iAnywhere SOLUTIONS™
A SYBASE COMPANY

Native Ultra Light for Java
ユーザーズ・ガイド

パート番号：DC37124-01-0902-01
改訂：2005 年 3 月

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、Native Ultra Light for Java について説明します。Native Ultra Light for Java を使用すると、Jeode VM または CrEme VM を実行している Windows CE 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。

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

SQL Anywhere Studio のマニュアル

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

SQL Anywhere Studio のマニュアル

SQL Anywhere Studio のマニュアルは、各マニュアルを 1 つの大きなヘルプ・ファイルにまとめたオンライン形式、マニュアル別の PDF ファイル、および有料の製本版マニュアルで提供されます。SQL Anywhere Studio のマニュアルは、次の分冊マニュアルで構成されています。

- **『SQL Anywhere Studio の紹介』** このマニュアルでは、SQL Anywhere Studio のデータベース管理と同期テクノロジの概要について説明します。また、SQL Anywhere Studio を構成する各部分について説明するチュートリアルも含まれています。
- **『SQL Anywhere Studio 新機能ガイド』** このマニュアルは、SQL Anywhere Studio のこれまでのリリースのユーザを対象としています。ここでは、製品の今回のリリースと以前のリリースで導入された新機能をリストし、アップグレード手順を説明しています。
- **『Adaptive Server Anywhere データベース管理ガイド』** このマニュアルでは、データベースおよびデータベース・サーバの実行、管理、設定について説明しています。
- **『Adaptive Server Anywhere SQL ユーザーズ・ガイド』** 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。
- **『Adaptive Server Anywhere SQL リファレンス・マニュアル』** このマニュアルは、Adaptive Server Anywhere で使用する SQL 言語の完全なリファレンスです。また、Adaptive Server Anywhere のシステム・テーブルとシステム・プロシージャについても説明しています。
- **『Adaptive Server Anywhere プログラミング・ガイド』** このマニュアルでは、C、C++、Java 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。Visual Basic や PowerBuilder などのツールのユーザは、それらのツールのプログラミング・インタフェースを使用できます。また、Adaptive Server Anywhere ADO.NET データ・プロバイダについても説明します。

- **『Adaptive Server Anywhere SNMP Extension Agent ユーザーズ・ガイド』** このマニュアルでは、SNMP 管理アプリケーションとともに使用するように Adaptive Server Anywhere SNMP Extension Agent を設定して、Adaptive Server Anywhere データベースを管理する方法を説明します。

- **『Adaptive Server Anywhere エラー・メッセージ』** このマニュアルでは、Adaptive Server Anywhere エラー・メッセージの完全なリストを、その診断情報とともに説明します。

- **『SQL Anywhere Studio セキュリティ・ガイド』** このマニュアルでは、Adaptive Server Anywhere データベースのセキュリティ機能について説明します。Adaptive Server Anywhere 7.0 は、米国政府から TCSEC (Trusted Computer System Evaluation Criteria) の C2 セキュリティ評価を授与されています。このマニュアルには、Adaptive Server Anywhere の現在のバージョンを、C2 基準を満たした環境と同等の方法で実行することを望んでいるユーザにとって役に立つ情報が含まれています。

- **『Mobile Link 管理ガイド』** このマニュアルでは、モバイル・コンピューティング用の Mobile Link データ同期システムについてあらゆる角度から説明します。このシステムによって、Oracle、Sybase、Microsoft、IBM の単一データベースと、Adaptive Server Anywhere や Ultra Light の複数データベースの間でのデータ共有が可能になります。

- **『Mobile Link クライアント』** このマニュアルでは、Adaptive Server Anywhere リモート・データベースと Ultra Light リモート・データベースの設定を行い、これらを同期させる方法について説明します。

- **『Mobile Link サーバ起動同期ユーザーズ・ガイド』** このマニュアルでは、Mobile Link のサーバによって開始される同期について説明します。サーバによって開始される同期とは、統合データベースから同期の開始を可能にする Mobile Link の機能です。

- **『Mobile Link チュートリアル』** このマニュアルには、Mobile Link アプリケーションの設定と実行を行う方法を説明するチュートリアルがいくつか用意されています。
- **『QAnywhere ユーザーズ・ガイド』** このマニュアルでは、Mobile Link QAnywhere について説明します。Mobile Link QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアントだけでなく、モバイル・クライアントや無線クライアント用のメッセージング・アプリケーションの開発と展開を可能にするメッセージング・プラットフォームです。
- **『Mobile Link およびリモート・データ・アクセスの ODBC ドライバ』** このマニュアルでは、Mobile Link 同期サーバから、または Adaptive Server Anywhere リモート・データ・アクセスによって、Adaptive Server Anywhere 以外の統合データベースにアクセスするための ODBC ドライバの設定方法について説明します。
- **『SQL Remote ユーザーズ・ガイド』** 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、あらゆる角度から説明します。このシステムによって、Adaptive Server Anywhere または Adaptive Server Enterprise の単一データベースと Adaptive Server Anywhere の複数データベースの間で、電子メールやファイル転送などの間接的リンクを使用したデータ共有が可能になります。
- **『SQL Anywhere Studio ヘルプ』** このマニュアルには、Sybase Central や Interactive SQL、その他のグラフィカル・ツールに関するコンテキスト別のヘルプが含まれています。これは、製本版マニュアル・セットには含まれていません。
- **『Ultra Light データベース・ユーザーズ・ガイド』** このマニュアルは、Ultra Light 開発者を対象としています。ここでは、Ultra Light データベース・システムの概要について説明します。また、すべての Ultra Light プログラミング・インタフェースに共通する情報を提供します。
- **Ultra Light のインタフェースに関するマニュアル** 各 Ultra Light プログラミング・インタフェースには、それぞれに対応するマニュアルを用意しています。これらのインタフェースは、RAD (

ラピッド・アプリケーション開発) 用の Ultra Light コンポーネントとして提供されているものと、C、C++、Java 開発用の静的インタフェースとして提供されているものがあります。

このマニュアル・セットの他に、PowerDesigner と InfoMaker には、独自のオンライン・マニュアル (英語版) がそれぞれ用意されています。

マニュアルの形式

SQL Anywhere Studio のマニュアルは、次の形式で提供されています。

- **オンライン・マニュアル** オンライン・マニュアルには、SQL Anywhere Studio の完全なマニュアルがあり、SQL Anywhere ツールに関する印刷マニュアルとコンテキスト別のヘルプの両方が含まれています。オンライン・マニュアルは、製品のメンテナンス・リリースごとに更新されます。これは、最新の情報を含む最も完全なマニュアルです。

  Windows オペレーティング・システムでオンライン・マニュアルにアクセスするには、[スタート] ー [プログラム] ー [SQL Anywhere 9] ー [オンライン・マニュアル] を選択します。オンライン・マニュアルをナビゲートするには、左ウィンドウ枠で HTML ヘルプの目次、索引、検索機能を使用し、右ウィンドウ枠でリンク情報とメニューを使用します。

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

- **PDF 版マニュアル** SQL Anywhere の各マニュアルは、Adobe Acrobat Reader で表示できる PDF ファイルで提供されています。

  PDF 版マニュアルは、オンライン・マニュアルまたは Windows の [スタート] メニューから利用できます。

- **製本版マニュアル** 製本版マニュアルをご希望の方は、ご購入いただいた販売代理店または弊社営業担当までご連絡ください。

表記の規則

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

SQL 構文の表記規則

SQL 構文の表記には、次の規則が適用されます。

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

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

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

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

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

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

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

- **オプション部分** 文のオプション部分は角カッコで囲みます。

  RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

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

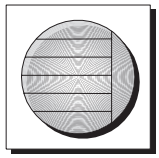
  QUOTES オプションを使用する場合は、ON または OFF のどちらかを選択する必要があります。角カッコと中カッコは入力しないでください。

グラフィック・アイコン

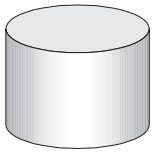
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

- Sybase Adaptive Server Anywhere などのデータベース・サーバ



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



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

- プログラミング・インタフェース

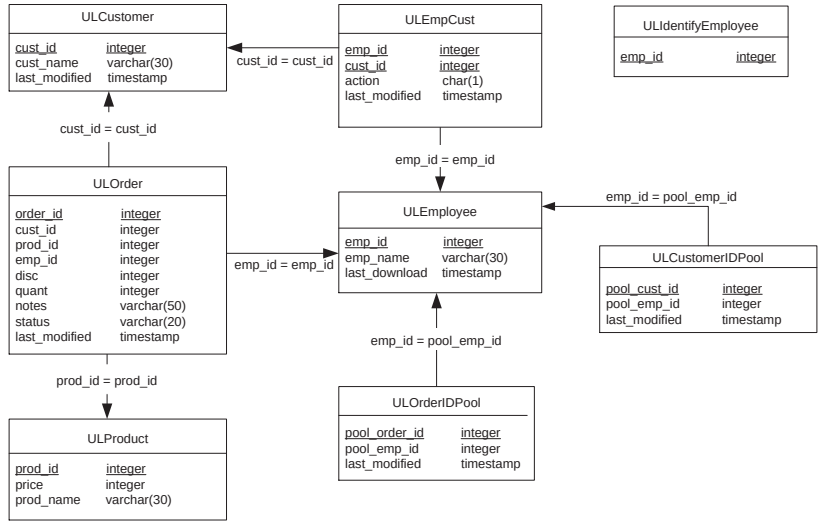
CustDB サンプル・データベース

Mobile Link と Ultra Light のマニュアルでは、多くの例で Ultra Light のサンプル・データベースが使用されています。

Ultra Light サンプル・データベースのリファレンス・データベースは、*custdb.db* という名前のファイルに保存され、SQL Anywhere ディレクトリのサブディレクトリ *Samples¥UltraLite¥CustDB* に置かれています。全面的にこのデータベースを使用して構築したアプリケーションも提供されています。

サンプル・データベースは、あるハードウェア販売会社の販売管理データベースです。データベースには、この販売会社の顧客、製品、営業戦力に関する情報が入っています。

次の図は、CustDB データベース内のテーブルと、各テーブル間の関係を示しています。



詳細情報の検索／フィードバックの提供

詳細情報の検索

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

ご質問がある場合や支援が必要な場合は、次に示す iAnywhere Solutions 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 Studio バージョンのビルド番号を明記し、現在発生している問題について詳しくお知らせくださいますようお願いいたします。バージョン情報は、コマンド・プロンプトで **dbeng9 -v** と入力して確認できます。

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

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

ニュースグループに関するお断り

iAnywhere Solutions は、ニュースグループ上に解決策、情報、または意見を提供する義務を負うものではありません。また、システム・オペレータ以外のスタッフにこのサービスを監視させて、操作状況や可用性を保証する義務もありません。

iAnywhere Solutions のテクニカル・アドバイザとその他のスタッフは、時間のある場合にかぎりニュースグループでの支援を行います。こうした支援は基本的にボランティアで行われるため、解決策や情報を定期的に提供できるとはかぎりません。支援できるかどうかは、スタッフの仕事量に左右されます。

