SYBASE iAnywhere

Ultra Light
データベース管理とリファレンス

改訂 2007 年 3 月

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、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。

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、埋め込みデバイスやモバイル・デバイスのデータを格納、同期することを目的とするすべての開発者を対象にしています。

SQL Anywhere 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 のマニュアル

SQL Anywhere の完全なマニュアルは、各マニュアルをまとめたオンライン形式とマニュアル別の PDF ファイルで提供されます。いずれの形式のマニュアルも、同じ情報が含まれ、次のマニュアルから構成されます。

- **『SQL Anywhere 10 - 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 10 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 10 - 変更点とアップグレード』** このマニュアルでは、SQL Anywhere 10 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 データベースの実行、管理、設定について説明します。管理ユーティリティとオプションのほか、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について説明します。
- **『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。
- **『SQL Anywhere サーバ - SQL リファレンス』** このマニュアルは、SQL Anywhere で使用する SQL 言語の完全なリファレンスです。また、SQL Anywhere のシステム・ビューとシステム・プロシージャについても説明しています。
- **『SQL Anywhere サーバ - プログラミング』** このマニュアルでは、C、C++、Java プログラミング言語、Visual Studio .NET 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。
- **『SQL Anywhere 10 - 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、その診断情報とともに説明します。
- **『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 - 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 - 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。
- **『Mobile Link - サーバ起動同期』** 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またはその他のリモート・アクションの開始を可能にする Mobile Link の機能です。
- **『QAnywhere』** このマニュアルでは QAnywhere について説明します。QAnywhere は、従来のデスクトップ・クライアントやラップトップ・クライアント用のメッセージング・プラットフォームであるほか、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・プラットフォームでもあります。
- **『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。
- **『SQL Anywhere 10 - コンテキスト別ヘルプ』** このマニュアルには、[接続] ダイアログ、クエリ・エディタ、Mobile Link モニタ、SQL Anywhere コンソール・ユーティリティ、インデックス・コンサルタント、Interactive SQL のコンテキスト別のヘルプが収録されています。
- **『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 - AppForge プログラミング』** このマニュアルでは、Ultra Light for AppForge について説明します。Ultra Light for AppForge を使用すると、Palm OS、Symbian OS、または Windows CE を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対してデータベース・アプリケーションを開発、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、モバイル、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 CE、または Windows XP を搭載しているハンドヘルド、モバイル、または埋め込みデバイスに対して Web ベースのデータベース・アプリケーションを開発、配備できます。
- **『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、モバイル、埋め込みデバイスに対してデータベース・アプリケーションを開発、配備できます。

マニュアルの形式

SQL Anywhere 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 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] − [プログラム] − [SQL Anywhere 10] − [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・ディレクトリまたはインストール CD に保存されている HTML マニュアルを参照してください。

- **PDF ファイル** SQL Anywhere の完全なマニュアル・セットは、Adobe Reader で表示できる Adobe Portable Document Format (pdf) 形式のファイルとして提供されています。

  Windows では、PDF 形式のマニュアルはオンライン・マニュアルの各ページ上部にある PDF のリンクから、または Windows の [スタート] メニュー ([スタート] − [プログラム] − [SQL Anywhere 10] − [オンライン・マニュアル - PDF フォーマット]) からアクセスできます。

  UNIX では、PDF 形式のマニュアルはインストール CD にあります。

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 つだけ選択する場合や、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、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。

オペレーティング・システムの表記規則

- **Windows**　デスクトップおよびラップトップ・コンピュータ用の 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 ファミリには Windows Vista や Windows XP も含まれます。
- **Windows CE**　Microsoft Windows CE モジュラ・オペレーティング・システムに基づいて構築されたプラットフォームです。Windows Mobile や Windows Embedded CE などのプラットフォームが含まれます。

  Windows Mobile は Windows CE 上に構築されています。これにより、Windows のユーザ・インタフェースや、Word や Excel といったアプリケーションの小規模バージョンなどの追加機能が実現されています。Windows Mobile は、モバイル・デバイスで最も広く使用されています。

  SQL Anywhere の制限事項や相違点は、基盤となっているオペレーティング・システム (Windows CE) に由来しており、使用しているプラットフォーム (Windows Mobile など) に依存していることはほとんどありません。
- **UNIX**　特に記述がないかぎり、UNIX は Linux プラットフォームと UNIX プラットフォームの両方のことです。

ファイルの命名規則

マニュアルでは、パス名やファイル名などのオペレーティング・システムに依存するタスクと機能を表すときは、通常 Windows の表記規則が使用されます。ほとんどの場合、他のオペレーティング・システムで使用される構文に簡単に変換できます。

- **ディレクトリ名とパス名**　マニュアルでは、ドライブを示すコロンや、ディレクトリの区切り文字として使用する円記号など、Windows の表記規則を使用して、ディレクトリ・パスのリストを示します。次に例を示します。

  ```
  MobiLink¥redirector
  ```

  UNIX、Linux、Mac OS X では、代わりにスラッシュを使用してください。次に例を示します。

  ```
  MobiLink/redirector
  ```

  SQL Anywhere がマルチプラットフォーム環境で使用されている場合、プラットフォーム間でのパス名の違いに注意する必要があります。
- **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、拡張子 *.exe* が付きます。UNIX、Linux、Mac OS X では、実行ファイルの名前には拡張子は付きません。NetWare では、実行ファイルの名前には、拡張子 *.nlm* が付きます。

  たとえば、Windows では、ネットワーク・データベース・サーバは *dbsrv10.exe* です。UNIX、Linux、Mac OS X では、*dbsrv10* になります。NetWare では、*dbsrv10.nlm* になります。
- **install-dir**　インストール・プロセスでは、SQL Anywhere をインストールするロケーションを選択できます。マニュアルでは、このロケーションは *install-dir* という表記で示されます。

インストールが完了すると、環境変数 SQLANY10 によって SQL Anywhere コンポーネントがあるインストール・ディレクトリのロケーション (*install-dir*) が指定されます。SQLANYSH10 は、SQL Anywhere が他の Sybase アプリケーションと共有しているコンポーネントがあるディレクトリのロケーションを指定します。

オペレーティング・システム別の *install-dir* のデフォルト・ロケーションの詳細については、「SQLANY10 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・プロセスでは、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できます。マニュアルでは、このロケーションは *samples-dir* という表記で示されます。

  インストールが完了すると、環境変数 SQLANYSAMP10 によってサンプルがあるディレクトリのロケーション (*samples-dir*) が指定されます。Windows の [スタート] メニューから、[プログラム] – [SQL Anywhere 10] – [サンプル・アプリケーションおよびプロジェクト] を選択すると、このディレクトリで [Windows エクスプローラ] ウィンドウが表示されます。

  オペレーティング・システム別の *samples-dir* のデフォルト・ロケーションの詳細については、「サンプル・ディレクトリ」『SQL Anywhere サーバ - データベース管理』を参照してください。

- **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 では、環境変数を参照するのに、構文 *%envvar%* が使用されます。UNIX、Linux、Mac OS X では、環境変数を参照するのに、構文 *$envvar* または *${envvar}* が使用されます。

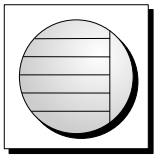
  UNIX、Linux、Mac OS X 環境変数は、*.cshrc* や *.tcshrc* などのシェルとログイン・スタートアップ・ファイルに格納されます。

グラフィック・アイコン

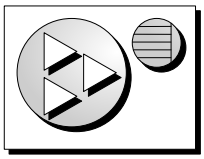
このマニュアルでは、次のアイコンを使用します。

- クライアント・アプリケーション

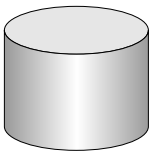
- SQL Anywhere などのデータベース・サーバ



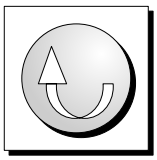
- Ultra Light アプリケーション



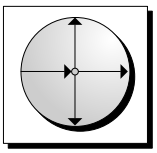
- データベース。高度な図では、データベースとデータベースを管理するデータ・サーバの両方をこのアイコンで表します。



- 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 サーバ、SQL Remote Message Agent などが挙げられます。



- Sybase Replication Server



- プログラミング・インタフェース

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 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。バージョン情報は、コマンド・プロンプトで **dbeng10 -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 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。

パート I. Ultra Light の概要

パート I では、小型デバイス用 Ultra Light リレーショナル・データベース・システムの概要を説明します。

第 1 章

Ultra Light の概要

目次

Ultra Light について

Ultra Light は、モバイル・データベースまたは組み込みデータベースですが、リレーショナル・データベースの主要機能を実装しつつ、占有容量を小さく抑えています。Ultra Light は、全社的なデータ管理用の SQL Anywhere ソリューションの一部としてインストールできます。また、スタンドアロンの組み込みソリューションの一部としてもインストールできます。

Ultra Light を使用すると、モバイル環境で仕事をする現場の従業員や担当者が、コーポレート・ネットワークと通信していないときでも、必要なデータの記録やデータへのアクセスができるようになります。大規模な SQL Anywhere ソリューションの一部である場合は、業務に必要な情報を必要なときに安定して提供する、同期中心のソリューションを実装できます。

Ultra Light は、SQL Anywhere 配備環境の一部として、リアルタイム・データ更新をリモートおよびモバイルの Ultra Light データベース・クライアントにダウンロードできます。Mobile Link サーバを使用すると、データをモバイル・デバイスからネットワーク上の統合データ・ソースに同期できます。

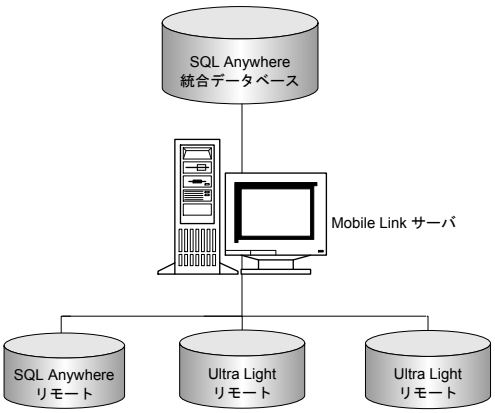
ユーザに付加価値を提供する必要がある場合は、サポートされているアプリケーション開発インタフェースを介して数多くのアプリケーション開発ツールを使用することで、アプリケーションを開発できます。

参照

- 「データ交換テクノロジの概要」 『SQL Anywhere 10 - 紹介』

統合された SQL Anywhere システムでの Ultra Light の使用

Ultra Light を SQL Anywhere 統合データベース環境の一部として配備した場合、ソリューションは、モバイル Ultra Light データベースおよびアプリケーションで構成されるリモート・レイヤと、Mobile Link サーバと適切な数の同期スクリプトで構成される同期レイヤと、SQL Anywhere データベース (サードパーティ製のデータベースもサポート) で構成されるエンタープライズ統合データベースとによる 3 層構造で提供されます。SQL Anywhere をリモート・データベースとして使用することもできます。



