実行する場合は false に設定します。

備考

デフォルトは false です。true を指定すると、setDownloadOnly と setUploadOnly が false に変更されます。

参照

- 「isPingOnly メソッド」 231 ページ
- 「setDownloadOnly メソッド」 232 ページ
- 「setUploadOnly メソッド」 237 ページ

setPublications メソッド

同期させるパブリケーションを設定します。

構文

```
abstract void SyncParms.setPublications(
  String pubs
) throws ULjException
```

パラメータ

- **pubs** パブリケーション名のカンマ区切りのリスト。カンマ区切りのリストの詳細については、クラスの説明を参照してください。

備考

デフォルトは、すべてのテーブルの同期を指定する Connection.SYNC_ALL です。すべてのパブリケーションを同期するには、Connection.SYNC_ALL_PUBS を使用します。

参照

- 「getPublications メソッド」 228 ページ
- 「SYNC_ALL 変数」 130 ページ
- 「SYNC_ALL_PUBS 変数」 130 ページ
- 「createPublication メソッド」 134 ページ

setSendColumnNames メソッド

同期中にカラム名を Mobile Link サーバに送信するかどうかを設定します。デフォルト値は false です。

構文

```
abstract void SyncParms.setSendColumnNames(
  boolean c
)
```

パラメータ

- **c** カラム名を送信する場合は true

備考

カラム名は、ダイレクト・ロー API を使用している場合にのみサーバで使用されます。

参照

- 「getSendColumnNames メソッド」 228 ページ

setSyncObserver メソッド

同期の進行状況をモニタする SyncObserver オブジェクトを設定します。

構文

```
abstract void SyncParms.setSyncObserver(
  SyncObserver so
)
```

パラメータ

- **so** SyncObserver オブジェクト。

備考

デフォルトは NULL で、これは observer なしを示します。

参照

- 「SyncObserver インタフェース」 218 ページ

setTableOrder メソッド

統合データベースにテーブルがアップロードされる順序を設定します。

構文

```
abstract void SyncParms.setTableOrder(
  String v
) throws ULjException
```

パラメータ

- **v** 同期する順序でのテーブル名のカンマ区切りのリスト、またはテーブル順序を指定しない場合は NULL。カンマ区切りのリストの詳細については、クラスの説明を参照してください。

備考

プライマリ・テーブルをリストの先頭に指定し、統合データベースで外部キー関係を持つすべてのテーブルをリストに含めます。

パブリケーションで同期対象として選択されているテーブルは、TableOrder パラメータで指定されているかどうかに関係なくすべて同期されます。指定されていないテーブルは、クライアン

ト・データベースでの外部キー関係の順序で同期されます。これらは、指定したテーブルの後に同期されます。

デフォルト値は NULL 参照で、テーブルのデフォルトの順序は上書きされません。

参照

- 「getTableOrder メソッド」 229 ページ
- 「setPublications メソッド」 235 ページ

setUploadOnly メソッド

同期をアップロード専用に設定します。

構文

```
abstract void SyncParms.setUploadOnly(
  boolean v
)
```

パラメータ

- **v** 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。

備考

デフォルトは false です。true を指定すると、setDownloadOnly と setPingOnly が false に変更されます。

参照

- 「isUploadOnly メソッド」 231 ページ
- 「setDownloadOnly メソッド」 232 ページ
- 「setPingOnly メソッド」 234 ページ

setUserName メソッド

Mobile Link サーバがクライアントをユニークに識別する Mobile Link ユーザ名を設定します。

構文

```
abstract void SyncParms.setUserName(
  String v
) throws ULjException
```

パラメータ

- **v** Mobile Link ユーザ名。

備考

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、データベースの他のユーザ ID とパスワードと異なります。このメソッドで、Mobile Link サーバに対してアプリケーションが認証されます。

このパラメータは、SyncParms クラス・オブジェクトの作成時に初期化されます。

参照

- 「getUserName メソッド」 230 ページ
- 「setPassword メソッド」 234 ページ
- 「setNewPassword メソッド」 233 ページ
- 「createSyncParms メソッド」 135 ページ

setVersion メソッド

使用する同期スクリプトを設定します。

構文

```
abstract void SyncParms.setVersion(
  String v
) throws ULjException
```

パラメータ

- **v**　スクリプト・バージョン。

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区別されます。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。バージョン文字列によって、アプリケーションが同期スクリプトのセットから適切に選択できます。

このパラメータは、SyncParms クラス・オブジェクトの作成時に初期化されます。

参照

- 「getVersion メソッド」 230 ページ
- 「createSyncParms メソッド」 135 ページ

SyncResult クラス

指定されたデータベース同期のステータス関連の情報をレポートします。

構文

public **SyncResult**

メンバ

SyncResult のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getAuthStatus メソッド」 239 ページ
- 「getAuthValue メソッド」 239 ページ
- 「getCurrentTableName メソッド」 240 ページ
- 「getIgnoredRows メソッド」 240 ページ
- 「getReceivedByteCount メソッド」 240 ページ
- 「getReceivedRowCount メソッド」 240 ページ
- 「getSentByteCount メソッド」 241 ページ
- 「getSentRowCount メソッド」 241 ページ
- 「getStreamErrorCode メソッド」 241 ページ
- 「getStreamErrorMessage メソッド」 241 ページ
- 「getSyncedTableCount メソッド」 242 ページ
- 「getTotalTableCount メソッド」 242 ページ
- 「isUploadOK メソッド」 242 ページ