フィードバック

このマニュアルに関するご意見、ご提案、フィードバックをお寄せください。

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

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

第 1 章

Native Ultra Light for Java の概要

この章の内容

この章では、Native Ultra Light for Java について説明します。ここでは、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light へようこそ」で説明した Ultra Light の機能について理解していることを前提としています。

Native Ultra Light for Java を使用したアプリケーションの作成の詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java 開発の概要」を参照してください。

Native Ultra Light for Java の概要のチュートリアルについては、『Native Ultra Light for Java ユーザーズ・ガイド』>「チュートリアル：Native Ultra Light for Java アプリケーションの構築」または『Native Ultra Light for Java ユーザーズ・ガイド』>「チュートリアル：CustDB サンプル・アプリケーション」を参照してください。

Ultra Light と Java

Ultra Light データベース機能を利用する Java 開発者には、2 つのオプションがあります。

- 静的型 Java API は、『Ultra Light 静的型 Java ユーザーズ・ガイド』>「Ultra Light 静的型 Java ユーザーズ・ガイド」に説明されている pure Java Ultra Light テクノロジです。
- Native Ultra Light for Java (このマニュアルで説明) では、Native Ultra Light for Java パッケージと、ネイティブ (C++) Ultra Light ランタイム・ライブラリが提供されています。これによって、Java 開発の利点と、ネイティブ・アプリケーションのパフォーマンスと ActiveSync 同期などのオペレーティング・システム固有の機能へのアクセスが組み合わせられます。

Native Ultra Light for Java は、Ultra Light for Java と次のように異なります。

- **ネイティブ・コンポーネント** Native Ultra Light for Java の Ultra Light ランタイムは、ネイティブ・メソッドを使用します。つまり、Java ではなく、C++ で記述され、基本となる CPU とオペレーティング・システムに固有のバイナリ・フォームにコンパイルされています。一方、静的型 Java API の Ultra Light ランタイムは、pure Java の実装です。
- **コンポーネント API** Native Ultra Light for Java は、Ultra Light コンポーネント・パッケージの他のメンバと API の機能と構造を共有しています。静的型 Java API は、プログラミング・インタフェースとして JDBC を使用します。
- **Windows CE への配備** Native Ultra Light for Java は、Windows CE を配備のターゲットとしています。多くの Windows CE デバイスに付属の Jeode VM と CrEme VM をサポートしています。また、ActiveSync 同期もサポートしています。

  プラットフォームのサポートについては、『SQL Anywhere Studio の紹介』>「Ultra Light ターゲット・プラットフォーム」を参照してください。

Ultra Light データベースの機能の詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light データベース」を参照してください。

システムの稼働条件とサポートされるプラットフォーム

開発プラットフォーム

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

- Microsoft Windows NT/2000/XP
- Native Ultra Light for Java アプリケーション開発には、JDK 1.1.8 または Personal Java 1.2 を推奨します。Borland JBuilder 7 と 8 が統合されます。

ターゲット・プラットフォーム

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

- ARM プロセッサを使用するデバイス上で動作する Windows CE 3.0 以降 (Compaq iPaq、NEC MobilePro P300、および Jeode Personal Java 1.2 と互換性のある VM を含む)。
- Windows NT ／ 2000 ／ XP と JRE 1.1.8 以降の組み合わせ。

詳細については、『SQL Anywhere Studio の紹介』> 「Ultra Light 開発プラットフォーム」と『SQL Anywhere Studio の紹介』> 「Ultra Light ターゲット・プラットフォーム」を参照してください。

Native Ultra Light for Java のアーキテクチャ

Native Ultra Light for Java パッケージの名前は、ianywhere.native_ultralite です。

よく使用される高度なクラスの一部を次のリストに示します。

- **DatabaseManager**　アプリケーションごとに 1 つの DatabaseManager オブジェクトを作成します。
- **Connection**　各 Connection オブジェクトは、Ultra Light データベースとの接続を表します。Connection オブジェクトは 1 つまたは複数作成できます。
- **Table**　Table オブジェクトは、データベース内のデータへのアクセスを提供します。
- **PreparedStatement、ResultSet、ResultSetSchema**　これらの動的 SQL オブジェクトを使って、動的 SQL 文の作成、クエリの記述、INSERT、UPDATE、および DELETE 文の実行、およびプログラムによるデータベースの結果セットの制御ができます。
- **SyncParms**　SyncParms オブジェクトを使用して、Ultra Light データベースを Mobile Link 同期サーバと同期させます。

『API リファレンス』は、SQL Anywhere インストール環境の *UltraLite¥NativeUltraLiteForJava¥doc* サブディレクトリに Javadoc フォーマットで提供されています。API リファレンスへのアクセスの詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」を参照してください。

第 2 章

チュートリアル：Native Ultra Light for Java アプリケーションの構築

この章の内容

この章は、Native Ultra Light for Java アプリケーションのサンプルを構築するプロセスを解説するチュートリアルです。

概要

このチュートリアルは、Native Ultra Light for Java アプリケーションを構築するプロセスを説明します。

このチュートリアルでは、テキスト・エディタを使用して Java ファイルを編集します。Borland JBuilder 開発環境でも Native Ultra Light for Java を使用できます。

詳細については、「Borland JBuilder でのアプリケーション開発」74 ページを参照してください。

所要時間

このチュートリアルは、コードをコピーして貼り付ける場合、約 30 分で終了します。自分でコードを入力する場合、これより長い時間が必要です。

能力と経験

このチュートリアルは、次のことを前提にしています。

- Java プログラミング言語に精通している
- JDK 1.1.8 または Personal Java 1.2 がコンピュータにインストールされている
- Ultra Light スキーマ・ペインタを使用して、Ultra Light スキーマを作成する方法を知っている

  詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「レッスン 1：Ultra Light データベース・スキーマの作成」を参照してください。

注意
このチュートリアルの同期に関する操作では、SQL Anywhere Studio が必要です。

目的

このチュートリアルの目的は、Native Ultra Light for Java アプリケーションの開発プロセスについて、知識と経験を得ることです。

レッスン 1：データベースへの接続

最初に、データベース・スキーマを作成します。次に、前のレッスンで作成したスキーマを使用して、データベースを作成したり既存のデータベースに接続する Java アプリケーションを記述、コンパイル、実行します。

❖ データベース・スキーマを作成するには、次の手順に従います。

1　このチュートリアルで作成されるファイルを保存するディレクトリを作成します。

このチュートリアルでは、保存先ディレクトリを *c:¥tutorial¥java* とします。別の名前のディレクトリを作成した場合は、チュートリアルを通じて、*c:¥tutorial¥java* の代わりにそのディレクトリを使用してください。

2　Ultra Light スキーマ・ペインタを使用して、以下の特性を持つデータベース・スキーマを作成します。

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「レッスン 1：Ultra Light データベース・スキーマの作成」を参照してください。

スキーマ・ファイルの名前 : **tutcustomer.usm**

テーブル名 : **Customer**

Customer のカラム :

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

カラム名	データ型 (サイズ)	カラムの NULL 値の許可	デフォルト値
Phone	char(12)	はい	555-1234

プライマリ・キー : 昇順 **ID**

❖ Ultra Light データベースに接続するには、次の手順に従います。

1 チュートリアルのディレクトリに、*Customer.java* という名前のファイルを作成します。

2 以下のコードを *Customer.java* にコピーします。

このコードでは、次のタスクを実行します。

- Ultra Light ライブラリと JDBC SQLException クラスをインポートします。
- Customer という名前のクラスを宣言します。
- データベース接続オブジェクトを格納する静的変数を宣言します。このオブジェクトは、チュートリアルの後半で、複数のメソッドの間で共有されます。
- クラス・コンストラクタを呼び出します。
- エラーが発生した場合は、エラー・コードとスタック・トレースを出力します。

  エラー・コードの詳細については、『ASA エラー・メッセージ』> 「ASA エラー・メッセージ」を参照してください。

```
import ianywhere.native_ultralite.*;
 import java.sql.SQLException;

 public class Customer {
   static Connection conn;
   public static void main( String args[] ) {
     try {
```

```
        Customer cust = new Customer();
        // Clean up
        conn.close();
      } catch( SQLException e ) {
        System.out.println(
          "Exception: " + e.getMessage() +
          " sqlcode=" + e.getErrorCode()
            );
        e.printStackTrace();
      }
    }
}
```

3　クラスにコンストラクタを追加します。

以下のコードをファイルにコピーします。このコードでは、次のタスクを実行します。

- 新しい DatabaseManager オブジェクトをインスタンス化します。Ultra Light オブジェクトはすべて、DatabaseManager オブジェクトから作成されます。
- 新しい CreateParms オブジェクトをインスタンス化して、定義します。CreateParms は、データベースへの接続またはデータベースの構築に必要なパラメータを格納します。この場合、パラメータは、デスクトップ上のデータベース・ファイルとスキーマ・ファイルのロケーションです。

  この例では、便宜的にロケーションはハードエンコードされています。実際のアプリケーションでは、ハードエンコードされていません。また、これらのパラメータによって満たされるのは開発環境の接続だけです。Windows CE デバイスでアプリケーションを実行するには、追加のパラメータが必要です。追加のパラメータについては、この後のチュートリアルで説明します。
- データベース・ファイルが存在しない場合は、SQLException がスローされます。この例外を取得するコードがスキーマ・ファイルを使用して新規データベースを作成し、データベースへの接続を確立します。
- データベース・ファイルが存在する場合は、接続が確立されます。

```
public Customer() throws SQLException
 {
   // Connect
   DatabaseManager dbMgr = new DatabaseManager();
   CreateParms parms = new CreateParms();
   parms.databaseOnDesktop =
"c:¥¥tutorial¥¥java¥¥tutcustomer.udb";
   parms.schema.schemaOnDesktop =
"c:¥¥tutorial¥¥java¥¥tutcustomer.usm";
   try {
     conn = dbMgr.openConnection( parms );
     System.out.println(
             "Connected to an existing database." );
   } catch( SQLException econn ) {
     if( econn.getErrorCode() ==
         SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND
){
       conn = dbMgr.createDatabase( parms );
       System.out.println(
             "Connected to a new database." );
     } else {
       throw econn;
     }
   }
 }
```

4 Customer クラスをコンパイルします。

Sun JDK 1.1.8 または Personal Java 1.2 を使用してクラスをコンパイルすることをおすすめします。また、Ultra Light ライブラリ *jul9.jar* をクラスパスに追加してください。このライブラリは、SQL Anywhere インストール環境の *ultralite¥NativeUltraLiteForJava* サブディレクトリにあります。

コマンド・プロンプトに次のコマンドを 1 行で入力します。

```
javac -classpath

"%ASANY9%¥ultralite¥NativeUltraLiteForJava¥jul9.jar
;%classpath%"
 Customer.java
```

5 アプリケーションを実行します。

前の手順で示したように、クラスパスには Ultra Light ライブラリ *jul9.jar* が含まれていなければなりません。

また、アプリケーションは、Ultra Light ネイティブ・メソッドが保存されている DLL もロードできなければなりません。DLL は、SQL Anywhere 9 インストール環境の *ultralite¥NativeUltraLiteForJava¥win32* サブディレクトリにある *jul9.dll* です。この DLL は、システム・パスに置くことも、java コマンド・ラインで、次のように指定することもできます。

```
java -classpath

".;%ASANY9%¥ultralite¥NativeUltraLiteForJava¥jul9.j
ar"
 -Djul.library.dir=
 "%ASANY9%¥ultralite¥NativeUltraLiteForJava¥win32"
Customer
```