Ultra Light クライアント

Ultra Light クライアントはリモートとも呼ばれます。これは、Ultra Light はモバイル・デバイス上で実行され、統合データ・ソースから離れたところにあるデータのローカル・コピーを格納するからです。統合 SQL Anywhere システムでは、コーポレート・ネットワーク上の統合データベースに対するデータの取得や同期を行うことができます。ユーザは Ultra Light アプリケーションをモバイル・デバイス上で使用して、コーポレート・データにアクセスしたり、変更をコーポレート・データにアップロードしたりできます。

参照

- Ultra Light - C/C++ プログラミング 『Ultra Light - C/C++ プログラミング』
- Ultra Light - .NET プログラミング 『Ultra Light - .NET プログラミング』
- Ultra Light - AppForge プログラミング 『Ultra Light - AppForge プログラミング』
- Ultra Light - M-Business Anywhere プログラミング 『Ultra Light - M-Business Anywhere プログラミング』

Mobile Link サーバ

Ultra Light アプリケーションは、Ultra Light アプリケーションに適した同期機能の呼び出しが含まれていると、自動的に Mobile Link 対応になります。Mobile Link は、1 つまたは複数の中央のデータ・ソースと断続的に接続する多数のリモート・クライアント間でデータを同期するように設計されています。

すべての Mobile Link アプリケーションで、Mobile Link サーバが同期処理の主要要素です。通常は、Mobile Link リモート・サイトで Mobile Link サーバへの接続を開くと、同期が開始されます。同期中に、リモート・サイト側の Mobile Link クライアントは、前回の同期後にリモート・データベースに対して行われたデータベースの変更をアップロードできます。Mobile Link サーバは、このデータを受信すると、統合データベースを更新し、変更内容を統合データベースからリモート・データベースにダウンロードできます。

参照

- Mobile Link - クイック・スタート 『Mobile Link - クイック・スタート』
- Mobile Link - クライアント管理 『Mobile Link - クライアント管理』
- Mobile Link - サーバ管理 『Mobile Link - サーバ管理』

リモート／統合データベースとしての SQL Anywhere

SQL Anywhere 統合データベースは、通常、コーポレート・サーバ上に常駐し、同期情報を追跡するよう設定されます。他の中央データは、非リレーショナル・データベース、Web サービス、テキスト・ファイルなど、任意の形式で保存されます。

ただし、大規模な SQL Anywhere データベースを別のリモート・データベースとして使用することも可能です。

参照

- 「統合リモート・データベース」 『SQL Anywhere 10 - 紹介』

- 「SQL Anywhere 統合データベース」 『Mobile Link - サーバ管理』

Ultra Light のデータ・アーキテクチャ

Ultra Light は、SQL Anywhere に似ていますが、規模が非常に小さいリレーショナル・データベースです。Ultra Light はプラットフォームに依存しないモバイル・データベースであり、携帯電話、ハンドヘルド・コンピュータ、埋め込みデバイスなどの小型デバイス用のカスタム・ソリューションを作成できるように設計されています。

ただし、Ultra Light は単なるデータベースではなく、次のレイヤから構成される完全なデータベース管理システムを備えています。

- **開発レイヤ**　Ultra Light ではさまざまなプログラミング・インタフェースがサポートされているので、1 つの配備プラットフォーム、開発ツール、または IT インフラストラクチャ製品に縛られません。

  API の選択については、「Ultra Light プログラミング・インタフェースの選択」 33 ページを参照してください。

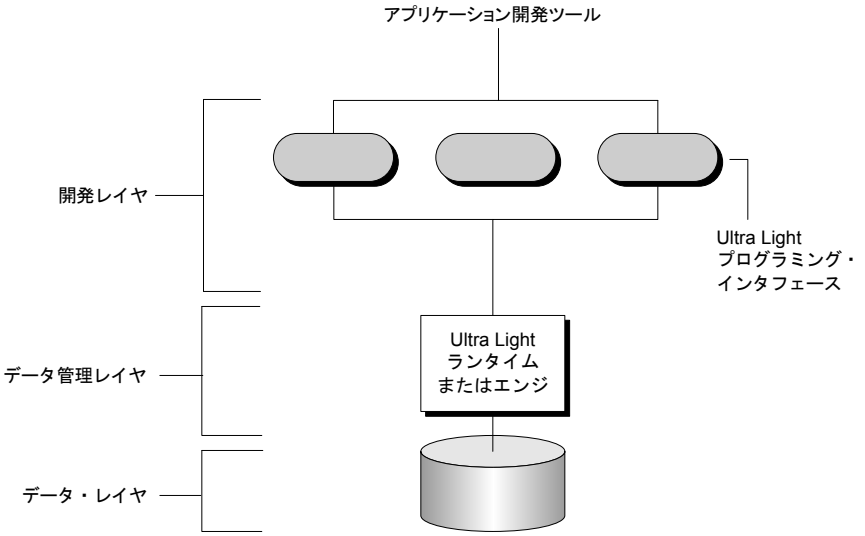
  Ultra Light プロジェクトの管理に役立つように、Ultra Light では、コマンド・ラインから、または Sybase Central や Interactive SQL でグラフィカルな管理ツール (Ultra Light のウィザードなど) から実行できる一連の管理ツールによっても開発がサポートされます。

- **データ管理レイヤと同期クライアント**　プロセス内ライブラリ (ランタイム) または独立プロセス (エンジン) を使用して、Ultra Light データベースに接続できます。どちらのプロセスでも、接続とデータ・アクセス要求を制御します。また、Ultra Light データベースと企業のバックエンド・システムとを結ぶ双方向の同期クライアントが組み込まれています。

  プロセスの選択については、「Ultra Light データ管理コンポーネントの選択」 32 ページを参照してください。

- **データ・レイヤ**　このレイヤは、ファイルまたは一連のデータ・レコード (Palm OS の場合) として保存されるローカル・データ・レポジトリです。「Ultra Light データベース」 11 ページを参照してください。

次の図は、データ、管理、開発の各レイヤを示します。



Ultra Light データベース

デバイスのファイル管理の要件によって、Ultra Light データベースの保管方法と命名規則が決まります。ほとんどのプラットフォームでは、従来のファイルベースの保存が使用されますが、Palm OS のレコードベース・ストアなどでは、データベースの保存方法が異なります。ただし、わかりやすくするために、Ultra Light データベースは通常、ファイルと呼びます。プラットフォームの制限によっては、デスクトップにデバイスを作成してから、同じファイルを複数のプラットフォームに配備できます。

参照

- 「Ultra Light 接続パラメータでのファイル・パスの指定」 85 ページ

Ultra Light データベース・スキーマ

データベースの論理的なフレームワークを**スキーマ**といいます。Ultra Light では、スキーマは、Ultra Light データベースのメタデータを含むシステム・テーブルのカタログとして管理されます。システム・テーブルには、次のメタデータが格納されています。

- インデックス定義。「sysindex システム・テーブル」 244 ページと「sysixcol システム・テーブル」 245 ページを参照してください。
- テーブル定義。「systable システム・テーブル」 242 ページを参照してください。
- カラム定義。「syscolumn システム・テーブル」 243 ページを参照してください。
- パブリケーション定義。「syspublication システム・テーブル」 246 ページと「sysarticle システム・テーブル」 246 ページを参照してください。
- ユーザ名とパスワード。「sysuldata システム・テーブル」 247 ページを参照してください。

DDL 文を使用したスキーマ変更

データベースのスキーマは、データ定義言語 (DDL) 文を使用して変更できます。スキーマの変更には時間がかかる場合があります。たとえば、カラムの型を変更すると、関連付けられているテーブル内のすべてのローが更新される必要があります。したがって、DDL 文は、次の場合にのみ正常に実行されます。

- コミットされていないトランザクションがない。
- スキーマのその他のアクティブな使用 (同期、準備されたが解放されていない文、実行中のデータベース操作など) がない。

このいずれかがあった場合、DDL 文は失敗します。DDL 文を実行すると、DDL 文によるスキーマの変更が完了するまで、その他のデータベースの操作はブロックされます。

参照

- 「Ultra Light SQL 文の概要」 378 ページ

Ultra Light のテンポラリ・ファイル

データベース・ファイルに加えて、Ultra Light では、データベース操作時に**テンポラリ・ファイル**が作成され、管理されます。このファイルの操作や管理は必要ありません。

テンポラリ・ファイルは Ultra Light によってのみ使用され、Ultra Light データベース自体と同じファイル・パス (存在する場合) に作成されます。テンポラリ・ファイルのファイル名は、作成したデータベースと同じですが、次の点が異なります。

- **ファイルベース・プラットフォームの場合**　ファイルの拡張子にチルダが追加されます (例：*.~db*)。たとえば、サンプル・データベース *CustDB.udb* を実行すると、このファイルと同じディレクトリに *CustDB.~db* というテンポラリ・ファイルが作成されます。
- **レコードベース・プラットフォームの場合**　ファイル名の末尾にチルダが追加されます。たとえば、*CustDB.udb* が Palm OS でレコードベースのファイルとして存在した場合、テンポラリ・ファイルは *CustDB.udb~* として作成されます。

ヒント
Ultra Light が実行中でなければ、テンポラリ・ファイルを削除してもデータは失われません。テンポラリ・ファイルには、セッション間で必要な情報は含まれません。

Ultra Light のテンポラリ・テーブル

テンポラリ・テーブルは、アクセス・プランによって、実行中にデータを格納するために、一時的なテーブル、つまりテンポラリ・ワーク・テーブルとして使用されます。このテーブルが存在するのは、アクセス・プランの実行中のみです。一般的に、テンポラリ・テーブルが使用されるのは、サブクエリをアクセス・プランの早い段階で評価する必要がある場合、または使用可能なメモリに中間結果が収まらない場合です。

テンポラリ・テーブルの使用を避けるには、ORDER BY 句または GROUP BY 句で使用されるカラムでインデックスを使用します。インデックスを使用しない場合、Ultra Light がクエリの処理中に使用する情報がテンポラリ・ファイルに保持されます。

テンポラリ・テーブルのデータは、単一の接続に対してだけ保持されます。テンポラリ・テーブルはローとカラムで構成されます。各カラムは電話番号や名前など情報の種類を特定し、各ローはデータのエントリを保持します。

参照

- 「Ultra Light のテンポラリ・ファイル」 12 ページ
- 「Ultra Light クエリ・パフォーマンスの最適化」 38 ページ
- 「クエリ・アクセス・プランを確認する場合」 291 ページ

Ultra Light のトランザクションとステータスの管理

Ultra Light では、データのほかにステータス情報をデータベースに保持します。ステータス情報はを追跡および保持することで、次の管理を行っています。

- 同時接続数。Ultra Light で必要に応じてリソースを共有できるようにするためです。「Ultra Light での同時実行性」 13 ページを参照してください。
- 同期の進行状況のカウント。同期が正常に実行されるのを確認するためです。「Ultra Light の組み込みの同期機能」『Mobile Link - クライアント管理』を参照してください。
- ローのステータス。同期間でのデータの変更内容を追跡することでデータの整合性を維持するためです。「Ultra Light でのローのステータス」 14 ページを参照してください。
- トランザクション。データをいつ、どのようにコミットするかを決定するためです。Ultra Light では、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。「Ultra Light でのトランザクション処理と独立性レベル」 16 ページを参照してください。
- リカバリとバックアップの情報。オペレーティング・システムのクラッシュ、およびストレージ・カードの取り出しやデバイスのリセットなどの Ultra Light の実行中のエンド・ユーザ操作からデータを保護するためです。「Ultra Light でのバックアップとリカバリ」 15 ページを参照してください。