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

abstract int **SyncResult.getAuthStatus()**

戻り値

AuthStatusCode の値。

getAuthValue メソッド

カスタム・ユーザ認証同期スクリプトで指定されている値を返します。

構文

abstract int **SyncResult.getAuthValue()**

戻り値

カスタム・ユーザ認証同期スクリプトから返された整数。

getCurrentTableName メソッド

現在同期中のテーブルの名前を返します。

構文

abstract String **SyncResult.getCurrentTableName()**

戻り値

テーブル名。

getIgnoredRows メソッド

前回行われた同期で、アップロードされたローが無視されたかどうかを確認します。

構文

abstract boolean **SyncResult.getIgnoredRows()**

戻り値

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、ローが無視されなかった場合は false。

getReceivedByteCount メソッド

データ同期中に受信したバイト数を返します。

構文

abstract long **SyncResult.getReceivedByteCount()**

戻り値

バイト数。

getReceivedRowCount メソッド

受信したローの数を返します。

構文

abstract int **SyncResult.getReceivedRowCount()**

戻り値

ローの数。

getSentByteCount メソッド

データ同期中に送信されたバイト数を返します。

構文

abstract long **SyncResult.getSentByteCount()**

戻り値

送信されたバイト数。

getSentRowCount メソッド

送信されたローの数を返します。

構文

abstract int **SyncResult.getSentRowCount()**

戻り値

ローの数。

getStreamErrorCode メソッド

ストリームによってレポートされたエラー・コードを返します。

構文

abstract int **SyncResult.getStreamErrorCode()**

備考

エラー・コードは HTTP の応答コードです。

戻り値

通信ストリーム・エラーがなかった場合は 0、それ以外の場合はサーバからの応答コード。

getStreamErrorMessage メソッド

ストリームによってレポートされたエラー・メッセージを返します。

構文

abstract String **SyncResult.getStreamErrorMessage()**

備考

エラー・コードは HTTP の応答メッセージです。

戻り値

メッセージがない場合は NULL、それ以外の場合は応答メッセージ。

getSyncedTableCount メソッド

現在までに同期されたテーブル数を返します。

構文

abstract int **SyncResult.getSyncedTableCount()**

戻り値

テーブル数。

getTotalTableCount メソッド

同期されるテーブル数を返します。

構文

abstract int **SyncResult.getTotalTableCount()**

戻り値

テーブル数。

isUploadOK メソッド

前回のアップロード同期が成功したかどうかを確認します。

構文

abstract boolean **SyncResult.isUploadOK()**

戻り値

前回のアップロード同期が成功した場合は true、それ以外の場合は false。

SyncResult.AuthStatusCode インタフェース

Mobile Link サーバから返された認証コードを列挙します。

構文

public **SyncResult.AuthStatusCode**

参照

- 「getAuthStatus メソッド」 239 ページ

メンバ

SyncResult.AuthStatusCode のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「EXPIRED 変数」 243 ページ
- 「IN_USE 変数」 243 ページ
- 「INVALID 変数」 243 ページ
- 「UNKNOWN 変数」 244 ページ
- 「VALID 変数」 244 ページ
- 「VALID_BUT_EXPIRES_SOON 変数」 244 ページ

EXPIRED 変数

ユーザ ID またはパスワードの有効期限が切れています。認証に失敗しました。

構文

final int **SyncResult.AuthStatusCode.EXPIRED**

INVALID 変数

ユーザ ID またはパスワードが不正です。認証に失敗しました。

構文

final int **SyncResult.AuthStatusCode.INVALID**

IN_USE 変数

ユーザ ID がすでに使用されています。認証に失敗しました。

構文

final int **SyncResult.AuthStatusCode.IN_USE**

UNKNOWN 変数

認証ステータスが不明です。

構文

final int **SyncResult.AuthStatusCode.UNKNOWN**

備考

このコードは、同期が実行されていないことを示します。

VALID 変数

ユーザ ID とパスワードは、同期時には有効でした。

構文

final int **SyncResult.AuthStatusCode.VALID**

VALID_BUT_EXPIRES_SOON 変数

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。

構文

final int **SyncResult.AuthStatusCode.VALID_BUT_EXPIRES_SOON**

TableSchema インタフェース

テーブルのスキーマを指定し、システム・テーブルの名前を定義する定数を提供します。

構文

public **TableSchema**

備考

このインタフェースをサポートするオブジェクトは、createTable 関数から返されます。

テーブルには、カラムが 1 つ以上とプライマリ・キーが必要です。

次の例は、単純なデータベースのスキーマを作成する方法を示しています。2 つのカラム、プライマリ・キー、インデックスのある T2 テーブルが作成されます。

```
// Assumes a valid Connection object conn
TableSchema table_schema;
IndexSchema index_schema;

table_schema = conn.createTable("T2");
table_schema.createColumn("num", Domain.INTEGER);
table_schema.createColumn("quantity", Domain.INTEGER);

index_schema = table_schema.createPrimaryIndex("primary");
index_schema.addColumn("num", IndexSchema.ASCENDING);
index_schema = table_schema.createIndex("index1");
index_schema.addColumn("quantity", IndexSchema.ASCENDING);

conn.schemaCreateComplete();
```

プライマリ・キーは、テーブル内の各ローをユニークに識別します。プライマリ・キーに含まれるカラムには NULL を使用できません。プライマリ・キーは、createPrimaryIndex 関数を使用して作成します。

ユニーク・キーは、テーブル内の各ローをユニークに識別する 1 つ以上のカラムを指定する制約です。テーブル内の異なるローが、指定されているすべてのカラムで同じ値を持つことはできません。1 つのテーブルに複数の一意性制約が存在することがあります。プライマリ・キーはユニーク・キーです。ユニーク・キーは、createUniqueKey 関数を使用して作成します。

ユニーク・インデックスによって、インデックス内のすべてのカラムで同じ値を持つローがテーブル内に複数存在しないようにします。各インデックス・キーはユニークであるか、少なくとも 1 つのカラムで NULL を持つ必要があります。ユニーク・インデックスは、createUniqueIndex 関数を使用して作成します。

インデックスが無制限であると、重複するインデックスのエントリや NULL のカラムが許可されます。標準のインデックスは、createIndex 関数を使用して作成します。

メンバ

TableSchema のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「createColumn メソッド」 248 ページ
- 「createColumn メソッド」 249 ページ
- 「createColumn メソッド」 250 ページ
- 「createIndex メソッド」 250 ページ
- 「createPrimaryIndex メソッド」 251 ページ
- 「createUniqueIndex メソッド」 251 ページ
- 「createUniqueKey メソッド」 252 ページ
- 「setNoSync メソッド」 252 ページ
- 「SYS_ARTICLES 変数」 246 ページ
- 「SYS_COLUMNS 変数」 246 ページ
- 「SYS_FKEY_COLUMNS 変数」 246 ページ
- 「SYS_FOREIGN_KEYS 変数」 247 ページ
- 「SYS_INDEX_COLUMNS 変数」 247 ページ
- 「SYS_INDEXES 変数」 247 ページ
- 「SYS_INTERNAL 変数」 247 ページ
- 「SYS_PRIMARY_INDEX 変数」 247 ページ
- 「SYS_PUBLICATIONS 変数」 248 ページ
- 「SYS_TABLES 変数」 248 ページ
- 「TABLE_IS_NOSYNC 変数」 248 ページ
- 「TABLE_IS_SYSTEM 変数」 248 ページ

SYS_ARTICLES 変数

パブリケーションのアーティクルに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_ARTICLES**

SYS_COLUMNS 変数

データベース内のテーブル・カラムに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_COLUMNS**

SYS_FKEY_COLUMNS 変数

外部キー・カラムに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_FKEY_COLUMNS**

SYS_FOREIGN_KEYS 変数

データベース内の外部キーに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_FOREIGN_KEYS**

SYS_INDEXES 変数

データベース内のテーブル・インデックスに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_INDEXES**

SYS_INDEX_COLUMNS 変数

データベース内のインデックス・カラムに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_INDEX_COLUMNS**

SYS_INTERNAL 変数

データベース・オプションと内部データベース・データに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_INTERNAL**

SYS_PRIMARY_INDEX 変数

システム・テーブルのプライマリ・キー・インデックスの名前です。

構文

final String **TableSchema.SYS_PRIMARY_INDEX**

SYS_PUBLICATIONS 変数

データベース・パブリケーションに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_PUBLICATIONS**