このコマンドは全体を 1 行で入力し、個々の引数の中にスペースを入れないでください。

レッスン 2：データベースへのデータの挿入

次の手順は、データベースにデータを追加する方法を示します。

❖ データベースにローを追加するには、次の手順に従います。

1 *Customer.java* ファイルに次のメソッドを追加します。

このメソッドでは、次のタスクを実行します。

- テーブルを開きます。テーブルの操作を実行するには、Table オブジェクトを開いてください。Table オブジェクトを取得するには、**connection.getTable()** メソッドを使用します。
- テーブルの一部のカラムの識別子を取得します。テーブルのその他のカラムは NULL 値を受け入れるか、デフォルト値を持つことができます。このチュートリアルでは、必要な値のみが指定されています。
- テーブルが空の場合は、ローを 2 つ追加します。各ローを挿入するために、InsertBegin を使用してモードを挿入モードに設定し、必要な各カラムに値を設定し、挿入を実行してデータベースにローを追加します。

  アプリケーションのデフォルト・モードでは、各挿入の後に操作をコミットするように設定されているため、ここでは commit メソッドは必須ではありません。オートコミット動作をオフにした場合、変更内容を永続的に保存するには変更をコミットしなければならないことを強調するためにコードに追加されています。
- テーブルが空でない場合は、テーブルのロー数をレポートします。
- Table オブジェクトを閉じます。

```
private void insert() throws SQLException
 {
   // Open the Customer table
   Table t = conn.getTable( "Customer" );
   t.open();

   short id = t.schema.getColumnID( "ID" );
   short fname = t.schema.getColumnID( "FName" );
   short lname = t.schema.getColumnID( "LName" );

   // Insert two rows if the table is empty
   if( t.getRowCount() == 0 ) {

       t.insertBegin();
       t.setString( fname, "Gene" );
       t.setString( lname, "Poole" );
       t.insert();

       t.insertBegin();
       t.setString( fname, "Penny" );
       t.setString( lname, "Stamp" );
       t.insert();

       conn.commit();
       System.out.println( "Two rows inserted." );
   } else {
       System.out.println( "The table has " +
       t.getRowCount() + " rows." );
   }
   t.close();
 }
```

2　作成した insert メソッドを呼び出します。

次の行を、`main()` メソッドの Customer コンストラクタの呼び出しの直後に追加します。

```
cust.insert();
```

3　アプリケーションをコンパイルし、実行します。詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「レッスン 1：データベースへの接続」を参照してください。

レッスン 3：テーブルのローの選択

次の手順は、テーブルのローをループする方法を示します。この手順では、テーブルからローを取り出し、コマンド・ラインに出力します。

❖ テーブルのローをリストするには、次の手順に従います。

1 *Customer.java* ファイルに次のメソッドを追加します。

このメソッドでは、次のタスクを実行します。

- Table オブジェクトを開きます。
- カラム識別子を取得します。
- 現在の位置を、テーブル内にある最初のローの前に設定します。

  テーブルに対するすべての操作は、現在の位置で実行されます。現在の位置は、最初のローの前、テーブルのいずれかのローの上、または最後のローの後ろです。この例のように、デフォルトでは、ローはプライマリ・キー値 (id) に基づいて順序付けられています。別の順序でローを並べ替えるには、データベースに新しいインデックスを追加し、そのインデックスを使用してテーブルを開きます。
- 各ローに対して、id と名前が書き出されます。ループは、最後のローの後に **moveNext** が false を返すまで繰り返されます。
- Table オブジェクトを閉じます。

```
private void select() throws SQLException
 {
   // Fetch rows
   // Open the Customer table
   Table t = conn.getTable( "Customer" );
   t.open();
   short id = t.schema.getColumnID( "ID" );
   short fname = t.schema.getColumnID( "FName" );
```

```
    short lname = t.schema.getColumnID( "LName" );
    t.moveBeforeFirst();
    while( t.moveNext() ) {
        System.out.println(
          "id= " + t.getInt( id )
          + ", name= " + t.getString( fname )
          + " " + t.getString( lname )
          );
    }
    t.close();
  }
```

2　作成した select 関数を呼び出します。

次の行を、main() メソッドの insert メソッドの呼び出しの直後に追加します。

```
cust.select();
```

3　アプリケーションをコンパイルし、実行します。詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「レッスン 1：データベースへの接続」を参照してください。

レッスン 4：Windows CE デバイスへのアプリケーションの配備

Windows CE でサンプル・アプリケーションを実行するには、サポートされている Java VM がデバイスにインストールされている必要があります。

Jeode Runtime (Java VM) は、HP/Compaq iPAQ や NEC MobilePro P300 を含むいくつかのデバイスにパッケージ化されています。また、CrEme Plus も使用でき、Symbol 付加価値機能がバンドルされた CrEme が含まれています。これらの VM は両方とも他のデバイスに対して購入できます。

❖ Windows CE デバイスに Jeode VM がインストールされていることを確認するには、次の手順に従います。

- [スタート] ー [プログラム] を選択します。

  Jeode VM がインストールされている場合、[Jeode] フォルダが [プログラム] フォルダに表示されます。

❖ Windows CE デバイスで実行するアプリケーションを準備するには、次の手順に従います。

1. schemaOnCE および databaseOnCE パラメータを指定し、データベース・ファイルとスキーマ・ファイルのロケーションを指定します。

   Customer コンストラクタの CreateParms オブジェクトを次のように変更します。

```
CreateParms parms = new CreateParms();
 parms.databaseOnDesktop =
"c:¥¥tutorial¥¥java¥¥tutcustomer.udb";
 parms.databaseOnCE =
"¥¥UltraLite¥¥tutorial¥¥tutcustomer.udb";
 parms.schema.schemaOnDesktop =
"c:¥¥tutorial¥¥java¥¥tutcustomer.usm";
 parms.schema.schemaOnCE =
"¥¥UltraLite¥¥tutorial¥¥tutcustomer.usm"";
```

2　アプリケーションをコンパイルします。詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「レッスン 1：CustDB アプリケーションの構築」を参照してください。

3　アプリケーションを実行して、エラーがないことを確認します。schemaOnCE および databaseOnCE パラメータは、デスクトップ環境で実行しても影響ありません。

4　アプリケーションを実行するショートカットを準備します。

ショートカットは、アプリケーションを実行するコマンド・ラインを持つ拡張子が *.lnk* のテキスト・ファイルです。次の手順では、このショートカット・ファイルをデバイス上のロケーションにコピーします。

テキスト・エディタを使用して、チュートリアル・ディレクトリに次の内容のファイル *tutorial.lnk* を作成します。内容はすべて 1 行で入力してください。

```
18#"¥Windows¥evm.exe"
 -Djeode.evm.console.local.keep=TRUE
 -Djeode.evm.console.local.paging=TRUE
 -Djul.library.dir=¥UltraLite¥lib
 -cp ¥UltraLite¥tutorial;¥UltraLite¥lib¥jul9.jar
 Customer
```

このコマンドの要素の意味は、次のとおりです。

- 最初の行は、Jeode VM 実行プログラムを起動します。
- `-Djeode` オプションは、アプリケーションからの出力に使用されるテキスト・コンソールの表示を制御します。

- `-Djul.library.dir` オプションは、Ultra Light ネイティブ・インタフェースのランタイム・ライブラリ (*jul9.dll*) の位置を VM に示します。
- `-cp` オプションは、VM のクラスパスを提供します。アプリケーションと Ultra Light ランタイム・ライブラリのロケーションを示します。
- 最後の引数はクラスで、この例では `Customer` です。

次の手順では、ファイルをデバイスにコピーします。Ultra Light ランタイム・ファイルとアプリケーション固有のファイルの両方をコピーしてください。

❖ Ultra Light ランタイムを Windows CE デバイスに配備するには、次の手順に従います。

1 Windows CE デバイスの Windows エクスプローラを起動します。

- デバイスとデスクトップ・コンピュータが接続されていることを確認します。
- [ActiveSync] ウィンドウで [エクスプローラ] をクリックします。[エクスプローラ] ウィンドウが開きます。

2 Ultra Light ランタイムとアプリケーションを保存するディレクトリを作成します。

- [エクスプローラ] ウィンドウで [マイ ポケット PC] をクリックするとルート・ディレクトリにアクセスします。
- *UltraLite* という名前のディレクトリを作成します。
- UltraLite ディレクトリを開き、*lib* と *tutorial* という名前のサブディレクトリを作成します。ディレクトリ・パス *¥UltraLite¥lib* は Ultra Light ランタイム・ファイルのロケーションで、パス *¥UltraLite¥tutorial* はアプリケーションのロケーションです。これらのディレクトリは、前述のショートカット・ファイルのオプションと一致しています。

3　Ultra Light ランタイム・ファイルを Windows CE デバイスにコピーします。

- 別の [エクスプローラ] ウィンドウを起動し、デスクトップ・マシン上の SQL Anywhere インストール・ディレクトリに移動します。
- デスクトップからデバイスに次のファイルをドラッグします。

SQL Anywhere ディレクトリからの相対パスで示したデスクトップ・ロケーション	Windows CE ロケーション
UltraLite¥NativeUltraLiteForJava¥jul9.jar	¥UltraLite¥lib¥jul9.jar
UltraLite¥NativeUltraLiteForJava¥ce¥arm¥jul9.dll	¥UltraLite¥lib¥jul9.dll

または、複数のプロセスによる Ultra Light データベースへの同時アクセスを可能にする実行プログラムである Ultra Light エンジンを配備することもできます。Ultra Light エンジンを使用する場合、コンポーネントの適切なバージョン (*julclient9.dll*) を配備し、Runtime Type プロパティを Database Manager に指定します。

4　アプリケーション・ファイルを Windows CE デバイスにコピーします。

- [エクスプローラ] ウィンドウで、チュートリアルのディレクトリに移動します。
- デスクトップからデバイスに次のファイルをドラッグします。

デスクトップ・ロケーション	Windows CE ロケーション
Customer.class	¥UltraLite¥tutorial
tutcustomer.usm	¥UltraLite¥tutorial

このチュートリアルでは、データベース・ファイルを Windows CE デバイスにコピーしないでください。代わりに、スキーマ・ファイルをデバイスに配備します。これによって、このスキーマ・ファイルに基づいてデータベースが作成されます。

5 CrEme VM を実行している場合、ActiveSync を使用するには、*¥Windows¥CrEme¥lib¥crmex.jar* をクラスパスに追加する必要があります。

6 ショートカット・ファイルを Windows CE デバイスにコピーします。

- デスクトップからデバイスに次のファイルをドラッグします。

デスクトップ・ロケーション	Windows CE ロケーション
tutorial.lnk	¥Windows¥ スタート メニュー

次の手順では、アプリケーションを実行します。

❖ アプリケーションを実行するには、次の手順に従います。

1 Windows CE デバイスで、[スタート] ー [Tutorial] を選択します。

このショートカットは、デバイスにコピーした *tutorial.lnk* ファイルです。

- Jeode VM がロードされ、コンソールが表示されます。
- 次のメッセージがコンソールに出力されます。

```
Connected to a new database.
 Two rows inserted.
 id= 1, name= Gene Poole
 id= 2, name= Penny Stamp
 Application finished. Please close console
```