Ultra Light での同時実行性

同時実行性とは、Ultra Light が複数の同時接続を許可することによってリソースを共有する形態のことです。同時要求を正確に処理するには、Ultra Light がデータベースにおける同時実行性を管理する方法について理解する必要があります。

- **複数のデータベース**　1 つの Ultra Light アプリケーションは、1 つまたは複数のデータベースに対する複数の接続を開くことができます。Ultra Light で開くことができるデータベースは、Palm OS と Symbian では最大 8 個、それ以外のプラットフォームでは最大 32 個です。
- **複数のアプリケーション**　Ultra Light データベースを開くことができるのは、一度に 1 つのプロセスだけです。複数のアプリケーション間での同時実行性をサポートする計画の場合は、データ管理コンポーネントとして Ultra Light エンジンを選択してください。「Ultra Light データ管理コンポーネントの選択」 32 ページを参照してください。
- **複数のスレッド**　Ultra Light では、マルチスレッド・アプリケーションがサポートされています。複数のスレッドを使用し、各スレッドが同じまたは異なるデータベースに接続できる、単一のアプリケーションを作成できます。Ultra Light がサポートする同時接続数の上限を超えないよう注意してください。上限は次のとおりです。
  - Palm OS と Symbian OS の場合は 16
  - その他のすべてのプラットフォームの場合は 64

これら上限への SQLCA の影響
一般的に、使用できる SQLCA の数は 31 ですが、Ultra Light.NET API の実装では、データベース・マネージャ 1 つにつき SQLCA を 1 つと、接続 1 つにつき SQLCA を 1 つ使用するようになっています。つまり、この API を使用すると、実際の接続数の上限は 30 になります。

- **複数のトランザクション／要求**　各接続は、進行中のトランザクションを一度に 1 つ使用できます。トランザクションは、1 つの要求または複数の要求で構成されます。トランザクションの実行中に行われたデータの変更は、トランザクションがコミットされるまでデータベースで継続保管されません。トランザクションで行われたデータの変更は、すべてコミットされるか、すべてロールバックされるかのどちらかです。「Ultra Light でのトランザクション処理と独立性レベル」 16 ページを参照してください。

- **同期**　同期は、個別の接続で発生します。アップロード・フェーズでは、Ultra Light アプリケーションは、読み込み専用として Ultra Light データベースにアクセスできます。ダウンロード・フェーズでは、読み込み／書き込みアクセスが許可されますが、アプリケーションがローを変更してからダウンロードによってこのローが変更されようとすると、ダウンロードが失敗し、ロールバックが行われます。Disable Concurrency 同期パラメータを設定して、同期時にデータへのアクセスを無効にすることができます。

  同期に失敗した場合、Ultra Light では、すべてのプラットフォームで再開可能なダウンロードがサポートされます。「失敗したダウンロードの処理」 『Mobile Link - サーバ管理』を参照してください。

参照

- 「Ultra Light クライアント」 『Mobile Link - クライアント管理』
- 「Disable Concurrency 同期パラメータ」 『Mobile Link - クライアント管理』

Ultra Light でのローのステータス

ローのステータス情報の管理は、Ultra Light の強力な機能の 1 つです。テーブルとローのステータスの追跡は、データ同期時に特に重要です。

データへの変更内容

Ultra Light データベースの各ローには、ローのステータスを記録する小さな内部マーカがあります。ローのステータスは、トランザクション処理、リカバリ、同期の制御に使用されます。Ultra Light データベースのローを挿入、更新、または削除すると、その操作内容を反映するようにローのステータスが変更されます。トランザクションがコミットされると、トランザクションに関係のあるすべてのローのステータスに、コミットの内容が反映されます。コミット中に予期せぬ障害が発生すると、トランザクション全体がロールバックされます。次のリストは、その動作をまとめたものです。

- **削除が実行された場合**　対象の各ローが削除されたことを反映してステータスが変更されます。削除のロールバックは、ローを元のステータスにリストアすることと同様、簡単に実行できます。

- **削除がコミットされた場合**　対象のローがメモリから削除されない場合もあります。一度も同期が行われていないローは削除されます。同期が行われたローは削除されません。削除操作

を行うには、まず統合データベースに同期される必要があるからです。次の同期が実行された後、ローはメモリから削除されます。

- **ローが更新された場合**　ローの新しいバージョンが作成されます。古いローと新しいローのステータスが変更され、古いローは表示されなくなり、新しいローが表示されます。
- **ローの更新がコミットされた場合**　トランザクションがコミットされると、トランザクションに関係のあるすべてのローのステータスに、コミットの内容が反映されます。更新が同期されると、ローの古いバージョンと新しいバージョンの両方で競合が検出され、解決が行われます。その後、古いローがデータベースから削除され、新しいローが通常のローになります。
- **ローが追加された場合**　ローがデータベースに追加され、コミット未実行のマークが付きます。
- **追加されたローがコミットされた場合**　ローにコミット済みのマークが付き、統合データベースとの同期が必要であると通知されます。

参照

- 「Ultra Light でのバックアップとリカバリ」 15 ページ
- 「単一のトランザクションまたはグループ化されたトランザクションのフラッシュ」 47 ページ
- 「Ultra Light でのトランザクション処理と独立性レベル」 16 ページ

Ultra Light でのバックアップとリカバリ

Ultra Light データベースを使用しているアプリケーションが予期せずに停止した場合、アプリケーションを再起動すると、Ultra Light データベースは自動的に一貫性を保つ状態にリカバリされます。障害が発生する前にメモリにフラッシュ済みのすべてのコミットされたトランザクションは、Ultra Light データベースに保持されます。障害発生時にフラッシュされていないすべてのトランザクションは、ロールバックされます。

Ultra Light では、リカバリの実行にトランザクション・ログを使用しません。各ローのステータス情報を保存して、リカバリ時のローの処理を決定します。「Ultra Light でのローのステータス」 14 ページを参照してください。

バックアップ

Ultra Light は、システム障害に対する保護機能は備えていますが、メディア障害に対する保護機能はありません。Ultra Light アプリケーションのバックアップ作成に最適な方法として、統合データベースとの同期を行うことが挙げられます。Ultra Light データベースをリストアするには、空のデータベースを作成し、統合データベースと同期してデータを移植します。

ただし、Ultra Light の小規模な配備環境では、手動のバックアップ・メカニズムとしてデータベース・ファイルをデスクトップ・コンピュータにコピーする方法もあります。

注意
Palm OS の場合は、PALM_ALLOW_BACKUP 接続パラメータを true に設定することもできます。これにより、データベースを HotSync を使用してバックアップできます。「Ultra Light PALM_ALLOW_BACKUP 接続パラメータ」 191 ページを参照してください。

参照

- 「単一のトランザクションまたはグループ化されたトランザクションのフラッシュ」 47 ページ

Ultra Light でのトランザクション処理と独立性レベル

トランザクションは、自動的に実行される処理の論理セットです。つまり、トランザクションのすべての処理がデータベースに格納されるか、トランザクションの処理が 1 つもデータベースに格納されないかのどちらかです。Ultra Light ランタイムへの Ultra Light アプリケーションのアクセスは直列化されます。複数のトランザクションを同時に開くことは可能ですが、Ultra Light では一度に 1 つのトランザクションしか処理されません。これにより、アプリケーションでは以下の現象が発生しません。

- トランザクションがブロックされること (デッドロック)。Ultra Light が既存のロー・ロックに基づいて要求をブロックすることはありません。このような事態になった場合、Ultra Light はすぐにエラーを返します。
- 未処理の変更を上書きすること。トランザクションは、他のトランザクションの未処理の変更を上書きすることはできません。トランザクションによってローが変更されると、Ultra Light は、トランザクションが「**コミット**」または「**ロールバック**」されるまでこのローをロックします。ロックによって、他のトランザクションはローを読み込むことはできても、ローを変更できなくなります。

たとえば、A と B という 2 つのアプリケーションが 1 つのデータベースの同じローを読み取っており、どちらもそのデータに基づいて当該ローのカラムの 1 つに入る新しい値を計算するとします。A がローを新しい値で更新し、B が同じローを変更しようとした場合、B はエラーになります。ロックされたローを変更しようとするとエラー SQLCODE SQLE_LOCKED が設定され、削除されたローを変更しようとするとエラー SQLE_NOTFOUND が設定されます。したがって、データの変更を試行した後には SQLCODE 値をチェックするようアプリケーションをプログラムしてください。

エラーの処理方法の詳細については、次のいずれかの項を参照してください。

- Ultra Light for C++ :「エラー処理」 『Ultra Light - C/C++ プログラミング』
- Ultra Light.NET :「エラー処理」 『Ultra Light - .NET プログラミング』
- Ultra Light for AppForge :「エラー処理」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere :「エラー処理」 『Ultra Light - M-Business Anywhere プログラミング』

プログラミングのヒント
C++ API を除くすべての Ultra Light API は「**オートコミット**」モードで動作します実際、一部の API はデフォルトでオートコミットを使用します。
つまり、各トランザクションは操作が終わるたびに自動的にコミットされます。これらのインタフェースのいずれかを使用している場合、複数操作トランザクションを使用するには、オートコミットをオフに設定します。オートコミットをオフに設定する方法は、使用しているプログラミング・インタフェースによって異なります。ほとんどのインタフェースでは、これは接続オブジェクトのプロパティです。
次の項を参照してください。

- Ultra Light.NET：「トランザクションの管理」『Ultra Light - .NET プログラミング』
- Ultra Light for AppForge：「トランザクションの管理」『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「トランザクションの管理」『Ultra Light - M-Business Anywhere プログラミング』

独立性レベルの副次的影響

Ultra Light は、独立性レベル 0 で動作します。Ultra Light では、独立性レベルを変更することはできません。つまり、この制限を念頭においてクエリの結果を解釈する必要があります。

独立性レベル 0 というのは、次のような副次的影響が発生する可能性を意味します。

- SELECT 文を実行するときは、ロック・オペレーションは必要ありません。
- アプリケーションは、コミットされていないデータにアクセスできます (ダーティ・リード)。このシナリオでは、コミットされていないデータベースのローにアプリケーションがアクセスできるため、別のトランザクションによりロールバックされる可能性が発生します。これにより幻ローが発生します。幻ローとは元のクエリの後に追加されたローのことで、これにより、繰り返されて重複したクエリで返された結果セットの内容に相違が生じます。

  ダーティ・リードの影響を示すチュートリアルについては、「チュートリアル：ダーティ・リード」『SQL Anywhere サーバ - SQL の使用法』を参照してください。幻ローを示すチュートリアルについては、「チュートリアル：幻ロー」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
- アプリケーションは、繰り返し不可能読み出しを実行できます。このシナリオでは、アプリケーションがデータベースからローを読み出し、他の操作の実行に移ります。そこへ、別のアプリケーションがローの更新や削除を行い、変更をコミットします。最初のアプリケーションが元のローを再度読み出すと、更新された情報が取得されるか、元のローが削除されたことを発見します。