SYS_TABLES 変数

データベース内のテーブルに関する情報を含むシステム・テーブルの名前です。

構文

final String **TableSchema.SYS_TABLES**

TABLE_IS_NOSYNC 変数

テーブルが NoSync テーブルである (テーブルは同期されない) ことを示すビット・フラグです。

構文

final short **TableSchema.TABLE_IS_NOSYNC**

備考

この値は、SYS_TABLES テーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

TABLE_IS_SYSTEM 変数

テーブルがシステム・テーブルであることを示すビット・フラグです。

構文

final short **TableSchema.TABLE_IS_SYSTEM**

備考

この値は、SYS_TABLES テーブルの table_flags カラムで他のフラグと論理的に組み合わせることができます。

createColumn メソッド

固定サイズ型の新しいカラムを作成します。

構文

```
ColumnSchema TableSchema.createColumn(
  String column_name,
  short column_type
) throws ULjException
```

パラメータ

- **column_name**　新しいカラムの名前。指定する名前は有効な SQL 識別子である必要があります。
- **column_type**　固定サイズのカラム型を表す Domain 型定数の 1 つ。

参照

「Domain インタフェース」 159 ページ

戻り値

指定された名前と型で作成されたカラムに割り当てられている ColumnSchema。

createColumn メソッド

可変サイズ型の新しいカラムを作成します。

構文

```
ColumnSchema TableSchema.createColumn(
  String column_name,
  short column_type,
  int column_size
) throws ULjException
```

パラメータ

- **column_name**　新しいカラムの名前。指定する名前は有効な SQL 識別子である必要があります。
- **column_type**　可変サイズのカラム型を表す Domain 型定数の 1 つ (BINARY、NUMERIC、VARCHAR)。
- **column_size**　カラムのサイズ。

備考

カラム型が固定サイズの場合、サイズは無視されます。

参照

「Domain インタフェース」 159 ページ

戻り値

指定された名前と型で作成されたカラムに割り当てられている ColumnSchema。

createColumn メソッド

サイズと精度が可変の新しいカラムを作成します。

構文

```
ColumnSchema TableSchema.createColumn(
  String column_name,
  short column_type,
  int column_size,
  int column_scale
) throws ULjException
```

パラメータ

- **column_name**　新しいカラムの名前。指定する名前は有効な SQL 識別子である必要があります。
- **column_type**　サイズと位取りが可変のカラム型を表す Domain 型定数の 1 つ (NUMERIC)。
- **column_size**　カラムのサイズ。
- **column_scale**　カラムの位取り。

備考

カラム型が固定サイズの場合、サイズまたは位取りは無視されます。

参照

「Domain インタフェース」 159 ページ

戻り値

指定された名前と型で作成されたカラムに割り当てられている ColumnSchema。

createIndex メソッド

新しいインデックスを作成します。

構文

```
IndexSchema TableSchema.createIndex(
  String index_name
) throws ULjException
```

パラメータ

- **index_name**　インデックスの名前。指定する名前は有効な SQL 識別子である必要があります。

戻り値

指定された名前で作成されたインデックスに割り当てられている IndexSchema。

createPrimaryIndex メソッド

テーブルにプライマリ・インデックスを作成します。

構文

```
IndexSchema TableSchema.createPrimaryIndex(
  String index_name
) throws ULjException
```

パラメータ

- **index_name**　インデックスの名前。指定する名前は有効な SQL 識別子である必要があります。

備考

各テーブルにはプライマリ・インデックスが 1 つだけ必要です。プライマリ・インデックス内のカラムは NULL 入力不可である必要があります。

戻り値

指定された名前で作成されたプライマリ・インデックスに割り当てられている IndexSchema。

createUniqueIndex メソッド

新しいユニークなインデックスを作成します。

構文

```
IndexSchema TableSchema.createUniqueIndex(
  String index_name
) throws ULjException
```

パラメータ

- **index_name**　インデックスの名前。指定する名前は有効な SQL 識別子である必要があります。

備考

各インデックス・キーはユニークであるか、少なくとも 1 つのカラムで NULL を持つ必要があります。

ユニーク・キー制約内のカラムと異なり、ユニーク・インデックス内のカラムは NULL 入力可です。外部キーではプライマリ・キーまたはユニーク・キーを参照できますが、ユニーク・インデックスは参照できません。

戻り値

指定された名前で作成されたユニーク・インデックスに割り当てられている IndexSchema。

createUniqueKey メソッド

新しいユニーク・キーを作成します。

構文

```
IndexSchema TableSchema.createUniqueKey(
  String index_name
) throws ULjException
```

パラメータ

- **index_name**　キーの名前。指定する名前は有効な SQL 識別子である必要があります。

備考

ユニーク・キーは、テーブル内の各ローをユニークに識別する 1 つ以上のカラムを指定する制約です。1 つのテーブルに複数の一意性制約が存在することがあります。

戻り値

指定された名前で作成されたユニーク・キーに割り当てられている IndexSchema。

setNoSync メソッド

テーブルを同期するかどうかを指定します。

構文

```
TableSchema TableSchema.setNoSync(
  boolean no_sync
) throws ULjException
```

パラメータ

- **no_sync**　テーブルの変更内容を同期する場合は true、それ以外の場合は false。

備考

true に設定した場合、ローの変更情報は維持されません。デフォルト値は false です。

戻り値

NoSync が定義された TableSchema。

ULjException クラス

Ultra Light J データベースからスローされた例外に取って代わります。

構文

public **ULjException**

メンバ

ULjException のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

getCausingException メソッド

この例外の原因となっている ULjException を返します。

構文

abstract ULjException **ULjException.getCausingException()** throws **ULjException**