- コンソールを閉じます。

2　Windows CE デバイスで、[スタート] ー [Tutorial] をもう一度選択します。

今回は、次の 2 つのメッセージが最初に表示されます。

```
Connected to an existing database.
 The table has 2 rows.
```

3　コンソールを閉じます。

レッスン 5：アプリケーションへの同期の追加

次の手順では、アプリケーションに同期コードを追加し、Mobile Link 同期サーバを起動し、アプリケーションを実行して同期します。

注意
このレッスンでは、SQL Anywhere Studio の一部である Mobile Link 同期を使用します。このレッスンを実行するには、SQL Anywhere Studio をコンピュータにインストールしておきます。

同期は、ASA 9.0 Sample データベースを使用して実行されます。ASA 9.0 サンプル・データベースの Customer テーブルのカラムは、作成した Ultra Light データベースの customer テーブルのカラムに対応します。同期中、このテーブルのデータは Ultra Light アプリケーションにダウンロードされます。

このレッスンは、Mobile Link 同期についてある程度の知識を持っていることを前提としています。

❖ アプリケーションに同期を追加するには、次の手順に従います。

1 *Customer.java* ファイルに以下のメソッドを追加します。

このコードでは、次のタスクを実行します。

- 同期ストリームを TCP/IP に設定します。同期は、HTTP、ActiveSync、または HTTPS を使用しても実行できます。HTTPS 同期の場合は、別途ライセンスが必要な SQL Anywhere Studio セキュリティ・オプションが必要です。

  詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』> 「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.StreamType**」と「**ianywhere.native_ultralite.Connection**」を参照してください。

Connection オブジェクトの SyncParms フィールドは、SyncParms オブジェクトにアクセスするのに便利です。詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』> 「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.SyncParms**」を参照してください。

- Mobile Link 同期は、スクリプトによって Mobile Link 同期サーバで制御されます。スクリプト・バージョンは、使用するスクリプト・セットを識別します。setSendColumnNames メソッドと Mobile Link 同期サーバのオプションによって、スクリプトが自動的に生成されます。

  詳細については、『Mobile Link 管理ガイド』> 「スクリプトの自動生成」を参照してください。

- Mobile Link ユーザ名を設定します。この値は、Mobile Link 同期サーバでの認証に使用されますが、Ultra Light データベースのユーザ ID とは異なります。ただし、一部のアプリケーションでは、同じ値を設定する場合もあります。

- 同期をデータのダウンロード専用に設定します。デフォルトでは、Mobile Link 同期は双方向です。ここでは、テーブルのローが Sample データベースにアップロードされないように、ダウンロード専用同期を使用します。

```
private void sync() throws SQLException {
   conn.syncParms.setStream( StreamType.TCPIP );
   conn.syncParms.setVersion( "ul_default" );
   conn.syncParms.setUserName( "sample" );
   conn.syncParms.setSendColumnNames( true );
   conn.syncParms.setDownloadOnly( true );
   conn.synchronize();
 }
```

2　sync メソッドを呼び出します。

次の行を、main() メソッドの insert メソッドの呼び出しの直後、select メソッドの呼び出しの前に追加します。

```
cust.sync();
```

3 アプリケーションをコンパイルします。詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「レッスン 1：データベースへの接続」を参照してください。アプリケーションを実行しないでください。

❖ データを同期するには、次の手順に従います。

1 Mobile Link 同期サーバを起動します。

コマンド・プロンプトでコマンド・ラインを使用して Mobile Link を実行します。

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -v+ -zu+ -za
```

ASA 9.0 Sample データベースの Customer テーブルは、作成した Ultra Light データベースとカラムが一致しています。Ultra Light アプリケーションは、ASA 9.0 Sample データベースと同期できます。

`-zu+` と `-za` コマンド・ライン・オプションによって、ユーザの追加と同期スクリプトの生成が自動的に行われます。

これらのオプションの詳細については、『Mobile Link 管理ガイド』>「Mobile Link 同期サーバのオプション」を参照してください。

2 アプリケーションを実行します。詳細については、「レッスン 1：データベースへの接続」9 ページ を参照してください。

Mobile Link 同期サーバのウィンドウでは、同期の進行状況を示すステータス・メッセージが表示されます。最後のメッセージ「同期が完了しました。」が表示されます。

まとめ

学習の成果

このチュートリアルでは、以下の作業を行いました。

- データベース・スキーマの作成
- Native Ultra Light for Java アプリケーションの作成
- Ultra Light リモート・データベースと Adaptive Server Anywhere 統合データベースの同期
- Native Ultra Light for Java アプリケーションの開発プロセスに関する知識の取得

サンプル

- その他のコード・サンプルについては、*Samples¥NativeUltraLiteForJava¥Simple¥Simple.java* を参照してください。

第 3 章

チュートリアル：CustDB サンプル・アプリケーション

この章の内容

この章はチュートリアル形式で、Native Ultra Light for Java を使用して複数テーブルのアプリケーションを構築および配備するプロセスを解説します。

また、このチュートリアルでは、Ultra Light リモート・データベースと統合データベース間の同期についても説明します。このチュートリアルの同期に関する操作では、SQL Anywhere Studio が必要です。

概要

このチュートリアルでは、CustDB サンプル・アプリケーションをコンパイル、実行、配備するプロセスを解説します。このチュートリアルでは、Native Ultra Light for Java を使用して、いくつかの一般的なタスクを実装する方法について説明します。

CustDB のソース・コードは、SQL Anywhere インストール環境の *Samples¥NativeUltraLiteForJava¥CustDB* サブディレクトリにあります。

所要時間

このチュートリアルは、コードをコピーして貼り付ける場合、約 30 分で終了します。自分でコードを入力する場合、これより長い時間が必要です。

能力と経験

このチュートリアルは、次のことを前提にしています。

- Jeode VM を実行している Windows CE デバイスにアクセスできる。
- Java Development Kit がコンピュータにインストールされている。配備に使用する Jeode VM との互換性のため、JDK 1.1.8 または Personal Java 1.2 の使用をおすすめします。
- Ultra Light スキーマ・ペインタを使用して、Ultra Light スキーマを作成する方法を知っている。

  詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「レッスン 1：Ultra Light データベース・スキーマの作成」を参照してください。

注意
このチュートリアルの同期に関する操作では、SQL Anywhere Studio が必要です。

目的

このチュートリアルの目的は、Native Ultra Light for Java アプリケーションの開発について知識と経験を得ることです。

レッスン 1：CustDB アプリケーションの構築

次の手順では、インストール環境の *Samples¥NativeUltraLite-ForJava¥CustDB* サブディレクトリにある *build.bat* スクリプトを使用して、Java ソース・ファイルから CustDB サンプルを構築します。

clean.bat スクリプトも、構築結果を削除するために提供されています。

❖ サンプルを構築するには、次の手順に従います。

1. コマンド・プロンプトを開き、SQL Anywhere インストール環境の *Samples¥NativeUltraLiteForJava¥CustDB* サブディレクトリに移動します。

   このコマンド・プロンプトは、チュートリアル全体を通じて開いたままにしておきます。

2. 必要な環境変数を設定します。

   環境変数 ASANY9 と JAVA_HOME が正しく定義されていることを確認します。

   - ASANY9 は、セットアップ・プログラムによって SQL Anywhere インストール環境のロケーションに設定されます。
   - JAVA_HOME 環境変数を、コンピュータ上の JDK のロケーションに設定します。配備に使用する Jeode VM との互換性のため、JDK 1.1.8 または Personal Java 1.2 の使用をおすすめします。

     たとえば、バージョン 1.1.8 の JDK が *c:¥jdk1.1.8* にある場合は、次のコマンドを実行します。

     ```
     set JAVA_HOME=c:¥jdk1.1.8
     ```

3. アプリケーションを構築します。

   コマンド・プロンプトで次のコマンドを実行します。