  繰り返し不可能読み出しの影響を示すチュートリアルについては、「チュートリアル：繰り返し不可能読み出し」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

例

たとえば、A と B という 2 つの接続があり、それぞれにトランザクションがあるとします。

1. 接続 A がクエリの結果セットを処理する場合、Ultra Light によって、現在のローのコピーが「**フェッチ**」されてバッファに格納されます。

   注意
   ローを読み込んだりフェッチしたりしても、ローはロックされません。接続 A がローをフェッチしても変更はしない場合、接続 B はこのローを変更できます。

2. 接続 A は現在のローを変更するときに、バッファ内のコピーを変更します。接続 A が Update メソッドを呼び出すか結果セットを閉じると、バッファ内のコピーはデータベースに書き戻されます。
3. このローに対して書き込みロックが掛けられ、他のトランザクションがこのローを変更できなくなります。この変更は、接続 A がコミットを実行するまで、コミットされないままになります。
4. 変更によっては、接続 B が現在のローをフェッチすると、次の現象が発生する可能性があります。

接続 A による変更	結果[1]
ローの削除	接続 B は結果セットの次のローを取得します。
ローの変更	接続 B はローの最新のコピーを取得します。

1 接続 A および B で使用されたクエリには、テンポラリ・テーブルは含まれていません。テンポラリ・テーブルは、他の副次的な影響を与える可能性があります。

Ultra Light の機能と制限事項

Ultra Light には、モバイルまたは組み込みコンピューティング・ソリューションに適した数多くの組み込み機能が用意されています。ただし、Ultra Light をコンパクトにするには、データベースのサイズを小さくする必要があります。

Ultra Light の機能比較

特定の機能や基準を参考にして、配備環境に適切なデータベースのタイプを決定できます。

Ultra Light データベースおよび管理システムの占有容量は小さく押さえられています。C/C++ バージョンでのアプリケーション・サイズの追加分は 400 ー 500 KB 程度です。一方、SQL Anywhere のデータベース、サーバ、および同期クライアントによる追加分は 6 MB ほどになります。Ultra Light では SQL Anywhere に含まれる一部の機能 (トリガやストアド・プロシージャなど) はサポートされていませんが、リレーショナル・データベースの基本機能はすべて含まれています。

SQL Anywhere と Ultra Light の機能比較

どちらのデータベースが適しているかがわからない場合は、SQL Anywhere と Ultra Light でサポートされている機能を比較してください。X は、サポートされている機能を示します。

必要な機能	SQL Anywhere	Ultra Light	考慮事項
トランザクション処理、参照整合性、マルチテーブル・ジョイン	X	X	
トリガ、ストアド・プロシージャ、ビュー	X		
外部ストアド・プロシージャ (呼び出し可能な外部 DLL)	X		
組み込みの参照整合性とエンティティ整合性	X	X	削除と更新がカスケードされる宣言参照整合性は、Ultra Light データベースではサポートされていない機能です。ただし、同期中はこの目的で削除が自動的にカスケードされます。「外部キー循環に関連する同期の問題の防止」『Mobile Link - クライアント管理』と「Table Order 同期パラメータ」『Mobile Link - クライアント管理』を参照してください。
カスケード更新および削除	X	制限あり	
動的な複数データベースのサポート	X	X[1]	

必要な機能	SQL Anywhere	Ultra Light	考慮事項
マルチスレッド・アプリケーションのサポート	X	X	
ローレベルのロック	X	X	
XML のアンロードとロードのユーティリティ		X	Ultra Light では、XML のロードとアンロードを行うために別々の管理ツールを使用します。ランタイムには組み込まれていません。「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページと「Ultra Light データベースの XML へのアンロード・ユーティリティ (ulunload)」 229 ページを参照してください。
XML のエクスポートとインポートのユーティリティ	X		SQL Anywhere では、SQL 文を使用して XML にデータをエクスポートまたはインポートします。また、dbunload を使用して、データをエクスポートすることもできます。「データのインポートとエクスポート」 『SQL Anywhere サーバ - SQL の使用法』を参照してください。
SQLX 機能	X		
SQL 関数	X	X	Ultra Light アプリケーションでは、すべての SQL 関数を使用できるわけではありません。サポートされていない関数を使用した場合は、「**Ultra Light では使用できない機能です。**」というエラーが発生します。「Ultra Light SQL 関数のリファレンス」 295 ページを参照してください。
SQL 文	X	制限あり	SQL Anywhere と比較すると、Ultra Light では SQL 文のスコープに制限があります。「Ultra Light SQL 文のリファレンス」 377 ページを参照してください。
統合 HTTP サーバ	X		
データベース・ファイルとネットワーク通信の強力な暗号化	X	X	
イベントのスケジュールと処理	X		

1 Ultra Light エンジンと組み合わせた場合のみ

必要な機能	SQL Anywhere	Ultra Light	考慮事項
高パフォーマンス、セルフチューニング、コストベースのクエリ・オプティマイザ	X		Ultra Light にはクエリ・オプティマイザがありますが、SQL Anywhere のオプティマイザほど機能が豊富ではありません。したがって、Ultra Light のオプティマイザでは、複雑なクエリでは、SQL Anywhere オプティマイザほどパフォーマンスが上がらない可能性があります。ただし、単純なクエリを実行している場合は、Ultra Light の動作は高速になります。「Ultra Light のクエリ・アクセス・プラン」 291 ページを参照してください。
複数のスレッド対応 API	X	X	Ultra Light には独特の柔軟なアーキテクチャがあるので、アプリケーション開発者は、変化する配備環境や複数の異なる配備環境を対象としたアプリケーションを作成できます。「Ultra Light プログラミング・インタフェースの選択」 33 ページを参照してください。
カーソルのサポート	X	X	
高度なキャッシュ管理システムによる動的なキャッシュのサイズ設定	X		Ultra Light ではキャッシュのサイズ設定は静的です。それでも、Ultra Light ではデータベースの起動時にキャッシュ・サイズを設定できるので、キャッシュ・サイズを適切に調整できます。「Ultra Light CACHE_SIZE 接続パラメータ」 170 ページを参照してください。
システムまたはアプリケーションの障害後のデータベースのリカバリ	X	X	
バイナリ・ラージ・オブジェクト (BLOB) のサポート	X	X	Ultra Light では、BLOB のインデックス付けまたは比較を実行できません。
Windows パフォーマンスモニタの統合	X		
オンラインのテーブルとインデックスの断片化解除	X		
オンライン・バックアップ	X		

必要な機能	SQL Anywhere	Ultra Light	考慮事項
500 KB 程度の小さい占有容量		X	小型デバイスでは、比較的低速のプロセッサを使用していることがあります。Ultra Light では、これらのデバイスを対象としたアルゴリズムとデータ構造が使用されているので、Ultra Light はパフォーマンスが高く、メモリ使用量が少なくなっています。
Palm OS と Symbian OS をどちらもサポート		X	
デスクトップから Windows CE デバイスへの直接デバイス接続		X	SQL Anywhere データベースでは、Windows CE デバイスに配備するデータベースにデスクトップ接続できるようになるには、データベース・サーバが必要です。Ultra Light では、接続文字列のプレフィクスとして **WCE: ¥** を使用するだけで済みます。「Windows CE」 86 ページを参照してください。
インデックスの使用による高パフォーマンスの更新と検索	X	X	Ultra Light には、テーブルの検索時にインデックスを使用するか、ローを直接スキャンするかを決定するメカニズムがあります。 また、インデックスをハッシュしてデータ検索を高速化することもできます。「Ultra Light でのインデックス・パフォーマンスの考慮事項」 64 ページを参照してください。
Oracle、DB2、Sybase Adaptive Server Enterprise、または SQL Anywhere との同期	X	X	
組み込みの同期		X	SQL Anywhere とは異なり、Ultra Light では、クライアント・エージェントによって同期を容易にする必要がありません。同期は Ultra Light ランタイムに組み込まれているので、配備する必要があるコンポーネントが最小限に抑えられます。「Ultra Light クライアント」『Mobile Link - クライアント管理』を参照してください。
プロセス内実行		X	
計算カラム	X		
宣言されたテンポラリ・テーブル／グローバル・テンポラリ・テーブル	X		

必要な機能	SQL Anywhere	Ultra Light	考慮事項
システム関数	X		Ultra Light では、プロパティ関数を含む SQL Anywhere のシステム関数はサポートされていません。Ultra Light アプリケーションにこれらの関数を含めることはできません。
timestamp カラム	X	X	SQL Anywhere の Transact-SQL のタイムスタンプ・カラムは DEFAULT TIMESTAMP がデフォルトで作成されます。 Ultra Light のタイムスタンプ・カラムは DEFAULT CURRENT TIMESTAMP がデフォルトで作成されます。したがって、Ultra Light では、ローの更新時にタイムスタンプが自動的に更新されません。
ユーザベースのパーミッションのスキームによるオブジェクトベースの所有権とアクセスの決定	X		Ultra Light は、認証システムの必要がない、単一ユーザのデータベース向けに設計されています。ただし、最大 4 つのユーザ ID とパスワードを設定し、認証のためだけに使用できます。これらのユーザは、すべてのデータベース・オブジェクトにアクセスできます。「ユーザ認証の役割」 79 ページを参照してください。

Ultra Light の制限事項

ほとんどのモバイル・デバイスや埋め込みシステムに必要なメモリ、CPU、記憶容量は、制限が厳しいので、Ultra Light の方が適したデータベース・オプションになります。

Ultra Light の制限と SQL Anywhere の制限の比較については、「SQL Anywhere のサイズと数の制限」 『SQL Anywhere サーバ - データベース管理』を参照してください。

データ・サイズとオブジェクトの制限事項

統計情報	Ultra Light の最大値
接続数／データベース	単一スレッド・アプリケーションで最大 14
同時に開いている接続数	Palm OS と Symbian OS の場合は最大 8 それ以外の場合は最大 32
アプリケーションあたりの同時接続数の合計	Palm OS と Symbian OS の場合は最大 16 それ以外の場合は最大 64
SQL Communication Area	最大 31

統計情報	Ultra Light の最大値
ファイルベースの継続保管 (データベース・サイズ)	2 GB ファイル、または OS の制限によるファイル・サイズ
Palm Computing Platform データベース・サイズ	128 MB (主記憶装置) 2 GB (拡張カード・ファイル・システム)
テーブルあたりのロー数	最大 1600 万[1]
データベースあたりのロー数	継続保管によって制限される。
テーブルの大きさ	データベース・サイズによってのみ制限される。
データベースあたりのテーブル数	データベース・サイズによってのみ制限される。
テーブルあたりのカラム数	ロー・サイズはページ・サイズによって制限されるため、テーブルあたりのカラム数の上限は、実際にはページ・サイズによって決まる。通常は 4000 よりもかなり小さい。
テーブルあたりのインデックス数	データベース・サイズによってのみ制限される。
トランザクションあたりの参照テーブル数	制限なし
ストアド・プロシージャの長さ	不適用
データベースあたりのストアド・プロシージャ数	不適用
データベースあたりのトリガ数	不適用
ネスト	不適用
パブリケーション数	32 パブリケーション
ページ・サイズ	16 KB