備考

戻り値

原因となっている例外がない場合は NULL、それ以外の場合は ULjException。

getErrorCode メソッド

この例外に関連付けられているエラー・コードを返します。

構文

abstract int **ULjException.getErrorCode()**

備考

戻り値

エラー・コード。

getSqlOffset メソッド

SQL 文字列内のエラー・オフセットを返します。

構文

abstract int **ULjException.getSqlOffset()**

備考

戻り値

エラー・メッセージに関連付けられている SQL 文字列がない場合は -1、それ以外の場合は、文字列内でエラーが発生した箇所の から始まるオフセット。

Value インタフェース

フェッチしたローのカラム値を表します。

構文

public **Value**

基本クラス

- 「ValueReader インタフェース」 261 ページ
- 「ValueWriter インタフェース」 265 ページ

メンバ

Value のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

compareValue メソッド

2 つの Value を比較します。

構文

int **Value.compareValue(**
  Value *other*
) throws **ULjException**

パラメータ

- **other**　比較対象の Value。

戻り値

Value インタフェースが other と同じ場合は 0、other よりも少ない場合は負の整数、other よりも大きい場合は正の整数。

duplicate メソッド

Value を複製して返します。

構文

Value **Value.duplicate()** throws **ULjException**

戻り値

複製した Value。

getDomain メソッド

Value の Domain オブジェクトを返します。

構文

Domain **Value.getDomain()** throws **ULjException**

戻り値

Domain オブジェクト。

getDomainSize メソッド

Value の Domain サイズを返します。

構文

int **Value.getDomainSize()** throws **ULjException**

戻り値

Domain のサイズ。

getSize メソッド

Value の現在のサイズを返します。

構文

int **Value.getSize()** throws **ULjException**

戻り値

サイズ。

getType メソッド

Value の Domain 型を返します。

構文

int **Value.getType()** throws **ULjException**

戻り値

Domain の型。

release メソッド

Value を閉じて、関連付けられているメモリ・リソースを解放します。

構文

void **Value.release()** throws **ULjException**

ValueReader インタフェース

Value オブジェクトを読み取り、java 変数型として解釈します。

構文

public **ValueReader**

派生クラス

- 「Value インタフェース」 257 ページ

メンバ

ValueReader のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getBlobInputStream メソッド」 261 ページ
- 「getBoolean メソッド」 261 ページ
- 「getBytes メソッド」 262 ページ
- 「getClobReader メソッド」 262 ページ
- 「getDate メソッド」 262 ページ
- 「getDecimalNumber メソッド」 262 ページ
- 「getDouble メソッド」 263 ページ
- 「getFloat メソッド」 263 ページ
- 「getInt メソッド」 263 ページ
- 「getLong メソッド」 263 ページ
- 「getString メソッド」 264 ページ
- 「getValue メソッド」 264 ページ
- 「isNull メソッド」 264 ページ

getBlobInputStream メソッド

blob InputStream です。

構文

java.io.InputStream **ValueReader.getBlobInputStream()** throws **ULjException**

備考

Value の blob InputStream を返します。

getBoolean メソッド

Value の boolean 解釈を返します。

構文

boolean **ValueReader.getBoolean()** throws **ULjException**

戻り値

boolean 値。

getBytes メソッド

Value の byte 配列を返します。

構文

byte[] **ValueReader.getBytes()** throws **ULjException**

戻り値

byte 配列。

getClobReader メソッド

Value の clob Reader を返します。

構文

java.io.Reader **ValueReader.getClobReader()** throws **ULjException**

戻り値

clob Reader。

getDate メソッド

Value の日付解釈を返します。

構文

java.util.Date **ValueReader.getDate()** throws **ULjException**

戻り値

Value の日付。

getDecimalNumber メソッド

Value の DecimalNumber 解釈を返します。

構文

DecimalNumber **ValueReader.getDecimalNumber()** throws **ULjException**

戻り値

DecimalNumber 値。

getDouble メソッド

Value の double 解釈を返します。

構文

double **ValueReader.getDouble()** throws **ULjException**

戻り値

double 値。

getFloat メソッド

Value の float 解釈を返します。

構文

float **ValueReader.getFloat()** throws **ULjException**

戻り値

float 値。

getInt メソッド

Value の integer 解釈を返します。

構文

int **ValueReader.getInt()** throws **ULjException**

戻り値

integer 値。

getLong メソッド

Value の long 解釈を返します。

構文

long **ValueReader.getLong()** throws **ULjException**

戻り値

long 値。

getString メソッド

Value の String 解釈を返します。

構文

String **ValueReader.getString()** throws **ULjException**

戻り値

String 値。

getValue メソッド

Value オブジェクトを返します。

構文

Value **ValueReader.getValue()** throws **ULjException**

isNull メソッド

値が NULL かどうかをテストします。

構文

boolean **ValueReader.isNull()**

戻り値

値に NULL が含まれる場合は true、含まれない場合は false。

ValueWriter インタフェース

java 変数型の値を Value オブジェクトに格納します。

構文

public **ValueWriter**

派生クラス

- 「Value インタフェース」 257 ページ

メンバ

ValueWriter のすべてのメンバ (継承されたメンバも含みます) を以下に示します。

- 「getBlobOutputStream メソッド」 265 ページ
- 「getClobWriter メソッド」 265 ページ
- 「set メソッド」 266 ページ
- 「set メソッド」 266 ページ
- 「set メソッド」 266 ページ
- 「set メソッド」 266 ページ
- 「set メソッド」 267 ページ
- 「set メソッド」 267 ページ
- 「set メソッド」 267 ページ
- 「set メソッド」 268 ページ
- 「set メソッド」 268 ページ
- 「set メソッド」 268 ページ
- 「setNull メソッド」 268 ページ

getBlobOutputStream メソッド

Value の blob OutputStream を返します。

構文

java.io.OutputStream **ValueWriter.getBlobOutputStream()** throws **ULjException**

戻り値

blob OutputStream。

getClobWriter メソッド

Value の clob Writer を返します。

構文

java.io.Writer **ValueWriter.getClobWriter()** throws **ULjException**

戻り値

clob Writer。

set メソッド

Value の boolean 値を設定します。

構文

```
void ValueWriter.set(
  boolean value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に DecimalNumber を設定します。

構文

```
void ValueWriter.set(
  DecimalNumber value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に Date を設定します。

構文

```
void ValueWriter.set(
  java.util.Date value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に integer を設定します。

構文

```
void ValueWriter.set(
  int value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に long integer を設定します。

構文

```
void ValueWriter.set(
  long value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に float を設定します。

構文

```
void ValueWriter.set(
  float value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に double を設定します。

構文

```
void ValueWriter.set(
  double value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に byte 配列を設定します。

構文

```
void ValueWriter.set(
  byte[] value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に String を設定します。

構文

```
void ValueWriter.set(
  String value
) throws ULjException
```

パラメータ

- **value**　設定する値。

set メソッド

Value に Value オブジェクトを設定します。

構文

```
void ValueWriter.set(
  Value value
) throws ULjException
```

パラメータ

- **value**　設定する値。

setNull メソッド

Value を NULL に設定します。

構文

```
void ValueWriter.setNull() throws ULjException
```

Ultra Light J のシステム・テーブル

目次

systable システム・テーブル

systable システム・テーブルの各ローは、データベース内のテーブル 1 つを示します。

カラム名	カラム型	説明
table_id	INTEGER	テーブルのユニークな識別子。
table_name	VARCHAR(128)	テーブルの名前。
table_flags	UNSIGNED SHORT	次のいずれかのフラグのビット単位の組み合わせ。 ● TABLE_IS_SYSTEM ● TABLE_IS_NO_SYNC
table_data	INTEGER	内部でのみ使用。
table_autoinc	BIG	内部でのみ使用。