```
build
```

build バッチ・ファイルは、次の操作を実行します。

- CustDB サンプル・コードをコンパイルします。

  コンパイル済みクラスは、*builddir* サブディレクトリに格納されます。

- コンパイル済みクラスを JAR ファイルに入れます。

  JAR ファイルは、*CustDB* ディレクトリに保管されます。

- *ulinit* コマンド・ライン・ユーティリティを使用してデータベース・スキーマ・ファイルを作成します。

  このスキーマ・ファイルは、Adaptive Server Anywhere Ultra Light 9.0 Sample データベースから作成されます。

  ulinit ツールの詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「ulinit ユーティリティ」を参照してください。

サンプル・コード

次の表は、サンプル・コード・ファイルとその簡単な説明を示します。

ファイル	**説明**
Application.java	メイン・ユーザ・インタフェース・フレーム、イベント処理、ActiveSync のサポート。
CustDB.java	データベースへのメイン・インタフェース。このファイルには、ほとんどのデータベース処理があります。
DialogDelOrder.java	削除の確認ダイアログ。
DialogNewOrder.java	データベースからリスト・ボックスへのデータの移植を示す新規注文入力フォーム。
Dialogs.java	ダイアログ共通の基本クラス。

ファイル	説明
DialogSyncHost.java	同期ホスト名の入力を要求する。
DialogUserID.java	従業員 ID の入力を要求する。
GetOrder.java	注文データを表すクラス。キー・ジョインの手順を示します。
GetOrderData.java	注文 ID の最小値と最大値を計算する。SELECT max(order_id), min(order_id) FROM ULOrder に相当します。

レッスン 2：CustDB アプリケーションの実行

次の手順は、Windows で CustDB サンプルを実行する方法を示します。Windows CE デバイスへの配備については、「レッスン 3：Windows CE デバイスへの CustDB の配備」36 ページを参照してください。このレッスンでは Mobile Link 同期を使用するため、SQL Anywhere Studio が必要です。

❖ Windows でサンプルを実行するには、次の手順に従います。

1 前のレッスンと同じコマンド・プロンプトから、`win32¥run.bat` と入力します。

このコマンドは、次の操作を実行します。

- Mobile Link 同期サーバを起動します。

  このサーバは、CustDB アプリケーションの統合データベースとして使用されている Ultra Light 9.0 Sample データベースに接続します。

- CustDB サンプルを起動します。

  初めてサンプルを実行するときは、Ultra Light データベース (*.udb* ファイル) なしでアプリケーションが起動されるため、このファイルが Ultra Light スキーマから作成されます。

- ログオン・ウィンドウを表示します。

  このウィンドウは画面の中央に表示されますが、他のウィンドウの背後に表示される場合もあります。ログオン・ウィンドウを表示するには、他のウィンドウを移動しなければならない場合もあります。

2 アプリケーションにログオンします。

[OK] をクリックして、Mobile Link ユーザ名 **50** のユーザとしてログオンします。アプリケーションが同期され、同期の進行状況情報が表示されます。少し間をおいてから、ウィンドウに 1 つの注文が表示されます。

3　アプリケーションのさまざまな機能を試します。

データベース内のローからローへの移動、注文の承認と拒否、同期ができます。アプリケーションを終了すると、バッチ・ファイルが Mobile Link 同期サーバを停止します。

レッスン 3：Windows CE デバイスへの CustDB の配備

次の手順は、アプリケーションを Windows CE デバイスに配備します。配備には、Jeode Runtime (Java VM) が必要です。

❖ Windows CE デバイスにアプリケーションを配備するには、次の手順に従います。

1 Windows CE デバイスの Windows エクスプローラを起動します。

- デバイスとデスクトップ・コンピュータが接続されていることを確認します。
- [ActiveSync] ウィンドウで [エクスプローラ] をクリックします。

  [エクスプローラ] ウィンドウが開きます。

2 Ultra Light ランタイムとアプリケーションを保存するディレクトリを作成します。

前のチュートリアルを完了した場合は、これらのディレクトリのいくつかがすでに存在していることがあります。

- [エクスプローラ] ウィンドウで、[My Pocket PC] をクリックします。これはルート・ディレクトリであり、パスは ¥ です。
- *UltraLite* という名前のディレクトリを作成します。
- UltraLite ディレクトリを開き、*lib* と *CustDB* という名前のサブディレクトリを作成します。

  ¥UltraLite¥lib は Ultra Light ランタイム・ファイルのロケーションで、*¥UltraLite¥CustDB* はアプリケーションのロケーションです。これらのディレクトリは、*ce* サブディレクトリにあるショートカット・ファイルのオプションと一致しています。

3　Ultra Light ランタイム・ファイルを Windows CE デバイスにコピーします。

前のチュートリアルを完了した場合は、この操作がすでに実行されていることがあります。手順を繰り返す必要はありません。

- 別の [エクスプローラ] ウィンドウを起動し、デスクトップ・マシン上の SQL Anywhere インストール・ディレクトリに移動します。
- デスクトップからデバイスに次のファイルをドラッグします。

SQL Anywhere ディレクトリからの相対パスで示したデスクトップ・ロケーション	Windows CE ロケーション
UltraLite¥NativeUltraLiteForJava¥jul9.jar	*¥UltraLite¥lib*
UltraLite¥NativeUltraLiteForJava¥ce¥arm¥jul9.dll	*¥UltraLite¥lib*

4　アプリケーション・ファイルを Windows CE デバイスにコピーします。

- [エクスプローラ] ウィンドウで、*Samples¥NativeUltraLiteForJava¥CustDB* ディレクトリに移動します。
- デスクトップからデバイスに次のファイルをドラッグします。

デスクトップ・ロケーション	Windows CE ロケーション
custdb.jar	*¥UltraLite¥CustDB*
ul_custapi.usm	*¥UltraLite¥CustDB*

このチュートリアルでは、データベース・ファイルを Windows CE デバイスにコピーしないでください。代わりに、スキーマ・ファイルをデバイスに配備します。これによって、データベースが作成されます。

5 ショートカット・ファイルを Windows CE デバイスにコピーします。

- デスクトップからデバイスに次のファイルをドラッグします。

SQL Anywhere ディレクトリからの相対パスで示したデスクトップ・ロケーション	Windows CE ロケーション
ce¥CustDB.lnk	*¥Windows¥Start Menu*

6 ActiveSync プロバイダをインストールします。

ActiveSync プロバイダは同期を行うために必要です。

手順については、『Mobile Link クライアント』> 「ActiveSync プロバイダ・インストール・ユーティリティ」を参照してください。

7 ActiveSync で使用するアプリケーションを登録します。

CustDB を登録するときは、次のプロパティを使用します。

プロパティ	値
名前	JULCustDB
クラス名	JULCustDB
ファイル・ロケーション	*¥Windows¥evm.exe*
引数	-Djeode.evm.console.local.keep=TRUE -Djul.library.dir=¥UltraLite¥lib -cp ¥UltraLite¥CustDB¥custdb.jar;¥UltraLite¥lib¥jul9.jar custdb.Application ACTIVE_SYNC_LAUNCH

バッチ・ファイル *Samples¥NativeUltraLiteFor-Java¥CustDB¥ce¥as_register.bat* を実行して、この処理を実行します。

SQL Anywhere Studio ユーザ
ActiveSync 同期の設定手順については、『Mobile Link クライアント』> 「ActiveSync プロバイダ・インストール・ユーティリティ」を参照してください。

❖ アプリケーションを実行するには、次の手順に従います。

1. Mobile Link 同期サーバを起動します。

   SQL Anywhere インストール環境の *Samples¥NativeUltraLiteForJava¥CustDB¥ce¥* サブディレクトリにある *startdb.bat* バッチ・ファイルを実行します。

2. CE デバイスの [スタート] メニューから *CustDB* アプリケーションを実行します。

まとめ

このチュートリアルでは、以下の作業を行いました。

- デスクトップ・コンピュータで CustDB サンプルを構築し、実行しました。
- Windows CE デバイスに Native Ultra Light for Java アプリケーションを配備しました。

第 4 章

Native Ultra Light for Java 開発の概要

この章の内容

この章では、Native Ultra Light for Java を使用してアプリケーションを開発する方法について説明します。

チュートリアルについては、『Native Ultra Light for Java ユーザーズ・ガイド』>「チュートリアル：Native Ultra Light for Java アプリケーションの構築」または『Native Ultra Light for Java ユーザーズ・ガイド』>「チュートリアル：CustDB サンプル・アプリケーション」を参照してください。

Ultra Light データベース・スキーマ

データベース・スキーマは、データベースに関する記述です。データベース・スキーマは、データベース内のテーブル、インデックス、キー、パブリケーションの集合であり、それらの間のすべての関係を示します。

Ultra Light データベースのスキーマを直接変更しないでください。代わりに、アプリケーションに組み込まれた Ultra Light 機能を使用して、スキーマ・ファイル (*.usm*) を作成し、作成したファイルからデータベース・スキーマをアップグレードします。

スキーマ・ファイルは、データベースを最初に作成するときにも使用され、データベースの構造を指定します。

Ultra Light データベース・スキーマ・ファイルの作成

Ultra Light スキーマ・ペインタまたは *ulinit* ユーティリティを使用して、Ultra Light スキーマ・ファイルを作成できます。

- **Ultra Light スキーマ・ペインタ**　Ultra Light スキーマ・ペインタは、Ultra Light スキーマ・ファイルの作成と編集に使用するグラフィカル・ユーティリティです。

  スキーマ・ペインタを起動するには、[スタート] － [プログラム] － [SQL Anywhere 9] － [Ultra Light] － [Ultra Light Schema Painter] を選択するか、Windows エクスプローラでスキーマ・ファイル (*.usm*) をダブルクリックします。

  Ultra Light スキーマ・ペインタの使用方法の詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「レッスン 1：Ultra Light データベース・スキーマの作成」を参照してください。

- **ulinit ユーティリティ**　Adaptive Server Anywhere データベース管理システムがある場合は、*ulinit* コマンド・ライン・ユーティリティを使用して Ultra Light スキーマ・ファイルを生成できます。

ulinit ユーティリティの使用の詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「ulinit ユーティリティ」を参照してください。

データベース・スキーマのアップグレード

既存のデータベース構造を修正するには、DatabaseSchema.applyFile メソッドを使用します。ほとんどの場合、データ損失は起こりません。ただし、カラムを削除したり、カラムのデータ型が互換性のない型に変更された場合などは、データ損失が起こる場合があります。

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』> 「Native Ultra Light for Java API リファレンス」の「DatabaseSchema.ApplyFile」を参照してください。

例

次のコードは、ApplyFile メソッドを使用して新しいスキーマ・ファイルを適用します。ここでは、すでにデータベースに接続していることを前提としています (「データベースへの接続」44 ページを参照)。

```
DatabaseSchema schema = conn.schema;
 SchemaParms parms = new SchemaParms();
 parms.schemaOnDesktop = "c:¥tutorial¥tutcustomer.usm";
 try {
   schema.applyFile(parms );
 } catch {
   // handle any errors
 }
```

データベースへの接続

Ultra Light アプリケーションをデータベースに接続しないと、データベースのデータを操作できません。この項では、Ultra Light データベースに接続する方法について説明します。

Connection オブジェクトの使用

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

- **コミット動作** デフォルトでは、Native Ultra Light for Java アプリケーションは AutoCommit モードに設定されています。insert 文、update 文、delete 文はすべて、すぐにデータベースにコミットされます。Connection.AutoCommit を false に設定して、アプリケーションにトランザクションを構築することもできます。

  詳細については、「Ultra Light でのトランザクション処理」63 ページを参照してください。

- **ユーザ認証** Grant と Revoke の接続許可メソッドを使用すると、アプリケーションのユーザ ID とパスワードをデフォルト値の DBA と SQL から別の値に変更できます。各アプリケーションは、最大 4 つのユーザ ID を保持できます。

  詳細については、「ユーザ認証」67 ページを参照してください。

- **同期** Connection オブジェクトから、同期を管理するオブジェクトのセットにアクセスできます。

  詳細については、「Ultra Light アプリケーションの同期」68 ページを参照してください。

- **テーブル** Ultra Light テーブルには、Connection オブジェクトのメソッドを使用してアクセスします。

  詳細については、「テーブル API によるデータ・アクセスと操作」55 ページを参照してください。

- **準備文** 動的 SQL 文の実行を処理し、結果セットをナビゲーションするために、オブジェクトのセットが提供されています。

詳細については、「動的 SQL を使用したデータへのアクセスと操作」 49 ページを参照してください。

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.Connection**」を参照してください。

マルチスレッド・アプリケーション

Connection オブジェクトと、このオブジェクトから作成されるすべてのオブジェクトは、それぞれ単一のスレッドで使用してください。アプリケーションが Ultra Light データベースにアクセスするのに複数のスレッドを必要とする場合は、スレッドごとに個別の接続が必要です。

❖ Ultra Light データベースに接続するには、次の手順に従います。

1　DatabaseManager オブジェクトを作成および初期化します。

DatabaseManager オブジェクトは、オブジェクト階層のルートにあります。DatabaseManager オブジェクトは、1 つのアプリケーションに 1 つだけ作成します。多くの場合、DatabaseManager オブジェクトは、アプリケーションに対してグローバルに宣言するのが最も効果的です。

```
DatabaseManager dbMgr = new DatabaseManager();
```

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.DatabaseManager**」を参照してください。

2　Connection オブジェクトを宣言します。

ほとんどのアプリケーションは、Ultra Light データベースへの単一接続を使用し、接続を開いたままにしておきます。複数の接続が必要になるのは、マルチスレッド・データ・アクセスの場合だけです。そのため、Connection オブジェクトは、アプリケーションに対してグローバルに宣言するのが最も効果的です。

```
static Connection conn;
```

3 既存のデータベースへの接続を開きます。指定されたデータベース・ファイルが存在しない場合は、新しいデータベースを作成します。

ほとんどの Ultra Light アプリケーションは、データベース・ファイルではなくスキーマ・ファイルを配備し、最初の接続時に Ultra Light にデータベース・ファイルを作成させます。次のコードは、既存のデータベースへの接続を試みます。データベース・ファイルが存在しない場合は、アプリケーションがデータベース・ファイルを作成します。