1 場合によっては、ローに対する変更 (削除および更新) とその他のステータス情報がロー・データとともに保持されます。これにより、変更が同期されます。必然的に、同期間のテーブルのトランザクション数によっては、またはテーブルがまったく同期されない場合は、実際のローの制限が 1600 万より小さくなります。「Ultra Light のトランザクションとステータスの管理」 13 ページを参照してください。

統計情報	Ultra Light の最大値
ロー・サイズ	パックされた各ローの長さはページ・サイズ以下である必要があります。「ローのパックとテーブル定義」 90 ページを参照してください。 文字列がカラム・サイズよりも短い場合は埋め込みなしで格納されます。long binary と long varchar として宣言されたカラムは個別に格納されるため、除外されます。
long binary と long varchar のサイズ	データベース・サイズによってのみ制限される。
準備文のサイズ	準備文のサイズは、64 KB 以下。 SQL クエリの結果セットで参照される各新規ローの宣言済みサイズと、テーブルのローの宣言済みサイズが含まれます。

パート II. 計画と設計

パート II では、Ultra Light の配備の計画と設計を行う方法について説明します。

第 2 章

Ultra Light の計画

目次

Ultra Light の考慮事項

Ultra Light ソリューションを実装することにした場合は、次の点を検討する必要があります。

- 配備環境の柔軟性またはスケーラビリティ。「Ultra Light のスケーラビリティ」 31 ページを参照してください。
- Ultra Light データベースに接続する必要があるアプリケーション数。同時接続数によって、Ultra Light プロセス内ランタイムと Ultra Light エンジンのどちらを使用するかが決まります。これらの違いについては、「Ultra Light データ管理コンポーネントの選択」 32 ページを参照してください。
- サポートするプラットフォーム。これは、アプリケーションのプログラミングに使用できる API に関係があります。「Ultra Light プログラミング・インタフェースの選択」 33 ページを参照してください。
- データベースを実行するプラットフォーム。ファイル・フォーマットは統合されているので、複数のプラットフォームで実行できるデータベースを作成できる場合があります。「Ultra Light データベースの作成と設定」 53 ページを参照してください。

ヒント
複数のプラットフォームに適したファイル・フォーマットを作成する必要がある場合は、Sybase Central の [データベースの作成] ウィザードを使用すると、これが可能かどうかを確認できます。

Ultra Light のスケーラビリティ

Ultra Light データ管理システムが必要ではあるが、データベースの完全な機能を犠牲にしたくない場合は、リソースが限られたデバイスで Ultra Light ソリューションを実装し、Mobile Link 同期テクノロジを使用して統合データベースに同期します。

Ultra Light データベースのデータを、SQL Anywhere などの中央の統合データベースと、TCP/IP、HTTP、HTTPS などのさまざまなネットワーク・プロトコルを使用して同期できます。統合データベースには、パーソナル・アプリケーション用のデスクトップ・データベースも、共有のエンタープライズ・データ用のマルチユーザ・データベースも指定できます。したがって、必要になったときにソリューションを拡大／縮小できます。

どのプラットフォームでどのネットワーク・プロトコル (ストリーム) がサポートされているかについては、「プラットフォーム別 SQL Anywhere 10.0.1 コンポーネント」の Ultra Light の表の「同期」の項を参照してください。

参照

- 「Ultra Light クライアント」『Mobile Link - クライアント管理』
- 「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Mobile Link - クライアント管理』
- 「Ultra Light 同期ユーティリティ (ulsync)」 227 ページ

Ultra Light データ管理コンポーネントの選択

Ultra Light では、占有容量が少ないリレーショナル・データベース・ソリューションを構築できます。このとき、別個のデータベース・サーバを設定する追加オーバヘッドは必要ありません。ただし、Ultra Light プログラミング・インタフェースでは、次の 2 種類のライブラリのうちの 1 つを使用します。

- **Ultra Light インプロセス・ランタイム・ライブラリ**　Ultra Light では、ランタイムとアプリケーションは一体のプロセスであり、データベースはアプリケーションに固有のものとなります。すべてのプラットフォームで、ランタイムによって Ultra Light のデータベースと組み込みの同期処理が管理されます。Ultra Light ランタイムは、一度に最大 14 のデータベースを管理できます。
- **Ultra Light データベース・エンジン・クライアント・ライブラリ**　Windows デスクトップと Windows CE の各プラットフォームには、データベース／アプリケーションとの同時接続を実現する実行プログラムがそれぞれ用意されています。どちらで実行するアプリケーションも、Ultra Light エンジンを使用するときにはクライアント・ライブラリを使用する必要があります。クライアント・ライブラリによって、各アプリケーションが Ultra Light エンジンと通信できます。Ultra Light エンジンには、Ultra Light ランタイムよりも多くのシステム・リソースが必要なので、パフォーマンスが低下する可能性があります。

注意

Ultra Light エンジンがサポートする同時接続数の上限を超えないよう注意してください。次に例を示します。

- Palm OS と Symbian OS の場合はデータベース数が 8、同時接続数が 16
- その他のすべてのプラットフォームの場合はデータベース接続数が 32、同時接続数が 64

また、Ultra Light.NET API で使用できる SQLCA の数は 31 に制限されています (データベース・マネージャ 1 つにつき SQLCA が 1 つと、接続 1 つにつき SQLCA が 1 つ)。ただし、Ultra Light for AppForge などのコンポーネントについては、データベース・マネージャ 1 つにつき SQLCA を 1 つだけです。

参照

- 「Ultra Light での同時実行性」 13 ページ
- 「Ultra Light の機能と制限事項」 19 ページ
- Ultra Light for AppForge：Ultra Light - AppForge プログラミング 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：Ultra Light - .NET プログラミング 『Ultra Light - .NET プログラミング』
- Ultra Light for C/C++ と Embedded SQL：Ultra Light - C/C++ プログラミング 『Ultra Light - C/C++ プログラミング』
- Ultra Light for M-Business Anywhere：Ultra Light - M-Business Anywhere プログラミング 『Ultra Light - M-Business Anywhere プログラミング』

Ultra Light プログラミング・インタフェースの選択

すべての Ultra Light API は、中核となるデータベース機能をさまざまな API を介して公開します。次の API の一部は開発環境と統合されており、プログラミング・タスクを簡単にします。

- C/C++ インタフェース
- C/C++ 用 Embedded SQL
- Ultra Light.NET (C# または VB.NET)
- AppForge Crossfire (C# または VB.NET)
- M-Business Anywhere (JavaScript)

Ultra Light API には、単純なテーブルベースのデータ・アクセス・インタフェースから、複雑なクエリ用の動的 SQL まで、さまざまなデータ・アクセス・モデルがあります。このように Ultra Light のアーキテクチャは柔軟なので、アプリケーション開発者はさまざまな配備環境向けのアプリケーションを作成できます。

注意
このバージョンの Ultra Light では、Pocket Builder をサポートしていません。Sybase PocketBuilder は、SQL Anywhere に含まれていません。詳細については、Sybase にお問い合わせください (http://www.sybase.com/products/developmentintegration/pocketbuilder を参照)。

参照

- Ultra Light for AppForge：Ultra Light - AppForge プログラミング 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：Ultra Light - .NET プログラミング 『Ultra Light - .NET プログラミング』
- Ultra Light for C/C++ と Embedded SQL：Ultra Light - C/C++ プログラミング 『Ultra Light - C/C++ プログラミング』
- Ultra Light for M-Business Anywhere：Ultra Light - M-Business Anywhere プログラミング 『Ultra Light - M-Business Anywhere プログラミング』

♦ プログラミング・インタフェースを選択するには、次の手順に従います。

1. 1 つまたは複数のターゲット・プラットフォームを選択します。Ultra Light では、Palm OS、Windows CE、Windows XP、Windows XP Embedded、Symbian OS がサポートされています。
2. サポートするプラットフォームごとに、API でそのプラットフォームがサポートされているかどうかを確認します。API によってサポートされているプラットフォームが異なります。プラットフォームを問わない開発を行う場合は、すべてのターゲットをサポートする API を選択します。

   このサポート対応表を使用すると、選択できる開発のオプションを簡単に特定できます。

配置ターゲット	MobileVB または C# 用 Ultra Light AppForge Crossfire[1]	Ultra Light for C/C++ と Embedded SQL	Ultra Light.NET[2]	Ultra Light for M-Business Anywhere[3]
Palm OS	バージョン 4 以降	バージョン 4 以降	なし	なし
Windows CE	CE 3.0 以降	CE 3.0 以降	CE 3.0 以降と .NET Compact Framework 1.0.3705 CE 5.0 以降と .NET Compact Framework 2.0	バージョン 3.0 以降
Windows XP Embedded	なし	サポート	.NET Framework 1.0.5000 と 2.0	バージョン 5.0 以降
Symbian OS	バージョン 7.0 と 8.0	バージョン 7.0 と 8.0	なし	なし

1 Microsoft Visual Basic または Microsoft Visual Studio.NET の拡張としての AppForge MobileVB の開発

2 Microsoft Visual Studio.NET の拡張としての開発。ドライバは、ADO.NET バージョン 1.0 と 2.0 の両方をサポートする。

3 JavaScript を使用した Ultra Light プログラミングのブラウザベースの配備

3. 次の影響を検討し、それに合わせて選択した内容を最終決定します。

   SQL Anywhere の互換性 SQL Anywhere とデータベースの互換性が問題であれば、次の点を検討します。

   - SQL Anywhere の Embedded SQL のサポートには、Ultra Light と SQL Anywhere の各データベースに共通のプログラミング・インタフェースがあります。
   - ADO.NET には、Ultra Light のコンポーネントと SQL Anywhere で共有される共通のプログラミング・モデルがあります。

   特に、両方のデータベースを使用できる Windows CE などのプラットフォームでは、共通のインタフェースを使用できれば便利です。Ultra Light から完全な機能を備えたより強力な SQL Anywhere データベースに移行する必要がある場合、Embedded SQL または ADO.NET を使用すると、アプリケーションの移行作業がより簡単になります。

ヒント
共通のインタフェースが存在する場合でも、アプリケーションの再生時に抽象データ・アクセス・レイヤを作成することをおすすめします。

単純化された配備　Ultra Light 配備の単純化が問題であれば、M-Business Anywhere API を使用したプログラミングを検討します。エンドユーザは、Ultra Light アプリケーションとデータベースを同時にダウンロードできます。

アプリケーション・サイズ　占有領域ができるだけ小さいアプリケーションを作成することが重要な場合は、C/C++ API を使用してアプリケーションをプログラミングします。このようなアプリケーションは、通常、高パフォーマンスを実現しつつ、アプリケーション・ファイル・サイズが抑えられます。

アプリケーションのパフォーマンス　各 API で実現されるパフォーマンスは異なります。たとえば、Ultra Light.Net より MobileVB の Ultra Light for AppForge の方がパフォーマンスが優れています。また、このどちらよりも Ultra Light C++ の方が優れており、さまざまな要因によっては、Ultra Light Embedded SQL の方が Ultra Light C++ よりパフォーマンスが高い場合もあります。