制約

PRIMARY INDEX (table_id)

syscolumn システム・テーブル

syscolumn システム・テーブルの各ローは、カラムを示します。

カラム名	カラム型	説明
table_id	INTEGER	カラムが属するテーブルの識別子。
column_id	INTEGER	カラムのユニークな識別子。
column_name	VARCHAR(128)	カラムの名前。「Domain インタフェース」 159 ページを参照してください。
column_flags	TINY	属性を表す次のフラグのビット単位の組み合わせ。 ● **0x01** カラムはプライマリ・キーに含まれます。 ● **0x02** カラムは NULL 入力可です。
column_domain	INTEGER	カラムのドメインを示す列挙値。
column_length	INTEGER	カラムの長さ。 VARCHAR 型と BINARY 型 (Domain インタフェースで定義) のカラムの場合は、バイト単位の最大長です。NUMERIC 型のカラムの場合は、最初のバイトに精度、2 番目のバイトに位取りが格納されます。
column_default	TINY	このカラムのデフォルト値。ColumnSchema インタフェースの COLUMN_DEFAULT 値の 1 つで指定されます。たとえば、COLUMN_DEFAULT_AUTOINC はオートインクリメントするデフォルト値を表します。

制約

PRIMARY KEY (table_id, column_id)

sysindex システム・テーブル

sysindex システム・テーブルの各ローは、データベース内のインデックスを示します。

カラム名	カラム型	説明
table_id	INTEGER	インデックスが適用されるテーブルのユニークな識別子。
index_id	INTEGER	インデックスのユニークな識別子。
index_name	VARCHAR(128)	インデックスの名前。
index_flags	TINY	インデックスのタイプとその永続性を示す次のフラグのビット単位の組み合わせ。 ● **0x01** ユニーク・キー。 ● **0x02** ユニーク・インデックス。 ● **0x04** インデックスは永続的。 ● **0x08** プライマリ・キー。
index_data	INTEGER	内部でのみ使用。

制約

PRIMARY KEY (table_id, index_id)

sysindexcolumn システム・テーブル

sysindexcolumn システム・テーブルの各ローは、sysindex にリストされているインデックスのカラムを示します。

カラム名	カラム型	説明
table_id	INTEGER	インデックスが適用されるテーブルのユニークな識別子。
index_id	INTEGER	このインデックス・カラムが属するインデックスのユニークな識別子。
order	INTEGER	インデックス内のカラムの順序。
column_id	INTEGER	インデックス対象カラムのユニークな識別子。
index_column_flag	TINY	インデックス内のカラムが保持されている場所 (昇順 (1) または降順 (0)) を示します。

制約

PRIMARY KEY (table_id, index_id, order)

sysinternal システム・テーブル

sysinternal システム・テーブルの各ローには、システム・オプションとその他の内部データが格納されます。

カラム名	カラム型	説明
name	VARCHAR(128)	オプションの名前。
value	VARCHAR(128)	オプションの値。

制約

PRIMARY KEY (name)

syspublications システム・テーブル

syspublications システム・テーブルの各ローは、パブリケーションを示します。

カラム名	カラム型	説明
publication_id	INTEGER	パブリケーションのユニークな識別子。
publication_name	VARCHAR(128)	パブリケーションの名前。
download_timestamp	TIMESTAMP	最後のダウンロードの時刻。
last_sync_sent	INTEGER	Mobile Link に送信されたアップロードを追跡する整数。
last_sync_confirmed	INTEGER	Mobile Link での受信が確認されたアップロードを追跡する整数。

制約

PRIMARY KEY (publication_id)

sysarticles システム・テーブル

sysarticles システム・テーブルの各ローは、パブリケーションに属するテーブルを示します。

カラム名	カラム型	説明
publication _id	INTEGER	このアーティクルが属するパブリケーションの識別子。
table_id	INTEGER	パブリケーションに属するテーブルの識別子。

制約

PRIMARY KEY (publication_id, table_id)

sysforeignkey システム・テーブル

sysforeignkey システム・テーブルの各ローは、テーブルに属する外部キーを示します。

カラム名	カラム型	説明
table_id	INTEGER	外部キーが属するテーブルの識別子。
foreign_table_id	INTEGER	この外部キー・カラムが参照するテーブルの識別子。
foreign_key_id	INTEGER	外部キーの識別子。
name	VARCHAR(128)	外部キーの名前。
index_name	VARCHAR(128)	外部キーが参照しているインデックスの名前。

制約

PRIMARY KEY (table_id, foreign_key_id)

sysfkcol システム・テーブル

sysfkcol システム・テーブルの各ローは、外部キーのカラムを示します。

カラム名	カラム型	説明
table_id	INTEGER	外部キーが適用されるテーブルのユニークな識別子。
foreign_key_id	INTEGER	このカラムが属する外部キーのユニークな識別子。
item_no	SHORT	外部キー内のカラムの順序。
column_id	INTEGER	外部カラムを参照するテーブルのカラムのユニークな識別子。
foreign_column_id	INTEGER	参照されているテーブルのカラムのユニークな識別子。

制約

PRIMARY KEY (table_id, foreign_key_id, item_no)

Ultra Light J のユーティリティ

目次

Ultra Light J には、Ultra Light J データベースの管理タスクを行うためのユーティリティが付属します。

J2SE 用ユーティリティ

これらのユーティリティは、Ultra Light J の J2SE 実装のみを対象としており、BlackBerry スマートフォン環境で使用できるように設計されていません。

Ultra Light J データベース情報ユーティリティ (ULjInfo)

ULjInfo ユーティリティは、既存の Ultra Light J データベースに関する情報を表示します。

構文

ULjInfo -c *filename* **-p** *password* [*options*]

オプション	説明
-c *filename*	必須。検査する Ultra Light J データベースのファイル名です。
-p *password*	必須。Ultra Light J データベースに接続するためのパスワードです。
-q	クワイエット・モードで実行します (メッセージを表示しません)。
-v	冗長メッセージを表示します。
-?	コマンド・ラインの使用法を表示します。

出力例 (非冗長)

次に、ULjInfo プログラムからの出力例 (-v オプションなし) を示します。

```
C:¥ULj¥bin>ULjInfo.cmd -c ..¥Samples¥Demo1.ulj -p sql
SQL Anywhere UltraLite J Database Information Utility
Database name: ..¥Samples¥Demo1.ulj
Disk file: '..¥Samples¥Demo1.ulj'
Database ID: 0
Page size: 1024
0 rows for next upload
Date format: YYYY-MM-DD
Date order: YMD
Nearest century: 50
Numeric precision: 30
Numeric scale: 6
Time format: HH:NN:SS.SSS
Timestamp format: YYYY-MM-DD HH:NN:SS.SSS
Timestamp increment: 1
Number of tables: 1
Number of columns: 2
Number of publications: 0
Number of tables that will always be uploaded: 0
Number of tables that are never synchronized: 0
Number of primary keys: 1
```



```
Number of foreign keys: 0
Number of indexes: 0
Last download occurred on Thu Jul 05 11:31:05 EDT 2007
Upload OK: true
```