```
ConnectionParms parms = new ConnectionParms();
 // specify the location of the database file
 parms.databaseOnDesktop = "mydata.udb";
 try {
   conn = dbMgr.openConnection( parms );
 } catch( SQLException econn ) {
   if( econn.getErrorCode() ==
     SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND ){
     CreateParms parms = new CreateParms();
     parms.databaseOnDesktop = "mydata.udb";
     parms.schema.schemaOnDesktop = "mydata.usm";
     try {
       conn = dbMgr.createDatabase( parms );
   }
 }
```

例 次のコードは、Ultra Light データベース *mydata.udb* への接続を開きます。データベースが存在しない場合は、*mydata.usm* という名前の Ultra Light スキーマ・ファイルを使用して作成されます。

詳細については、*Samples¥NativeUltraLiteForJava¥Simple¥Simple.java* のコードを参照してください。

```
CreateParms parms = new CreateParms();
 parms.databaseOnDesktop = "mydata.udb";
 parms.schema.schemaOnDesktop = "mydata.usm";
 try {
   conn = dbMgr.openConnection( parms );
   System.out.println(
     "Connected to an existing database." );
 }
 catch( SQLException econn ) {
   if( econn.getErrorCode() ==
       SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND ){
```

```
            conn = dbMgr.createDatabase( parms );
            System.out.println(
            "Connected to a new database." );
          } else {
            throw econn;
          }
        }
```

暗号化と難読化

Native Ultra Light for Java を使用して Ultra Light データベースを暗号化または難読化できます。

暗号化

暗号化を使用してデータベースを作成するには、**ianywhere.native_ultralite.ConnectionParms.EncryptionKey** プロパティを使用して暗号化キーを指定し、次に **createDatabase** メソッドを呼び出します。**createDatabase** メソッドを呼び出すと、データベースが作成され、指定されたキーを使用して暗号化されます。

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「EncryptionKey 接続パラメータ」を参照してください。

暗号化キーを変更するには、**ianywhere.native_ultralite.Connection.changeEncryptionKey** メソッドを使用して新しい暗号化キーを指定します。

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』> 「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.Connection**」を参照してください。

データベースが暗号化された後は、データベースへの接続で正しい暗号化キーを指定する必要があります。そうしないと、接続は失敗します。

難読化

データベースを難読化するには、作成パラメータとして obfuscate=1 を指定します。

データベースの暗号化の詳細については、『Ultra Light データベース・ユーザーズ・ガイド』> 「Ultra Light データベースの暗号化」を参照してください。

動的 SQL を使用したデータへのアクセスと操作

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

テーブル API の使用方法については、「テーブル API によるデータ・アクセスと操作」55 ページを参照してください。

この項では、動的 SQL を使用して次の操作を行う方法について説明します。

- ローの挿入、削除、更新
- 結果セットのローの取得
- 結果セットのローのスクロール

この項では、SQL 言語そのものについては説明しません。動的 SQL 機能の詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「動的 SQL」を参照してください。

SQL 操作に必要な一連の操作の概要については、『Ultra Light データベース・ユーザーズ・ガイド』>「動的 SQL の使用」を参照してください。

データ操作 : INSERT、UPDATE、DELETE

Ultra Light では、SQL データ操作言語の操作を実行できます。この操作は、PreparedStatement.executeStatement メソッドを使用して実行されます。

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.PreparedStatement**」を参照してください。

準備文で使用するパラメータ

SQL 文のパラメータのプレースホルダは、? 文字を使用して指定します。INSERT、UPDATE、DELETE で、準備文での並び順に従ってそれぞれの ? が参照されます。たとえば、最初の ? は 1、2 番目の ? は 2 のようになります。

❖ ローを挿入するには、次の手順に従います。

1 PreparedStatement を宣言します。

```
PreparedStatement prepStmt;
```

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

```
prepStmt = conn.prepareStatement(
    "INSERT INTO MyTable(MyColumn) values (?)");
```

3 文の入力パラメータ値を割り当てます。

次のコードは、文字列パラメータを示します。

```
String newValue;
 // assign value
 prepStmt.setStringParameter(1, newValue);
```

4 文を実行します。

戻り値は、文に影響されたローの数を示します。

```
long rowsInserted = prepStmt.executeStatement();
```

5 AutoCommit をオフにしている場合は、手動で変更内容をコミットします。

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

❖ ローを更新するには、次の手順に従います。

1 PreparedStatement を宣言します。

```
PreparedStatement prepStmt;
```

2　文を PreparedStatement オブジェクトに割り当てます。

```
prepStmt = conn.prepareStatement(
    "UPDATE MyTable SET MyColumn1 = ? WHERE
MyColumn2 = ?");
```

3　文の入力パラメータ値を割り当てます。

```
String newValue;
 String oldValue;
 // assign values
 prepStmt.setStringParameter( 1, newValue );
 prepStmt.setStringParameter( 2, oldValue );
```

4　文を実行します。

```
long rowsUpdated = prepStmt.executeStatement();
```

5　AutoCommit をオフにしている場合は、手動で変更内容をコミットします。

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

❖ ローを削除するには、次の手順に従います。

1　PreparedStatement を宣言します。

```
PreparedStatement prepStmt;
```

2　文を PreparedStatement オブジェクトに割り当てます。

```
prepStmt = conn.prepareStatement(
    "DELETE FROM MyTable WHERE MyColumn = ?");
```

3　文の入力パラメータ値を割り当てます。

```
String deleteValue;
 prepStmt.setStringParameter(1, deleteValue);
```

4　文を実行します。

```
long rowsDeleted = prepStmt.executeStatement();
```

5　AutoCommit をオフにしている場合は、手動で変更内容をコミットします。

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

データの取得 : SELECT

SELECT 文を使用すると、データベースから情報を取り出すことができます。この項では、SELECT 文の実行方法と返される結果セットの処理方法について説明します。

❖ SELECT 文を実行するには、次の手順に従います。

1 クエリを保持する PreparedStatement オブジェクトを宣言します。

```
PreparedStatement prepStmt;
```

2 このオブジェクトに文を割り当てます。

```
prepStmt = conn.prepareStatement(
 "SELECT MyColumn FROM MyTable");
```

3 文を実行します。

以下のコードでは、SELECT のクエリ結果に文字列が含まれています。これはコマンド・プロンプトへ出力されます。

```
ResultSet customerNames = prepStmt.executeQuery();
 customerNames.moveBeforeFirst();
 while( customerNames.moveNext() ) {
    for ( int i = 1;
          i <=
customerNames.schema.getColumnCount();
          i++ ) {
       System.out.print(
         customerNames.getString( i ) + " "
         );

   }
   System.out.println();
 }
```

動的 SQL 結果セットのナビゲーション

ResultSet オブジェクトに関連したメソッドを使用して、結果セット内をナビゲートすることができます。

結果セット・オブジェクトは、結果セットをナビゲーションする次のメソッドを提供します。

- **moveAfterLast()** 最後のローの後に移動します。
- **moveBeforeFirst()** 最初のローの前に移動します。
- **moveFirst()** 最初のローに移動します。
- **moveLast()** 最後のローに移動します。
- **moveNext()** 次のローに移動します。
- **movePrevious()** 前のローに移動します。
- **moveRelative(offset)** 現在のローを基準にして、オフセットが指定した数だけローを移動します。オフセット値を正の値で指定すると、現在の結果セットのカーソル位置から前方に移動します。負の値で指定すると後方に移動します。オフセット値が 0 の場合、カーソルは移動しませんが、ロー・バッファが再配置されます。

結果セット・スキーマの説明

ResultSet.schema プロパティを使用して、カラム名、カラムの総数、カラム・スケール、カラム・サイズ、カラム SQL 型など、結果セットに関する情報を取得できます。

例

次のサンプル・コードは、ResultSet.schema プロパティを使用して、スキーマ情報をコンソール・ウィンドウに表示する方法を示しています。

```
for ( int i = 1;
        i <= MyResultSet.schema.getColumnCount();
        i++ ) {
    System.out.println(
        MyResultSet.schema.getColumnName(i) + " " +
        MyResultSet.schema.getColumnSQLType(i)
    );
 }
```

テーブル API によるデータ・アクセスと操作

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

動的 SQL については、「動的 SQL を使用したデータへのアクセスと操作」49 ページを参照してください。

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

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

テーブルのローのナビゲーション

Native Ultra Light for Java は、幅広いナビゲーション作業を行うため、テーブルをナビゲーションする方法を多数提供します。

テーブル・オブジェクトは、テーブルをナビゲーションする次のメソッドを提供します。

- **moveAfterLast()** 最後のローの後に移動します。
- **moveBeforeFirst()** 最初のローの前に移動します。
- **moveFirst()** 最初のローに移動します。
- **moveLast()** 最後のローに移動します。
- **moveNext()** 次のローに移動します。
- **movePrevious()** 前のローに移動します。

- **moveRelative(offset)** 現在のローを基準にして、オフセットが指定した数だけローを移動します。オフセット値を正の値で指定すると、現在のテーブルのカーソル位置から前方に移動します。負の値で指定すると後方に移動します。オフセット値が 0 の場合、カーソルは移動しませんが、ロー・バッファが再配置されます。

例 次のサンプル・コードは、MyTable テーブルを開き、各ローの MyColumn カラムの値を表示します。

```
Table t = conn.getTable( "MyTable" );
 short colID = t.schema.getColumnID( "MyColumn" );
 t.open();
 t.moveBeforeFirst();
 while ( t.moveNext() ){
    System.out.println( t.getString( colID ) );
 }
```

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

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

```
Table t = conn.getTable( "MyTable" );
 t.open( "ix_col" );
 t.moveFirst();
```

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.Table**」と「**ianywhere.native_ultralite.TableSchema**」を参照してください。

Ultra Light のモードの使用

Ultra Light モードは、バッファ内の値の使用目的を指定します。Ultra Light には、デフォルト・モードに加えて、次の 4 つの操作モードがあります。

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

現在のローの値へのアクセス

Table オブジェクトは、次のいずれかの位置に常に置かれています。

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

Table オブジェクトがローの上に置かれている場合は、そのデータ型に適したメソッド・セットを使用して、各カラムの値を取得したり、変更したりできます。

カラム値の取得

Table オブジェクトは、カラム値を取得するメソッド・セットを提供します。これらのメソッドは、カラム ID を引数として取ります。

例

次のコードは、lname カラムの値を取得します。このカラムの値は文字列です。

```
short lname = t.schema.getColumnID( "lname" );
 String lastname = t.getString( lname );
```

次のコードは、cust_id カラムの値を取得します。このカラムの値は整数です。

```
short cust_id = t.schema.getColumnID( "cust_id" );
 int id = t.getInt( cust_id );
```

カラム値の変更

値を取り出すメソッド以外に、値を設定するメソッドもあります。値を設定するメソッドは、カラム ID と値を引数として取ります。

例

たとえば、次のコードは、lname カラムの値を Kaminski に設定します。