第 3 章

Ultra Light のパフォーマンスと最適化

目次

Ultra Light クエリ・パフォーマンスの最適化

Ultra Light では、優れた SQL クエリ・パフォーマンスを自動的に提供します。Ultra Light がインデックス・スキャン、ダイレクト・ページ・スキャン、テンポラリ・テーブルを実装する方法は、この製品から最高のパフォーマンスを引き出すことができる内部的な最適化の方法です。こうした機能は、実行するクエリ・パフォーマンス・テストの結果に応じて調整することも可能です。

インデックス・スキャンの使用

インデックスとは、1 つ以上のテーブル・カラムに含まれているデータ値の順序に基づく、テーブル・ローのポインタ・セットです。インデックスは、データベース・オブジェクトです。インデックスが作成されると、Ultra Light によって自動的に保持されます。1 つ以上のインデックスを作成すると、クエリのパフォーマンスが向上します。インデックスのタイプによっては、ローの値がユニークであることを保証できます。

インデックスは、一部またはすべてのカラムの値を基に、テーブルのローに順序を付けます。インデックスを作成する場合、インデックス付けするカラムを指定する順序が、インデックスでカラムが実際に出現する順序になります。したがって、インデックスを計画的に使用すると、それによってインデックス・カラムの検索パフォーマンスが大幅に向上します。

Ultra Light では、次のインデックスをサポートしています。インデックスは、シングルカラムとマルチカラム (複合インデックス) が可能です。LONG VARCHAR カラムや LONG BINARY カラムをインデックス付けすることはできません。

インデックス	特性
プライマリ・キー	必須。ユニーク・キーのインスタンスです。プライマリ・キーは 1 つだけ持つことができます。インデックス付けされたカラムの値はユニークである必要があります。NULL は不可です。
外部キー[1]	省略可。インデックス付けされたカラムの値は重複していてかまいません。NULL 入力可能かどうかは、NULL を許可するようにカラムが作成されたかどうかで決まります。外部キー・カラムの値は、参照されているテーブルに存在している必要があります。
ユニーク・キー[2]	省略可。インデックス付けされたカラムの値はユニークである必要があります。NULL は不可です。
ユニークでないインデックス	省略可。インデックス付けされたカラムの値は重複していてかまいません。NULL を指定できます。
ユニーク・インデックス	省略可。インデックス付けされたカラムの値が重複していてはいけません。NULL を指定できます。

1 外部キーは、プライマリ・キーまたはユニーク・キーを参照できます。
2 一意性制約と同じです。

パフォーマンスに関するヒント

- 次のようなカラムにインデックスを作成してください。
  - 定期的に検索する値
  - クエリでテーブルのジョインに使用するカラム
  - ORDER BY 句、GROUP BY 句、または WHERE 句で頻繁に使用するカラム
- 複合インデックスを作成する場合、インデックスの最初のカラムは、クエリの述部で最も頻繁に使用されるカラムにしてください。
- インデックスにより生じる更新管理オーバヘッドがデバイスのメモリに負荷をかけすぎていないことを確認してください。
- 不要なインデックスを作成または保持しないでください。インデックスは、カラムのデータが変更されたときに更新する必要があります。したがって、挿入、更新、削除の各操作はインデックスにも実行されます。
- 大規模なテーブルでインデックスを作成してください。
- 冗長なインデックスは作成しないでください。たとえば、カラム (x、y) に対するインデックスをテーブル T に作成した場合に、カラム (x、y、z) に対する別の既存インデックスが T にあると、冗長的になります。

オプティマイザで使用するアクセス方法の決定

Ultra Light オプティマイザは、クエリの最適化に使用するインデックスを選択する際に、優れた最適化方式を取ります。単純なクエリの場合を除き、オプティマイザがクエリ・パフォーマンスの最適化に使用するインデックスをあらかじめ決めることや、そもそもインデックスを使用するかどうかを決めることは、簡単ではありません。複雑さが増すにつれて、選択するインデックスはクエリで必要な句によって変わってきます。場合によっては、FOR READ ONLY 句が存在することで、オプティマイザが、クエリ・パフォーマンスの向上のためにインデックスではなくダイレクト・テーブル・スキャンを選択することもあります。

クエリを最適化する場合、オプティマイザはクエリの要件を確認し、パフォーマンスの向上に使用できるインデックスがないかどうかをチェックします。どのインデックスを使用してもパフォーマンス向上が望めない場合、オプティマイザはインデックスをスキャンせず、テンポラリ・テーブルかダイレクト・ページ・スキャンを使用します。したがって、インデックスを使用して実験し、生成されたクエリ・プランを頻繁にチェックして、次の事項を確認する必要があります。

- オプティマイザによって使用されないインデックスを保持していないこと。
- 作成されるテンポラリ・テーブルの数を最小限に抑えていること。「テンポラリ・テーブルの管理」 44 ページを参照してください。

複雑なクエリの場合には、使用されるインデックスを予測するのはさらに困難です。たとえば、クエリに WHERE 述部があり、ORDER BY 句に加えて GROUP BY 句が存在する場合、インデックスが 1 つだけではクエリの検索条件を満たさない可能性があります。したがって、WHERE 述部の選択性要件を満たすインデックスを作成していても、オプティマイザがそれを使用しない可能性があります。代わりに、オプティマイザは、最も多くの処理を必要とする ORDER BY 条件でのパフォーマンスが最も優れているインデックスを使用する可能性があります。

クエリ・プランのチェック

クエリ・プランは、API 呼び出しを使用してプログラムでチェックしたり、Interactive SQL の [プラン] タブでチェックしたりできます。

- **インデックスが使用されていない場合**　クエリ・プランは次のように表示されます。

  scan(T)

- **テンポラリ・テーブルが使用されている場合**　クエリ・プランは次のように表示されます。

  temp [scan(T)]

- **インデックスが使用されている場合**　インデックス名がクエリ・プランに含まれている場合は次のように表示されます。

  scan (T, *index_name*)

参照

- 「テンポラリ・テーブルの管理」 44 ページ
- 「ダイレクト・ページ・スキャンの使用」 45 ページ
- 「Ultra Light のクエリ・アクセス・プランの表示」 291 ページ
- 「複合インデックスについて」 100 ページ
- 「EXPLANATION 関数 [その他]」 327 ページ
- Ultra Light for C/C++：「GetPlan 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for M-Business Anywhere：「getPlan メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light でのページ・サイズの考慮事項」 68 ページ
- 「Ultra Light page_size プロパティ」 148 ページ

インデックスのハッシュによるクエリ・パフォーマンスのチューニング

「**ハッシュ**」の上限として特定のサイズを選択することでクエリのパフォーマンスをチューニングできます。ハッシュ・キーは、インデックス付けされたカラムの実際の値を表します。インデックスのハッシュ・キーの目的は、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。ロー ID は、ハッシュが使用されない場合は常にインデックス・エントリの一部です。

ロー ID を使用することで、Ultra Light がデータベース・ファイル内の実際のロー・データを見つけることができます。ハッシュ・サイズを 0 に設定する (インデックスのハッシュを無効にする) と、インデックス・エントリにはこのロー ID だけが含まれます。ハッシュ・サイズを 0 以外に設定すると、そのローの変換されたデータすべてまたはその一部を含むハッシュ・キーがロー ID とともにインデックスのページに格納されます。

ただし、ハッシュ・キーに含まれるロー・データの量は、設定した最大ハッシュ・サイズ・プロパティと、カラムのデータ型に実際に必要とされるデータ・サイズとによって決まります。「最適なハッシュ・サイズの選択」 42 ページを参照してください。

ハッシュの例

インデックスのハッシュの値は、インデックス付けされたカラムの実際のロー・データの順序を保持しています。たとえば、Employees テーブルの LastName カラムにインデックスを作成した場合、4 つの名前が次の順序で表示されます。

Anders

Anderseck

Andersen

Anderson

つまり、最初の 6 文字をハッシュした場合、これらのロー値のハッシュ・キーは次のようになります。

Anders

Anders

Anders

Anders

これらのエントリは同じに見えますが、リストの最初の Anders は実際のロー値 **Anders** を表すために使用され、リストの最後の Anders は実際のロー値 **Anderson** を表すために使用されます。

ここで、次の文を考えてみます。

```
SELECT *
FROM Employees
WHERE LastName = 'Andersen'
```

Employees テーブルに Andersen に似た名前ばかりが数多く含まれている場合、現状のハッシュのままでは、パフォーマンスが向上するほどの一意性がありません。これは、Ultra Light がこの文の条件に実際に一致するハッシュ・キーを決定できないためです。このように、インデックスのハッシュ・キーの重複があると、目的のロー ID に一致するテーブル・ローを探さなくてはならず、データをロードしてアンパックし、値を評価する必要が生じます。

パフォーマンスが向上するのは、Ultra Light が各ハッシュ・キーの一意性を区別可能で、クエリ条件の評価がそのままインデックス自体につながる場合のみです。そうなれば、それ以上の処理は不要になります。たとえば、Employees テーブルに何千という名前があっても、6 文字によるハッシュで十分なパフォーマンス向上が得られます。これは、すべてのロー・エントリのインデックスにユニークなハッシュ・キーが適度な数だけ含まれているからです。ただし、Employees テーブルに Andersen に似た名前が突出して含まれている場合は、少なくとも 7 文字でハッシュして、インデックスで使用されるユニークなハッシュ・キーの数を増やす必要があります。そうすると、インデックスのハッシュ・キーは次のようになります。

Anders

Anderse

Anderse

Anderso

これにより、前述の文を使用すると、4 つローの値のうち、アンパックして評価する必要があるのが、4 つではなく 2 つになります。

参照

- 「Ultra Light でのインデックス・パフォーマンスの考慮事項」 64 ページ
- 「最適なハッシュ・サイズの選択」 42 ページ
- 「Ultra Light クエリ・パフォーマンスの最適化 」 38 ページ
- 「Ultra Light インデックスの追加」 102 ページ

最適なハッシュ・サイズの選択

Ultra Light のデフォルトの最大ハッシュ・サイズである 4 バイトは、ほとんどの場合に適切であるように慎重に選ばれています。値を大きくしてロー ID とともに含めるデータを増やすことはできますが、そうすると、インデックスのサイズが大きくなり、複数ページに断片化され、結果的にデータベースのサイズが大きくなる可能性があります。最大ハッシュ・サイズが大きくなることによる影響は、テーブルに含まれるローの数によって異なります。たとえば、ローの数が少なければ、インデックスのハッシュ・キーは、大きくてもインデックス・ページに収まる可能性があります。その場合、インデックスの断片化は起こりません。

最適なハッシュ・サイズを選択するには、この後の各項で説明するように、データ型、ローのデータ、データベースのサイズ (特にテーブルに含まれるローの数が多い場合) を考慮してください。最適なハッシュ・サイズを選択したかどうかを確認する唯一の方法は、ターゲット・デバイスで Ultra Light クライアント・アプリケーションに対してベンチマーク・テストを実行することです。さまざまなハッシュ・サイズによって、アプリケーションとクエリのパフォーマンスにどのように影響するか、さらにデータベースのサイズ自体がどう変化するかを確認する必要があります。

データ型

カラム内の値全体をハッシュする場合は、次の表で、データ型ごとに必要なサイズを確認してください。Ultra Light では、最大ハッシュ・サイズは本当に必要になった場合にのみ使用し、指定した最大ハッシュ・サイズを超えることはありません。カラムのデータ型がバイト制限いっぱいを使用しないかぎり、常により値の小さいハッシュ・サイズを使用します。