Ultra Light J データベース・ロード・ユーティリティ (ULjLoad)

ULjLoad ユーティリティは、XML ソース・ファイルから Ultra Light J データベースをロードする機能を提供します。XML ファイルは通常は ULjUnload ユーティリティで作成し、またカスタマイズが可能です。

構文

ULjLoad -c *filename* **-p** *password* [*options*] *inputfile*

オプション	説明
-a	XML ファイルから既存のデータベースに情報を追加します。このオプションを指定しないと、新しいデータベースが作成されます。
-c *filename*	必須。データベース・ファイルの名前です。
-d	データのみをロードし、スキーマ情報を無視します。
-f *directory*	blob の最大サイズ (ULjUnload の実行時に -b オプションによって指定) よりも大きなカラムのデータを取り出すディレクトリです。
-i	アップロード同期するローを挿入します。
-n	スキーマ情報だけをロードし、ロー・データを無視します。
-p *password*	必須。データベースに接続するためのパスワードです。
-q	クワイエット・モードで実行します (メッセージを表示しません)。
-v	冗長メッセージを表示します。
-y	出力ファイルが存在する場合、それを上書きします (-a オプションは指定しません)。
-?	コマンド・ラインの使用法を表示します。
inputfile	XML 文を含む入力ファイルです。

使用法の概要の例

ULjLoad ユーティリティ・コマンド・ラインに **-?** オプションを指定すると、次の使用法が表示されます。

```
SQL Anywhere UltraLite J Database Load Utility
Usage: uljload [options] <XML file>
```

```
        Create and load data into a new UltraLite J database from <XML file>.

    Options:
        -a      Add to existing database.
        -c      <file> Database file.
        -d      Data only -- ignore schema.
        -f      <directory>
                Directory to store columns larger than <max blob size>.
        -i      Insert rows for upload synchronization.
        -n      Schema only -- ignore data.
        -p      Password to connect to database.
        -q      Quiet: do not print messages.
        -v      Verbose messages.
        -y      Overwrite file if it already exists.
```

Ultra Light J データベース・アンロード・ユーティリティ (ULjUnload)

ULjUnload ユーティリティは、Ultra Light J データベース (データとスキーマのいずれか、またはその両方) を XML ファイルにアンロードする機能を提供します。

構文

ULjUnload -c *filename* **-p** *password* [*options*] *outputfile*

オプション	説明
-b *max-blob-size*	XML に出力される blob または char データの最大サイズ (バイト単位) です。
-c *filename*	必須。アンロードするデータベース・ファイルの名前です。
-d	データのみをアンロードし、スキーマ情報は出力しません。
-e *table, ...*	リスト内に指定されたテーブルのデータを除外します。
-f *directory*	blob の最大サイズ (-b オプションによって指定) よりも大きなカラムのデータを格納するディレクトリです。
-n	スキーマ情報のみをアンロードし、データは出力しません。
-p *password*	必須。データベースに接続するためのパスワードです。
-q	クワイエット・モードで実行します (メッセージを表示しません)。
-t *table, ...*	リスト内に指定されたテーブルのデータのみを出力します。
-v	冗長メッセージを表示します。
-y	出力ファイルがすでに存在する場合、それを上書きします。

オプション	説明
-?	オプションの使用法またはヘルプ情報を表示します。
outputfile	出力ファイルの名前です (このファイルにはデータベースの内容を記述する XML 文が含まれています)。

XML ファイルの内容の例

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ul:ulschema xmlns:ul="urn:ultralite">
 <collation name="1252LATIN1" case_sensitive="no"/>
 <options>
  <option name="dateformat" value="YYYY-MM-DD"/>
  <option name="dateorder" value="YMD"/>
  <option name="nearestcentury" value="50"/>
  <option name="precision" value="30"/>
  <option name="scale" value="6"/>
  <option name="timeformat" value="HH:NN:SS.SSS"/>
  <option name="timestampformat" value="YYYY-MM-DD HH:NN:SS.SSS"/>
  <option name="timestampincrement" value="1"/>
 </options>
 <tables>
  <table name="ULCustomer" sync="changes">
   <columns>
    <column name="cust_id" type="integer" null="no"/>
    <column name="cust_name" type="char(30)" null="yes"/>
   </columns>
   <primarykey>
    <primarycolumn name="cust_id" direction="asc"/>
   </primarykey>
   <indexes/>
  </table>
 </tables>
 <uldata>
  <table name="ULCustomer">
   <row cust_id="2000" cust_name="Apple St. Builders"/>
   <row cust_id="2001" cust_name="Art's Renovations"/>
   <row cust_id="2002" cust_name="Awnings R Us"/>
   <row cust_id="2003" cust_name="Al's Interior Design"/>
   <row cust_id="2004" cust_name="Alpha Hardware"/>
   <row cust_id="2005" cust_name="Ace Properties"/>
   <row cust_id="2006" cust_name="A1 Contracting"/>
   <row cust_id="2007" cust_name="Archibald Inc."/>
   <row cust_id="2008" cust_name="Acme Construction"/>
   <row cust_id="2009" cust_name="ABCXYZ Inc."/>
   <row cust_id="2010" cust_name="Buy It Co."/>
   <row cust_id="2011" cust_name="Bill's Cages"/>
   <row cust_id="2012" cust_name="Build-It Co."/>
   <row cust_id="2013" cust_name="Bass Interiors"/>
   <row cust_id="2014" cust_name="Burger Franchise"/>
   <row cust_id="2015" cust_name="Big City Builders"/>
   <row cust_id="2016" cust_name="Bob's Renovations"/>
   <row cust_id="2017" cust_name="Basements R Us"/>
   <row cust_id="2018" cust_name="BB Interior Design"/>
   <row cust_id="2019" cust_name="Bond Hardware"/>
   <row cust_id="2020" cust_name="Cat Properties"/>
   <row cust_id="2021" cust_name="C &amp;amp; C Contracting"/>
   <row cust_id="2022" cust_name="Classy Inc."/>
   <row cust_id="2023" cust_name="Cooper Construction"/>
   <row cust_id="2024" cust_name="City Schools"/>
```