```
t.setString( lname, "Kaminski" );
```

これらのプロパティへの値の割り当てによって、データベース内のデータの値が変更されることはありません。現在の位置がテーブルの最初のローの前でも、テーブルの最後のローの後ろであっても、プロパティに値を割り当てることができます。ただし、これらの位置に現在のローが置かれている場合に、たとえば、変数へのプロパティの割り当てなどによって、データにアクセスしようとするとエラーになります。

```
// This code is incorrect
 tCustomer.moveBeforeFirst();
 id = tCustomer.getInt( cust_id );
```

値のキャスト

選択するメソッドは、割り当てるデータ型に一致させてください。データ型に互換性がある場合は、Ultra Light が自動的にデータベースのデータ型をキャストするため、getString メソッドを使用して整数値を文字列変数にフェッチしたりできます。

find と lookup を使用したローの検索

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

注意

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

- **find メソッド**　Table オブジェクトを開いたときに指定したソート順に基づいて、指定された検索値と正確に一致する最初のローに移動します。検索値が見つからない場合は、最初のローの前、または最後のローの後ろに位置設定されます。
- **lookup メソッド**　Table オブジェクトを開いたときに指定したソート順に基づいて、指定された検索値と一致するか、それより大きい値の最初のローに移動します。

❖ ローを検索するには、次の手順に従います。

1　検索モードまたはルックアップ・モードを開始します。

モードを開始するには、テーブル・オブジェクトのメソッドを呼び出します。たとえば、次のコードは検索モードを開始します。

```
t.findBegin();
```

2　検索値を設定します。

検索値は、現在のローの値を設定することで設定します。これらの値の設定は、データベースではなく、現在のローを保持しているバッファにのみ影響します。たとえば、次のコードは、バッファの値を Kaminski に設定します。

```
short lname = t.schema.getColumnID( "lname" );
 t.setString( lname, "Kaminski" );
```

3　ローを検索します。

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

マルチカラム・インデックスの場合、最初のカラムの値が常に使用され、ほかのカラムは省略できます。

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

4　ローの次のインスタンスを検索します。

適切なメソッドを使用して検索を実行します。検索操作では、findNext() で、インデックス内の次のインスタンスを検索します。ルックアップ操作では、moveNext() で次のインスタンスを検索します。

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.Table**」を参照してください。

ローの更新

次の手順では、ローを更新します。

❖ ローを更新するには、次の手順に従います。

1. 更新するローに移動します。

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

2. 更新モードを開始します。

   たとえば、次の指示は、t 上で更新モードを開始します。

   ```
   t.beginUpdate();
   ```

3. 更新するローの新しい値を設定します。たとえば、次の指示は、バッファ内の id カラムを 3 に設定します。

   ```
   t.setInt( id , 3);
   ```

4. Update を実行します。

   ```
   t.update();
   ```

更新操作が終了すると、更新したローが現在のローになります。Table オブジェクトを開いたときに指定したインデックスのカラム値を変更した場合は、現在のローの定義が解除されます。

デフォルトでは、Native Ultra Light for Java は AutoCommit モードで動作するため、更新は永続的な記憶領域のローに即時適用されます。AutoCommit モードを無効にした場合は、コミット操作を実行するまで、更新は適用されません。詳細については、「Ultra Light でのトランザクション処理」63 ページを参照してください。

警告
ローのプライマリ・キーは更新できません。代わりに、ローを削除して新しいローを追加してください。

ローの挿入

ローの挿入手順は、ローの更新手順とほぼ同じです。ただし、挿入操作の場合は、テーブル内のローをあらかじめ指定する必要はありません。テーブルへのローの挿入順序に意味はありません。

例

次のコードでは、新しいローが挿入されます。

```
t.insertBegin();
 t.setInt( id, 3 );
 t.setString( lname, "Carlo" );
 t.insert();
```

カラムの値を設定しない場合、そのカラムにデフォルト値があるときはデフォルト値が使用されます。カラムにデフォルトがない場合は、次のエントリが使用されます。

- NULL 入力可能なカラムの場合は NULL
- NULL 入力不可の数値カラムの場合は 0
- NULL 入力不可の文字カラムの場合は空の文字列
- 明示的に値を NULL に設定するには、setNull メソッドを使用します。

更新操作の場合は、コミットを実行したときに、永続的な記憶領域のデータベースに挿入が適用されます。autoCommit モードでは、insert メソッドの一部としてコミットが実行されます。

ローの削除

ローの削除手順は、ローの挿入や更新よりも簡単です。挿入モードや更新モードに対応する削除モードはありません。

次の手順は、ローを削除します。

❖ ローを削除するには、次の手順に従います。

1 削除するローに移動します。

2 Table.delete() メソッドを実行します。

```
t.delete();
```

Ultra Light でのトランザクション処理

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

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

Connection.AutoCommit プロパティを false に設定すると、複数文のトランザクションを使用できます。たとえば、2 つの口座間で資金を移動するアプリケーションでは、振込み元の口座からの引き落としと、振り込み先口座への振り込みを、両方とも完了するか、どちらも完了しないかのいずれかです。

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

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.Connection** クラス」を参照してください。

スキーマ情報へのアクセス

API のオブジェクトは、テーブル、カラム、インデックス、同期パブリケーションを表します。各オブジェクトには、そのオブジェクトの構造情報へアクセスするための schema プロパティがあります。

API によるスキーマの変更はできません。スキーマに関する情報の取得のみが可能です。

スキーマの修正については、「データベース・スキーマのアップグレード」43 ページを参照してください。

次のスキーマ・オブジェクトと情報にアクセスできます。

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

  DatabaseSchema オブジェクトを取得するには、Connection.schema にアクセスします。

  詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.Connection**」を参照してください。

- **TableSchema** このテーブル内のカラムとインデックスの数と名前。

  TableSchema オブジェクトを取得するには、Table.schema にアクセスします。

- **IndexSchema** インデックス内のカラムに関する情報。インデックスには直接に対応するデータがないため、個別の Index クラスはなく、IndexSchema クラスのみが存在します。

  IndexSchema オブジェクトを取得するには、TableSchema.getIndex、TableSchema.getOptimalIndex、または TableSchema.getPrimaryKey メソッドを呼び出します。

- **PublicationSchema**　パブリケーションに含まれるテーブルとカラムのリスト。パブリケーションもスキーマのみで構成されているため、Publication オブジェクトは存在しません。

  PublicationSchema オブジェクトを取得するには、DatabaseSchema.getPublicationSchema メソッドを呼び出します。

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.TableSchema**」を参照してください。

エラー処理

Java の標準エラー処理機能を使用して、エラーを処理できます。ほとんどのメソッドは、java.sql.SQLException エラーをスローします。SQLException.getErrorCode() を使用して、このエラーに割り当てられた SQLCode 値を取得できます。SQLCode エラーは、エラー・タイプを示す負の数字です。

エラー・コードのリストについては、『ASA エラー・メッセージ』>「ASA エラー・メッセージ」を参照してください。

同期後は、接続の SyncResult プロパティを使用して詳細なエラー情報を取得できます。

詳細については、次の項目を参照してください。

- 『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.SyncResult**」

ユーザ認証

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

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

詳細については、『Ultra Light データベース・ユーザーズ・ガイド』>「Ultra Light のユーザ認証」を参照してください。

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

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. Connection.grantConnectTo メソッドを使用して、希望するパスワードによるデータベースへのアクセスをユーザに付与します。

   新規ユーザを追加する場合も、既存のユーザのパスワードを変更する場合も、この手順は同じです。

   詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**ianywhere.native_ultralite.Connection**」を参照してください。

❖ 既存のユーザを削除するには、次の手順に従います。

1. DBA 権限のあるユーザとしてデータベースに接続します。
2. Connection.revokeConnectFrom メソッドを使用して、ユーザの接続権限を取り消します。

Ultra Light アプリケーションの同期

Ultra Light アプリケーションを中央の統合データベースと同期します。同期には、SQL Anywhere Studio に付属の Mobile Link 同期ソフトウェアが必要です。

この項では、同期の概要について簡単に説明し、Native Ultra Light for Java のユーザにとって特に関心があるいくつかの機能について説明します。

同期の詳細な説明については、『Mobile Link クライアント』> 「Ultra Light クライアント」を参照してください。

また、CustDB サンプル・アプリケーションには、同期の実例もあります。詳細については、SQL Anywhere 9 インストール環境の *Samples¥Ultralite¥NativeUltraLiteForJava¥CustDB* サブディレクトリを参照してください。

Native Ultra Light for Java は、TCP/IP、HTTP、HTTPS 通信による同期をサポートします。同期は、Ultra Light アプリケーションによって開始されます。いずれの場合でも、SyncParms オブジェクトのプロパティを使用して同期を制御します。

注意
HTTPS を使用して同期するには、別途セキュリティ・オプション・ライセンスが必要です。このオプションのご注文については、ご購入いただいた販売代理店または弊社営業担当までご連絡ください。

詳細については、『SQL Anywhere Studio の紹介』> 「SQL Anywhere Studio へようこそ」を参照してください。

アプリケーションへの ActiveSync 同期の追加

この項では、ActiveSync を Native Ultra Light for Java アプリケーションに追加する方法について説明します。また、エンド・ユーザのコンピュータ上の ActiveSync で使用するアプリケーションの登録方法についても説明します。ActiveSync は、Jeode および CrEme Java VM を通じて Windows CE デバイスで使用できます。

CrEme VM を使用している場合、ActiveSync を使用するには、*¥Windows¥CrEme¥lib¥crmex.jar* をクラスパスに追加する必要があります。

同期には、SQL Anywhere Studio が必要です。ActiveSync 同期を設定する概要については、『Mobile Link クライアント』>「ActiveSync を使用するアプリケーションの配備」を参照してください。アプリケーションに同期を追加する概要については、『Mobile Link クライアント』>「Ultra Light クライアント」を参照してください。

ActiveSync 同期を開始できるのは、ActiveSync 自体だけです。デバイスがクレードルにある場合や、[ActiveSync] ウィンドウで同期コマンドが選択された場合に、ActiveSync は同期を開始します。

ActiveSync が同期を開始すると、Ultra Light アプリケーションがまだ実行されていない場合は Mobile Link ActiveSync プロバイダが Ultra Light アプリケーションを起動し、メッセージを送信します。Mobile Link プロバイダからのメッセージを受信し、処理するには、ActiveSyncListener をアプリケーションに実装します。アプリケーションは、setActiveSyncListener メソッドを使用してリスナ・オブジェクトを指定する必要があります。ここで、MyAppClassName は、アプリケーションのユニークな Windows クラス名です。