データ型	値全体のハッシュに使用されるバイト数
FLOAT、DOUBLE、REAL	ハッシュなし
BIT と TINYINT	1
SMALL INT と SHORT	2
INTEGER、LONG、DATE	4
DATETIME、TIME、TIMESTAMP、BIG	8

データ型	値全体のハッシュに使用されるバイト数
CHAR と VARCHAR	文字列全体をハッシュするには、バイト単位の最大ハッシュ・サイズが、カラムの宣言されたサイズと一致する必要があります。UTF-8 でエンコードされたデータベースでは、宣言されたサイズを常に 2 倍にします。ただし、最大値は 32 バイトです。 たとえば、UTF-8 でエンコードされていないデータベースでカラムを VARCHAR(10) と宣言した場合は、必要なサイズは 10 バイトです。しかし、同じカラムを UTF-8 でエンコードされたデータベースで宣言した場合は、文字列全体のハッシュに使用されるサイズは 20 バイトです。
BINARY	バイト単位の最大ハッシュ・サイズは、カラムの宣言されたサイズと一致する必要があります。 たとえば、カラムを BINARY(30) と宣言した場合は、必要なサイズは 30 バイトです。
UUID	16

たとえば、INTEGER と BINARY (20) と宣言した 2 カラムの複合インデックスに対して最大ハッシュ・サイズを 6 バイトに設定した場合、データ型のサイズの要件に基づいて、次の処理が行われます。

- INTEGER カラムのローの値全体がハッシュされ、インデックスに格納されます。これは、整数のデータ型のハッシュに必要なのは 4 バイトだけだからです。
- ハッシュの最初の 4 バイトは INTEGER カラムで使用されるので、BINARY カラムは最初の 2 バイトだけがハッシュされ、インデックスに格納されます。残りの 2 バイトで、BINARY カラムの適切な量がハッシュされない場合は、最大ハッシュ・サイズを大きくしてください。

ローのデータ

データベースに格納されているデータのローの値も、インデックスのハッシュの有効性に影響します。

たとえば、特定のカラムのエントリ間で共通のプレフィクスを共有している場合は、プレフィクスだけがハッシュされるサイズを選択するとハッシュの効果がなくなる可能性があります。この場合、共通のプレフィクスがハッシュされる以上のサイズを選択する必要があります。共通のプレフィクスが長い場合、値をハッシュしないことも検討してください。

ユニークでないインデックスに重複する値が数多く含まれており、Ultra Light が値全体をハッシュできない場合は、ハッシュによるパフォーマンスの向上は期待できません。

データベース・サイズ

各インデックス・ページには固定のオーバヘッドがありますが、ページのほとんどの領域は実際のインデックス・エントリに使用されます。ハッシュ・サイズを大きくすると、各インデックス・エントリが大きくなり、1 ページに収まるエントリの数が少なくなります。大規模なテーブルになると、大規模なハッシュを使用するインデックスの方が、小規模のハッシュを使用するテーブルやハッシュを使用しないテーブルより、使用するページ数が多くなります。これにより、データベース・ファイルが大きくなり、(場合によっては) パフォーマンスが低下します。こ

れは、データベース・キャッシュで保持できるページ数が固定であるため、キャッシュでページのスワップが発生するからです。

次の表に、インデックスに含まれるデータの格納に必要なページ数にハッシュ・サイズがどう影響するかの概要を示します。

テーブル	ページ・サイズ	ハッシュ・サイズ	エントリ数	必要なページ
テーブル A	4 KB	0	1200	3 ページ
テーブル B	4 KB	32 バイト	116	3 ページ
テーブル C	4 KB	32 バイト	1200 エントリ	11 ページ

参照

- 「Ultra Light でのインデックス・パフォーマンスの考慮事項」 64 ページ
- 「Ultra Light クエリ・パフォーマンスの最適化 」 38 ページ
- 「Ultra Light インデックスの追加」 102 ページ
- 「Ultra Light のデータ型」 262 ページ

最大ハッシュ・サイズの設定

最大ハッシュ・サイズは次の 2 つの方法で設定できます。

- データベースでの最大サイズの*デフォルト*を格納するには、データベースの作成時に max_hash_size というデータベース・プロパティを設定します。デフォルトでインデックスをハッシュしない場合は、この値を 0 に設定します。インデックスをハッシュする場合は、この値を 32 バイト以内の任意の値に変更するか、Ultra Light のデフォルト値 4 バイトを使用できます。
- このパラメータで設定したデフォルトを*変更*する場合は、新しいインデックスを作成するときに特定のハッシュ・サイズを設定します。次の 2 つの方法で設定できます。
  - Sybase Central で、新しいインデックスを作成するときに [最大ハッシュ・サイズ] プロパティを設定します。
  - SQL で、CREATE TABLE 文または CREATE INDEX 文で WITH MAX HASH SIZE 句を使用します。

テンポラリ・テーブルの管理

一般的に、オプティマイザは、クエリ結果を返すことを目的としたテンポラリ・テーブル作成を回避しようとします。これは、最初のローを返す前に、テンポラリ・テーブル全体を移植する必要があるからです。オプティマイザがインデックスを使用する場合、テンポラリ・テーブルの作成は回避されます。オプティマイザがテンポラリ・テーブルを作成するのは最後の手段としてのみです。

作成したインデックスでテンポラリ・テーブルの作成を回避できるかどうかを予測することは困難です。そのため、クエリ・プランは必ずチェックし、作成したインデックスが Ultra Light クエリ・オプティマイザによって実際に使用されているかどうかを確認する必要があります。

参照

- 「Ultra Light のテンポラリ・テーブル 」 12 ページ
- 「オプティマイザで使用するアクセス方法の決定」 39 ページ
- 「Ultra Light のアクセス・プランの解釈」 292 ページ

ダイレクト・ページ・スキャンの使用

ダイレクト・ページ・スキャンは、インデックス・スキャンの代わりに使用されます。ダイレクト・ページ・スキャンが使用されるのは、次の条件が発生した場合です。

- Ultra Light オプティマイザが、結果を効率的に返すために役立つ既存のインデックスがないと判断した。
- Ultra Light オプティマイザが、更新の実行にクエリを使用していないと*確信*した。つまり、文が FOR READ ONLY であると宣言されている場合、またはデータが更新されないことが明らかなようにクエリが記述されている場合です。

Ultra Light では、ローが格納されているページからローを直接読み込むので、クエリ結果は順序に関係なく返されます。その後のクエリ結果の順序は予想できません。そのため、ローの順序を予測可能で決定的にするには、ORDER BY 句を使用して結果に一貫性を持たせる必要があります。一方、順序が重要ではない場合は、ORDER BY 句を省略することで、クエリのパフォーマンスを向上させることができます。

> **注意**
> アプリケーションのプログラムにテーブル API を使用している場合、ダイレクト・ページ・スキャンは使用できません。

Ultra Light がいつページを直接スキャンするか、または結果を返すのにどのインデックスが使用されたかを確認できます。「オプティマイザで使用するアクセス方法の決定」 39 ページを参照してください。

プライマリ・キー・インデックス順序の復元

10.0.1 より前のバージョンの Ultra Light では、Ultra Light のオプティマイザによって他のインデックスが使用されていない場合には、結果を返すのにプライマリ・キーが使用されていました。その結果、ローはプライマリ・キー・インデックスの順序で並びました。

結果をプライマリ・キーの順序で並べる必要がある場合は、クエリを書き直して ORDER BY 句を含める必要があります。この句を使用してローを並べ替えることが現実的ではない場合は、**ORDERED_TABLE_SCAN** 接続パラメータの使用を検討してください。

注意
できるだけ ORDER BY 句を使用することをおすすめします。そうしないと、この Ultra Light 機能によるパフォーマンス上の利点が得られません。

参照

- 「Ultra Light ORDERED_TABLE_SCAN 接続パラメータ」 190 ページ
- 「インデックス・スキャンの使用」 38 ページ
- 「Ultra Light SELECT 文」 401 ページ

単一のトランザクションまたはグループ化されたトランザクションのフラッシュ

コミットされたトランザクションのフラッシュを遅延させることで、Ultra Light でのリカバリ・ポイントを選択できます。アプリケーションのリカバリ・ポイントを独自に選択することで、Ultra Light がコミットを記憶領域に解放したときに、トランザクションまたは作業単位に含まれている SQL 文のサブセットがトリガする追加操作オーバヘッドを的確に制御できるようになります。

Ultra Light では、コミットされたトランザクションは個別にまたはグループとしてフラッシュできます。

デフォルトでは、Ultra Light は操作ベースの方法を使用しており、トランザクションはコミットされると個別にすぐ記憶領域にフラッシュされます。配備環境によっては、負荷が重くなり、トランザクション・スループットが制限される可能性があります。

操作ベースのデフォルト処理によるパフォーマンスの低下を防ぐため、一部の配備環境、特にオートコミット操作に依存しているアプリケーションでは、ステータスベースの方法を使用して、コミットされたトランザクションの記憶領域へのフラッシュに必要な追加オーバヘッドを遅延させています。

- **チェックポイントでの制御**　独自のチェックポイントを設定し、これを使用して、それまでの期間に実行された作業を解放します。チェックポイントは、単一トランザクション内にも、複数のトランザクションにわたるものでも、必要に応じていくつでも使用できます。
- **グループ化して制御**　トランザクション・カウント・スレッショルドかタイムアウト・スレッショルドまたはその両方を使用して、実行した作業を解放できます。

ステータスに応じてコミットのフラッシュを遅延させる方が、トランザクションを記憶領域にすぐにフラッシュする場合より、パフォーマンスへの影響が少なく、アプリケーションの設計もすっきりします。パフォーマンスに関して、アプリケーションは Ultra Light からの応答を待機する必要はありません。コミットと応答のタイムラグは、トランザクションの管理を Ultra Light に依存しているスキャンタイプのアプリケーションでは、大きな影響を及ぼします。

コミット・フラッシュを遅延させることで、作業が完全には完了していないデータへのきめ細かい制御が可能になり、トランザクションへの影響が最小限に抑えられます。たとえば、販売アプリケーションでは、注文が、すべての項目が追加されたり承認されたりする前に、別のアプリケーションに対しても使用可能になります。

ただし、コミット・フラッシュが遅延されるトランザクションのリカバリ性を考慮しておくことが重要です。解放されていないトランザクションはリカバリできません。したがって、アプリケーションのデータ整合性とパフォーマンスとの間のトレードオフを評価しておく必要があります。

参照

- 「Ultra Light でのコミット・フラッシュの考慮事項」 73 ページ
- 「Ultra Light COMMIT_FLUSH 接続パラメータ」 172 ページ
- 「Ultra Light commit_flush_count オプション [テンポラリ]」 160 ページ

- 「Ultra Light commit_flush_timeout オプション [テンポラリ]」 161 ページ
- 「Ultra Light CHECKPOINT 文」 385 ページ
- Ultra Light for Embedded SQL：「ULCheckpoint 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「Checkpoint 関数」 『Ultra Light - C/C++ プログラミング』