```
    <row cust_id="2025" cust_name="Can Do It Co."/>
    <row cust_id="2026" cust_name="City Corrections"/>
    <row cust_id="2027" cust_name="City Sports Arenas"/>
    <row cust_id="2028" cust_name="Cantaloupe Interiors"/>
    <row cust_id="2029" cust_name="Chicken Franchise"/>
   </table>
  </uldata>
 </ul:ulschema>
```


J2ME (BlackBerry スマートフォン) 用ユーティリティ

これらのユーティリティは、BlackBerry スマートフォン環境だけで使用できるように設計されています。

Ultra Light J データベース転送ユーティリティ (ULjDbT)

ULjDbT ユーティリティは、Ultra Light J データベースを BlackBerry スマートフォンから、デスクトップ、ラップトップ、サーバなどの外部デバイスに転送する機能を提供します。また、データベースを削除したり、データベース情報を表示したり、データベース転送ログを表示または電子メールで送信したりすることができます。このユーティリティは、Ultra Light J データベース転送デスクトップ・アプリケーション (ULjDbT) と BlackBerry スマートフォン・クライアント・アプリケーション (*ULjDatabaseTransfer.cod*) の 2 つのアプリケーションから構成され、これらのアプリケーションは同時に実行される必要があります。

Ultra Light J データベース転送デスクトップ・アプリケーション

デスクトップ・アプリケーションは USB 接続または HTTP 接続を使用して Ultra Light J データベースを受信します。サーバ・アプリケーションを起動すると、サーバ・アプリケーションは、クライアント・アプリケーションとの指定された接続を通じて BlackBerry スマートフォンからデータベースが転送されるのを待機します。接続は、アプリケーションがタイムアウトになったときにアプリケーション・インタフェースを使用して手動で切断されるか、または転送が完了したときに切断されます。

BlackBerry スマートフォン・クライアント・アプリケーション

BlackBerry スマートフォン・クライアント・アプリケーションは、USB ケーブル、またはデスクトップ・アプリケーションに対して指定された TCP ポートを使用して Ultra Light J データベースを送信します。

また、データベースを削除したり、データベース情報を表示したり、データベース転送ログを表示または電子メールで送信したりすることができます。

クライアント・アプリケーションは、SQL Anywhere インストール・ディレクトリの *UltraLite¥UltraLiteJ¥J2meRim11* ディレクトリにある署名付きファイルです。

♦ **クライアント・アプリケーションを起動するには、次の手順に従います。**

1. SQL Anywhere インストール・ディレクトリの *UltraLite¥UltraLiteJ¥J2meRim11* ディレクトリから *ULjDatabaseTransfer.cod* をロードします。

   クライアント・アプリケーションのアイコンがアプリケーションのリストに表示されます。
2. アプリケーションを起動し、トラックホイールを押します。
3. **[データベース接続]** 画面で、次のフィールドを完成させます。
   - **[データベース名]**　外部デバイスに転送するデータベースの名前。
   - **[データベースのパスワード]**　データ転送を許可するデータベース・パスワード。

4. **[次へ]** をクリックします。**[アクション]** 画面が表示されます。この画面からすべてのクライアント・アプリケーション機能にアクセスできます。

♦ BlackBerry スマートフォン・クライアント・アプリケーションを使用してデータベースを転送するには、次の手順に従います。

1. **[アクション]** 画面で、希望の接続方法 (USB または HTTP) を選択します。
2. USB 転送の場合は、**[USB を使用してデータベースをサーバに転送します]** を選択します。HTTP 転送の場合は、手順 4 に進みます。
3. 手順に従って、データベース転送デスクトップ・アプリケーションを起動します (後述の「Ultra Light J データベース転送アプリケーションを使用してデータベースを受信するには、次の手順に従います。」を参照)。

> **注意**
> データベースが正常に転送されるためには、デバイスまたはシミュレータが BlackBerry Device Manager に接続されていることを確認してください。シミュレータの場合は、**[USB Cable Connected]** を使用して USB 接続がシミュレートされていることを確認してください。

   a. デスクトップ・アプリケーションで、**[USB]** が選択されていることを確認して **[開始]** をクリックします。
   b. クライアント・アプリケーションで **[次へ]** をクリックします。
   c. BlackBerry スマートフォンで、外部デバイスへのデータベースの転送が開始されます。進行状況はデスクトップ・アプリケーションに表示されます。
   d. クライアント・アプリケーションとデスクトップ・アプリケーションの両方で **[OK]** をクリックして閉じます。
4. HTTP 転送の場合は、**[HTTP を使用してデータベースをサーバに転送します]** を選択します。
   a. **[HTTP 転送]** 画面で、**[次へ]** をクリックします。
   b. 次の値を指定します。
      - **[ホスト]** デスクトップの IP アドレス。
      - **[ポート]** デスクトップ・アプリケーションの **[接続プロパティ]** で指定されたポート。
      - **[URL サフィックス]** 転送を受信するサーバのホスト名で、http:// サフィックスを含む (このサフィックスは必須)。

      [次へ] をクリックします。

注意
個人特定不可デバイス上の BES を介して転送する場合、**[URL サフィックス]** は空のままにします。個人特定可能デバイスでは、サフィックス **;deviceside=false** を使用します。

ダイレクト TCP 経由で転送する場合、サフィックス **;deviceside=true** を使用します。これをサポートしない通信事業者もあります。

通信事業者の WAP ゲートウェイについて APN 情報がわかっている場合は、その WAP ゲートウェイを使用できることがあります。その情報もサフィックスに付加する必要があります。BES 経由の場合であっても、Ultra Light J データベース転送ユーティリティを実行しているコンピュータと BES との間にファイアウォールがある可能性があります。その場合は、SSL トンネルを使用する必要があります。**[HTTP Transfer Parms]** 画面で、ファイアウォールの BES 側で実行している SSL サーバのポートと名前または IP アドレスを指定します。また、転送アプリケーションに SSL クライアントがマッピングされているポートを指定する必要もあります。

データベースを BlackBerry シミュレータから転送する場合は、BlackBerry MDS シミュレータが実行されているか、または BlackBerry シミュレータで実行されている Ultra Light J データベース転送ユーティリティに対して URL サフィックス **;deviceside=true** を指定する必要があります。

c. デスクトップ・アプリケーションで、**[HTTP]** が選択されていることを確認して **[開始]** をクリックします。

d. クライアント・アプリケーションで **[次へ]** をクリックします。

   BlackBerry スマートフォンで、外部デバイスへのデータベースの転送が開始されます。進行状況はデスクトップ・アプリケーションに表示されます。

e. クライアント・アプリケーションとデスクトップ・アプリケーションの両方で **[OK]** をクリックして閉じます。

♦ Ultra Light J データベース転送アプリケーションを使用してデータベースを受信するには、次の手順に従います。

1. SQL Anywhere のインストール・ディレクトリの *Bin32* ディレクトリから *ULjDbTServ.cmd* を実行します。

   Ultra Light J データベース転送アプリケーションがロードされます。

2. **[接続]** タブで、**[接続方法]** を選択します。
3. **[接続プロパティ]** の下で次の値を指定します。
   - **[ポート]** このフィールドは、HTTP 接続にだけ適用されます。BlackBerry スマートフォンが接続する TCP ポート番号を入力します。通常、このポート番号は BlackBerry デバイス上で動作する Ultra Light J データベース転送ユーティリティに指定されているポート番号と一致します。ただし、SSL を使用している場合、ポート番号は異なる場合があります。
   - **[Blackberry のパスワード]** このフィールドは、USB 接続にだけ適用されます。接続されている BlackBerry スマートフォンがロックされているときにアクセスするためのパスワードを入力します。パスワードがない場合はこのフィールドを空白のままにします。

- **[タイムアウト]**　サーバ・アプリケーションがタイムアウトし、接続を切断するまでの分単位のアイドル時間。
- **[出力]**　転送されたデータベースを保存するファイル名とロケーションを指定します。

4. **[開始]** をクリックして BlackBerry スマートフォンへの接続を開きます。サーバ・アプリケーションは、タイムアウトするか、接続が確立するまで待機します。既存のファイルを指定した場合は、そのファイルを上書きするかどうか尋ねられます。

[ログ] タブには、エラー・メッセージを含む、サーバの状態と転送の進行状況に関する詳細が表示されます。

♦ データベースを削除するには、次の手順に従います。

1. **[アクション]** 画面で、**[データベースを削除します]** を選択します。
2. 確認のダイアログで、**[削除]** をクリックしてデータベースを削除します。
3. **[データベースは削除されました。]** ダイアログで、**[OK]** をクリックしてクライアントを閉じます。

♦ データベース情報を表示するには、次の手順に従います。

1. **[アクション]** 画面で、**[データベース情報を表示します]** を選択します。スクロールしてすべてのデータベース情報を確認します。
2. **[前へ]** をクリックして **[アクション]** 画面に戻ります。

♦ ログ・ファイルを表示するには、次の手順に従います。

1. **[データベース接続]** 画面で、メニューを表示します。
2. **[ログ]** をクリックします。ログ画面が表示されます。
3. ログ・ファイルを電子メールで送信するには、ログの送信先の電子メール・アドレスを入力して、**[電子メールを送信します]** をクリックします。前の画面に戻るには、リターン・キーを押します。

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」 296 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」 301 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」 305 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)」 305 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」 313 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」 293 ページ
- 「jConnect」 293 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」 293 ページ
- 「iAnywhere JDBC ドライバ」 292 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」 293 ページ
- 「iAnywhere JDBC ドライバ」 292 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」 301 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」 296 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」 320 ページ
- 「同期」 320 ページ
- 「Ultra Light」 297 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」 301 ページ
- 「Listener」 293 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」 295 ページ
- 「サーバ起動同期」 301 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」 301 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」 291 ページ。

Replication Agent

参照：「LTM」 293 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」 293 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」 303 ページ
- 「SQL」 296 ページ