```
dbMgr.setActiveSyncListener(
    listener, "MyAppClassName" );
```

詳細については、『Native Ultra Light for Java ユーザーズ・ガイド』>「Native Ultra Light for Java API リファレンス」の「**anywhere.native_ultralite.DatabaseManager**」を参照してください。

Ultra Light は、ActiveSync メッセージを受信すると、指定されたリスナの activeSyncInvoked(boolean) メソッドを別のスレッド上で呼び出します。マルチスレッドの問題を回避するには、activeSyncInvoked(boolean) メソッドでユーザ・インタフェースにイベントを通知してください。

マルチスレッド・アプリケーションを使用する場合は、スレッドごとに別々の接続を使用し、Java **synchronized** キーワードを使用して、アプリケーションの共有オブジェクトにアクセスします。activeSyncInvoked() メソッドは、使用する接続の syncInfo ストリームに StreamType.ACTIVE_SYNC を指定してから Connection.synchronize() を呼び出します。

アプリケーションを登録するときは、次のパラメータを設定します。

- **クラス名** Connection.setActiveSyncListener メソッドでアプリケーションが使用したクラス名と同じクラス名
- **パス** Jeode VM のパス *(¥Windows¥evm.exe*)
- **引数** クラス・パス (-cp) とその他の Jeode コマンド・ライン引数、アプリケーション名とアプリケーションの引数を含みます。

  ActiveSync のアクティブ化を示すユニークな引数を指定すると、特別な起動シーケンスを実行でき、ActiveSync 同期の完了時にアプリケーションが閉じられます。

CustDB と ActiveSync

Native Ultra Light for Java バージョンの CustDB サンプルでは、ソケットと ActiveSync を使用して、アプリケーション・メニューから同期の例を提供します。

このサンプルのソース・コードは、SQL Anywhere ディレクトリの下の *Samples¥NativeUltraLiteForJava¥CustDB¥Application.java* にあります。この項では、このサンプルのコードについて説明します。

- CustDB は引数を解析して、Mobile Link プロバイダによって ActiveSync 用に起動されたかどうかを示す特殊なフラグを確認します。ActiveSync 用に起動されたアプリケーションは同期の

完了時に停止されるため、アプリケーションが Mobile Link プロバイダによって起動されたことを確認できれば、フォームの入力などを省いて初期化を簡素化できます。

```
// Normal versus Active sync launch
 boolean isNormalLaunch = true;
 int alen = args.length;
 if( alen > 0 ) {
   String asflag = args[ alen - 1 ].toUpperCase();
   if( asflag.compareTo( "ACTIVE_SYNC_LAUNCH" ) ==
0) {
     isNormalLaunch = false;
     --alen;
   }
```

- 通常の起動 (ActiveSync による起動以外) の場合、CustDB は接続を初期化し、従業員 ID を決定します。その後、リスナを指定し、メイン・フォームをロードして、ActiveSync のための初期化を行います。ActiveSync 用に起動した場合、CustDB は ActiveSync 同期の実行後に停止します。

```
if( isNormalLaunch ) {
   db.initActiveSync( "JULCustDB", main );
   db.getOrder( 1 );
 } else {
   // ActiveSync launch
   db.activeSync( false );
   main.quit();
 }
 public void initActiveSync(
    String appName, ActiveSyncListener listener )
 {
   DEBUG( "initActiveSync" );
   _conn.setActiveSyncListener( appName, listener
);
 }
 public void activeSync( boolean useDialog )
 {
   try {
     // Change stream
     _conn.syncInfo.setStream(
                      StreamType.ACTIVE_SYNC );
     // since if "stream=" not in parms,
     //it defaults to tcpip, no
     // need to change stream parms
```

```
        _conn.synchronize( useDialog );
        freeLists();
        allocateLists();
        skipToValidOrder();
      } catch( SQLException e ) {
        System.out.println(
        "Can't synchronize, sql code=" +
        e.getErrorCode()
          );
      }
    }
```

- Application クラスに ActiveSyncListener インタフェースを実装すると、実行中のアプリケーションに ActiveSync 同期を実行するように通知できます。

```
public class Application
 extends Frame
 implements ActionListener,
 // ActiveSyncListener functional only on CE
devices
 ActiveSyncListener
```

- activeSyncInvoked() が呼び出されると、UI スレッドにメッセージが送られます。

```
static final int ACTIVE_SYNC_EVENT_MASK =
    AWTEvent.RESERVED_ID_MAX + 1;
 static class ActiveSyncEvent extends AWTEvent {
    ActiveSyncEvent( Object source ) {
       super( source, ACTIVE_SYNC_EVENT_MASK );
    }

 /** Process ActiveSync message
 */
 public void activeSyncInvoked(
         boolean launchedByProvider ) {
   // This method is invoked on a special thread.
   // Post an event so that active sync
   // takes places on the same thread
   // as the rest of the application.
   DEBUG( "activeSyncInvoked()" );
```

```
    getToolkit().getSystemEventQueue().postEvent(
              new ActiveSyncEvent( this )
    );
    DEBUG( "ActiveSync Event posted" );
  }
```

- UI スレッドは、processEvent を無効にしてメッセージを取得します。

```
/** Intercept my special action events
  *    for ActiveSync
  */
 protected void processEvent( AWTEvent e ) {
   if( e instanceof ActiveSyncEvent ) {
     _db.activeSync( true );
     refresh();
   } else {
     super.processEvent( e );
   }
 }
```

ただし、アプリケーションがイベントを受け取るには、イベント処理を有効に設定します。これは Application のコンストラクタで行われます。

```
// ActiveSync support
 enableEvents( ACTIVE_SYNC_EVENT_MASK );
```

Borland JBuilder でのアプリケーション開発

Borland JBuilder は、Java アプリケーションの開発環境です。Native Ultra Light for Java には、JBuilder 7 および JBuilder 8 との統合が含まれます。この項では、JBuilder 環境での Native Ultra Light for Java の使用方法について説明します。

JBuilder で Native Ultra Light for Java を使用するための準備

JBuilder が事前にインストールされている場合は、Ultra Light セットアップ・プログラムによって JBuilder の統合が有効になります。Ultra Light をインストールした後に JBuilder をインストールした場合は、Ultra Light セットアップ・プログラムを再実行して JBuilder の統合を有効にします。

JDK の設定

Windows CE デバイス上の Jeode VM との互換性のため、Native Ultra Light for Java アプリケーションを開発する場合は、JDK 1.1.8 または Personal Java 1.2 を使用してください。

JBuilder SE と JBuilder Enterprise では、JDK の切り替えが完全にサポートされていますが、JBuilder Personal で編集できるのは 1 つの JDK だけです。

❖ **JDK バージョンを設定するには、次の手順に従います。**

1. JBuilder Personal で、[ツール] ー [JDK の設定] をクリックし、JDK を編集します。
2. JBuilder SE と JBuilder Enterprise で、プロジェクト・ウィンドウ枠のプロジェクト・ファイルを右クリックし、[プロパティ] を選択します。[パス] タブの [JDK] をクリックし、目的の JDK の場所を検索します。

Native Ultra Light for Java 設定の使用

Native Ultra Light for Java 設定は、既存または新規のプロジェクトで使用できます。

❖ JBuilder プロジェクトに Ultra Light 機能を追加するには、次の手順に従います。

1　JBuilder プロジェクトを開きます。

2　Native Ultra Light for Java 設定を追加します。

- [ファイル] ー [新規] を選択します。オブジェクト・ギャラリが表示されます。
- [一般] タブで、[Native Ultra Light のセットアップ] を選択し、[OK] をクリックします。
- Windows CE デバイス上のデータベース名と開発ディレクトリを選択します。[次へ] をクリックします。
- Windows CE ショートカットの名前 ([スタート] メニューに表示される)、JAR 名、メイン・クラスを追加します。[完了] をクリックします。

プロジェクトにリンク・ファイルが追加されます。このリンク・ファイルは、Windows CE デバイス上のアプリケーションを実行するために使用します。

Native Ultra Light for Java 設定は、JBuilder プロジェクトに次の変更を加えます。

- Native Ultra Light for Java を使用可能なライブラリの一覧に追加する
- コード・インサイト・テンプレートのプロジェクト・プロパティを追加する
- 開発環境からアプリケーションを実行するときに *jul9.dll* が指定されるようにランタイム設定を変更する

Native Ultra Light for Java テンプレートの使用

開発時、標準的なコードの一部に Native Ultra Light for Java コードのテンプレートを使用できます。テンプレートを使用するには、.java ファイル内の該当する場所にテンプレート名を入力し、[CTRL + J] キーを押してテンプレートを展開します。

次のテンプレートが用意されています。

- **julimp** Native Ultra Light for Java パッケージをインポートする行を追加します。このテンプレートは、ファイルのインポート・セクションで使用します。
- **juldb** DatabaseManager オブジェクトを宣言するコードを追加します。
- **julconn** データベースに接続するコードを追加します。
- **julskel** juldb コードと julconn コードの両方を `main` メソッドとして追加します。

JBuilder からの Native Ultra Light for Java ユーティリティへのアクセス

JBuilder インタフェースから、次の Native Ultra Light for Java ユーティリティにアクセスできます。

- **スキーマ・ペインタ** Ultra Light スキーマ・ペインタは、データベース定義の作成と編集のためのツールです。

  スキーマ・ペインタを開くには、[ツール] ー [Ultra Light Schema Painter] を選択します。

- **オンライン・ヘルプ** このマニュアルは、インタフェースから利用できます。

  オンライン・ヘルプを開くには、[ヘルプ] ー [ネイティブ Ultra Light リファレンス] を選択します。

第 5 章

Native Ultra Light for Java API リファレンス

この章の内容

Native Ultra Light for Java パッケージは ianywhere.native_ultralite という名前です。「Native Ultra Light for Java API リファレンス」は、SQL Anywhere 9.0 インストール環境の *docs¥javadocs¥NativeUltraLiteForJava* サブディレクトリに Javadoc フォーマットで提供されています。

詳細については、次のクラスを参照してください。

- **ianywhere.native_ultralite.ActiveSyncListener**
- **ianywhere.native_ultralite.AuthStatusCode**
- **ianywhere.native_ultralite.Connection**
- **ianywhere.native_ultralite.ConnectionParms**
- **ianywhere.native_ultralite.CreateParms**
- **ianywhere.native_ultralite.Cursor**
- **ianywhere.native_ultralite.CursorSchema**
- **ianywhere.native_ultralite.DatabaseManager**
- **ianywhere.native_ultralite.DatabaseNameParms**
- **ianywhere.native_ultralite.DatabaseSchema**
- **ianywhere.native_ultralite.IndexSchema**
- **ianywhere.native_ultralite.PreparedStatement**
- **ianywhere.native_ultralite.ResultSet**
- **ianywhere.native_ultralite.ResultSetSchema**
- **ianywhere.native_ultralite.SchemaParms**
- **ianywhere.native_ultralite.SchemaUpgradeData**
- **ianywhere.native_ultralite.SchemaUpgradeListener**
- **ianywhere.native_ultralite.SQLCode**
- **ianywhere.native_ultralite.SQLType**
- **ianywhere.native_ultralite.StreamErrorCode**
- **ianywhere.native_ultralite.StreamErrorID**
- **ianywhere.native_ultralite.StreamType**
- **ianywhere.native_ultralite.SyncParms**
- **ianywhere.native_ultralite.SyncProgressData**
- **ianywhere.native_ultralite.SyncProgressDialog**

- ianywhere.native_ultralite.SyncProgressListener
- ianywhere.native_ultralite.SyncResult
- ianywhere.native_ultralite.Table
- ianywhere.native_ultralite.TableSchema
- ianywhere.native_ultralite.UUID

索引

J

L

M

N

O

P

R

S

T

U

あ

い

え

か

き

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

に

は

ひ

ふ

ま

も

ゆ

り

る

ろ