- 「データベース管理システム (DBMS)」 305 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」 297 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」 313 ページ
- 「パブリケーション」 308 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」 307 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」 300 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」 321 ページ
- 「コード・ページ」 301 ページ
- 「照合」 318 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」 297 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」 300 ページ
- 「カーソル位置」 299 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」 299 ページ
- 「カーソル結果セット」 300 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」 299 ページ
- 「カーソル位置」 299 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」 296 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」 306 ページ
- 「ローカル・テンポラリ・テーブル」 314 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」 301 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」 321 ページ
- 「エンコード」 299 ページ
- 「照合」 318 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」 308 ページ
- 「Mobile Link ユーザ」 294 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」 302 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」 302 ページ
- 「生成されたジョイン条件」 319 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」 298 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」 321 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」 320 ページ
- 「SQL ベースの同期」 297 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」 315 ページ
- 「プライマリ・キー」 310 ページ
- 「データベース管理システム (DBMS)」 305 ページ
- 「リレーショナル・データベース管理システム (RDBMS)」 313 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」 291 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)」 313 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)」 305 ページ
- 「SYS」 297 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」 305 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」 306 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」 301 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」 314 ページ
- 「グローバル・テンポラリ・テーブル」 300 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」 306 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」 307 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」 314 ページ
- 「文レベルのトリガ」 321 ページ
- 「整合性」 318 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」 308 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」 313 ページ
- 「アーティクル」 298 ページ
- 「パブリケーションの更新」 308 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」 313 ページ
- 「パブリケーション」 308 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」 313 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」 318 ページ
- 「ユーザ定義データ型」 312 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」 309 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」 315 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」 318 ページ
- 「検査制約」 317 ページ
- 「外部キー制約」 316 ページ
- 「一意性制約」 315 ページ
- 「整合性」 318 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」 297 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」 298 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」 311 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」 306 ページ
- 「ビュー」 309 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」 301 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」 311 ページ
- 「ビュー」 309 ページ

ミラー・ログ

参照：「トランザクション・ログ・ミラー」 307 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」 303 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」 313 ページ
- 「FILE」 292 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」 300 ページ
- 「サーバ・メッセージ・ストア」 301 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」 313 ページ
- 「統合データベース」 320 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」 306 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」 301 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」 320 ページ
- 「統合データベース」 320 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)」 305 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」 313 ページ
- 「パブリケーションの更新」 308 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」 313 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」 307 ページ
- 「文レベルのトリガ」 321 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」 306 ページ
- 「グローバル・テンポラリ・テーブル」 300 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」 307 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」 315 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」 307 ページ
- 「トランザクション・ログ・ミラー」 307 ページ
- 「フル・バックアップ」 310 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」 321 ページ
- 「同時性 (同時実行性)」 321 ページ
- 「整合性」 318 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」 316 ページ
- 「プライマリ・キー制約」 310 ページ
- 「制約」 318 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」 310 ページ
- 「外部テーブル」 316 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」 318 ページ
- 「検査制約」 317 ページ
- 「プライマリ・キー制約」 310 ページ
- 「一意性制約」 315 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」 302 ページ
- 「内部ジョイン」 321 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」 315 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」 318 ページ
- 「外部キー制約」 316 ページ
- 「プライマリ・キー制約」 310 ページ
- 「一意性制約」 315 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」 315 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」 310 ページ
- 「外部キー」 315 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」 310 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」 321 ページ
- 「コード・ページ」 301 ページ
- 「エンコード」 299 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」 309 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」 317 ページ
- 「外部キー制約」 316 ページ
- 「プライマリ・キー制約」 310 ページ
- 「一意性制約」 315 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」 317 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」 302 ページ
- 「ジョイン条件」 303 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」 301 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」 313 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」 320 ページ
- 「レプリケーション」 313 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」 293 ページ
- 「SQL Remote」 296 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」 307 ページ
- 「独立性レベル」 321 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」 303 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」 302 ページ
- 「外部ジョイン」 316 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」 307 ページ
- 「ロー・レベルのトリガ」 314 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」 301 ページ
- 「エンコード」 299 ページ
- 「照合」 318 ページ

文字列リテラル

文字列リテラルとは、一重引用符 (') で囲まれ、シーケンスで並べられた文字のことです。

論理インデックス

物理インデックスへの参照 (ポインタ) です。ディスクに保存される論理インデックス用のインデックス構造はありません。

索引

記号

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

あ

い

う

え

お

か

き

く

す

せ

そ

た

ち

つ

て

と

な

に

ね

は

ひ

ふ

へ

ほ

ま

み

め