Ultra Light のプラットフォーム別の最適化方法

Ultra Light の配備環境によっては、ニーズに応じてデータベースをチューニングするための専用の設定が必要になる場合があります。

Palm OS の初期バージョン

Palm デバイスの初期バージョン (バージョン 4.x など) には RAM が 200 KB ほどしかありません。この制限事項により、Ultra Light 10 の動的メモリ要件に起因する問題が発生することがあります。

プラットフォームごとの方法　対応策として、次の 2 つの方法があります。

- Palm OS 5.x にアップグレードする。この方法を推奨します。
- Ultra Light をバージョン 9.x にダウングレードする。

データベースの最適化方法　次の事項を行っていることを確認してください。

- **CACHE_SIZE パラメータ**　接続の確立にこのパラメータを使用せず、Ultra Light によるデフォルト値の設定を受け入れてください。独自の値を設定すると、同期のための動的メモリ要件に関連して予期しない問題が発生する可能性があります。
- **page_size データベース・プロパティ**　アプリケーションに最低限必要なページ・サイズを使用する新しいデータベースを作成してください。最小ページ・サイズは 1 KB で、Palm OS 用として一般的に使用されているページ・サイズは 2 KB です。

パート III. Ultra Light データベースの使用

パート III では、Ultra Light データベースの作成方法と使用について説明します。

第 4 章

Ultra Light データベースの作成と設定

目次

Ultra Light データベースの作成

採用するデータベースの作成方法に関係なく、作成時には、データベースの特性の制御や定義を行うデータベース・プロパティを設定します。デフォルト値を使用する場合は、データベース・プロパティを設定する必要はありません。ただし、データベースが作成された後に、これらの値を設定し直すことはできません。データベース・プロパティの変更が必要な場合は、データベースを再作成する必要があります。

データベースの作成方法は、次の 2 つのカテゴリに分類できます。

- データベース作成用の Ultra Light 管理ツールを使用してデスクトップで作成する方法。この章では、主にこのカテゴリの方法について説明します。
- Ultra Light の API を使用してデバイス上で作成する方法。デバイス上で作成する方法については、主に各 API に固有の Ultra Light プログラミング・ガイドで説明しています。ここでは、万全を期して、これらのメソッドについて簡単に説明します。

データベースを作成した後は、そのデータベースに接続して、必要なテーブルやその他のデータベース・オブジェクトを構築できます。

複数のプラットフォームでのデータベースの共有

オペレーティング・システムが異なるために設定が異なる場合でも、データベースをデバイス間でコピーできる場合があります。複数のプラットフォーム間でのプロパティの互換性が不明な場合は、Sybase Central で Ultra Light の [データベース作成] ウィザードを使用してデータベースを作成してください。このウィザードによってファイルの互換性の論理が処理されるので、配備デバイスの特定の組み合わせでサポートされないファイルを作成してしまうのを防ぐことができます。

参照

- 「Ultra Light のプラットフォーム別の最適化方法」 49 ページ
- 「Ultra Light で使用する作成時のデータベース・プロパティの選択」 61 ページ
- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「Ultra Light データベースの操作」 89 ページ

デスクトップでの Ultra Light の作成

Ultra Light の [データベース作成] ウィザード　データベースの作成時に設定可能なプロパティをナビゲーションしながら作成する場合は、この方法を使用します。このウィザードを使用すると、選択するターゲットのプラットフォームに基づいて設定できる項目が制限されるので、選択が簡単になります。データベースが作成されると、コマンド・ライン構文が表示されます。この構文は、記録して、後で ulcreate ユーティリティで使用できます。「Sybase Central からの Ultra Light データベースの作成」 55 ページを参照してください。

Mobile Link の [同期モデル作成] ウィザード　この方法は、次の場合に使用します。

- 複数のリモート Ultra Light データベースと集中型の統合データベースがある複雑な同期システムを作成している。
- SQL Anywhere データベースのほかにリファレンス・データベースや統合データベースを使用する必要がある。

  「Mobile Link 統合データベース」『Mobile Link - サーバ管理』と「Mobile Link 同期モデルからの Ultra Light データベースの作成」 56 ページを参照してください。

コマンド・ライン 次のいずれかのユーティリティを使用します。

- データベースの作成時に設定するプロパティに精通し、空の新しい Ultra Light データベースを短時間で作成したい場合は、ulcreate ユーティリティを使用します。このユーティリティは、バッチ処理でデータベースを作成する場合に特に便利です。「コマンド・プロンプトからの Ultra Light データベースの作成」 56 ページを参照してください。
- SQL Anywhere のリファレンス・データベース・スキーマに基づいて、空の新しい Ultra Light データベースを作成する場合は、ulinit ユーティリティを使用します。「SQL Anywhere リファレンス・データベースからの Ultra Light データベースの作成」 57 ページを参照してください。
- 新しい Ultra Light データベースのスキーマかデータまたはその両方のソースとして使用する XML ファイルがある場合は、ulload ユーティリティを使用します。「XML からの Ultra Light データベースの作成」 58 ページを参照してください。

Sybase Central からの Ultra Light データベースの作成

Sybase Central では、Ultra Light の [データベース作成] ウィザードを使用してデータベースを作成できます。

♦ 新しい Ultra Light データベースを作成するには、次の手順に従います (Sybase Central の場合)。

1. [スタート] – [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択して、Sybase Central を起動します。
2. [ツール] – [Ultra Light 10] –[データベースの作成] を選択して、Ultra Light データベースを作成します。

   [データベース作成] ウィザードが表示されます。
3. ウィザードの指示に従います。

参照

- 「コマンド・プロンプトからの Ultra Light データベースの作成」 56 ページ
- 「SQL Anywhere リファレンス・データベースからの Ultra Light データベースの作成」 57 ページ
- 「XML からの Ultra Light データベースの作成」 58 ページ
- 「Ultra Light で使用する作成時のデータベース・プロパティの選択」 61 ページ

- 「Ultra Light のアップグレード」 『SQL Anywhere 10 - 変更点とアップグレード』

コマンド・プロンプトからの Ultra Light データベースの作成

ulcreate ユーティリティを使用して、コマンド・プロンプトからデータベースを作成できます。このユーティリティをオプションとともに使用すると、さまざまなデータベース・プロパティを指定できます。

♦ 新しい Ultra Light データベースを作成するには、次の手順に従います (コマンド・プロンプトの場合)。

1. コマンド・プロンプトを開きます。
2. 必要なパラメータを指定して ulcreate ユーティリティを実行します。

   たとえば、大文字と小文字を区別する *test.udb* という UTF-8 のデータベースを作成し、同じ名前のデータベースが存在する場合に上書きする場合は、次の構文でコマンドを実行します。

   ```
   ulcreate -c "DBF=test.udb" -o "case=respect;utf_encoding=1" -y
   ```

   データベース・ファイルを接続文字列で指定する代わりに、他のコマンド・プロンプト・オプションの後にデータベース・ファイルを指定しても、同じ処理が実行されます。次に例を示します。

   ```
   ulcreate -o "case=respect;utf_encoding=1" -y test.udb
   ```

参照

- 「Sybase Central からの Ultra Light データベースの作成」 55 ページ
- 「SQL Anywhere リファレンス・データベースからの Ultra Light データベースの作成」 57 ページ
- 「XML からの Ultra Light データベースの作成」 58 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light で使用する作成時のデータベース・プロパティの選択」 61 ページ
- 「Ultra Light のアップグレード」 『SQL Anywhere 10 - 変更点とアップグレード』

Mobile Link 同期モデルからの Ultra Light データベースの作成

Mobile Link には、[同期モデル作成] ウィザードで作成したスキーマ定義とテーブル・マップを使用してリモート・クライアントを作成できるモデリング・システムが用意されています。このウィザードによって、同期モデル (リモート・データベースと統合データベース) に関する情報を含むファイルが作成されます。

モデルを作成したら、Sybase Central をモデル・モードで操作し、同期モデルを配備する前にカスタマイズできます。モデルの準備ができたら配備できます。このウィザードによって、Ultra Light リモートに必要なスクリプトやテーブルが生成されます。「Mobile Link のモデルの概要」『Mobile Link - クイック・スタート』を参照してください。

SQL Anywhere リファレンス・データベースからの Ultra Light データベースの作成

リファレンス・データベースは、作成する Ultra Light データベースのテンプレートとして機能する SQL Anywhere データベースです。Ultra Light データベースは、このリファレンス・データベース内のカラム、テーブル、インデックスのサブセットです。これらのオブジェクトは、リファレンス・データベース内のパブリケーションの一部として選択します。

以前のバージョンの Ultra Light では、開発の一部のモードでリファレンス・データベースが必要でした。現在では、SQL Anywhere データベースを統合データベースとして使用して複数のリモート・データベースのデータを収集したり管理しないかぎりは、この方法の開発に対するメリットはありません。ただし、Sybase の PowerDesigner Physical Data Model などのアーキテクチャ・ツールを使用して最初にデータをモデリングする場合はこの方法を使用できます。

Ultra Light データベースは Ultra Light 用の ulinit ユーティリティを使用して初期化できます。このユーティリティをオプションとともに使用すると、さまざまなデータベース・プロパティを指定できます。

♦ リファレンス・データベースから新しい Ultra Light データベースを初期化または抽出するには、次の手順に従います (コマンド・プロンプトの場合)。

1. 新しい SQL Anywhere データベースを作成します。

   dbinit ユーティリティか Sybase Central を使用して、新しいデータベースを作成できます。また、サード・パーティのファイルからデータを移行することで、SQL Anywhere 以外のデータベースから SQL Anywhere データベースを作成することもできます。

   Ultra Light での使用を考慮したデータベース・プロパティの設定
   Ultra Light データベースは、リファレンス・データベースと同じプロパティ設定で生成されます。これらのオプションをリファレンス・データベースで設定することによって、Ultra Light データベースの動作も制御することになります。これらのオプションを以下に示します。

   - 日付フォーマット
   - 日付順
   - 基準年
   - Precision
   - Scale
   - 時間フォーマット
   - タイムスタンプ・フォーマット

2. Ultra Light データベースに必要なオブジェクトを追加してリファレンス・データベースを準備します。

   - **テーブルとキー**　テーブルを追加し、Ultra Light に必要なプライマリ・キーを設定します。必要な場合は、Ultra Light アプリケーション内で必要な外部キーの関係を割り当てることもできます。Sybase Central、Sybase PowerDesigner Physical Data Model、別のデー

タベース設計ツールなど、使いやすい任意のツールを使用できます。「Ultra Light のテーブルとカラムの操作」 90 ページを参照してください。

- **インデックス**
  特に低速のデバイスでは、インデックスによってパフォーマンスが大きく向上します。プライマリ・キーには自動的にインデックス付きになりますが、他のカラムは自動的にインデックス付きにはなりません。「インデックスを使用する場合」 101 ページを参照してください。
- **パブリケーション** テーブルによって同期のタイミングを変えるには、複数の Ultra Light 固有のパブリケーションを使用してテーブルのサブセットを定義して、サブセットごとに同期の優先度を設定できます。「Ultra Light のパブリケーション」 『Mobile Link - クライアント管理』を参照してください。

パフォーマンスに関するヒント
Ultra Light アプリケーションで情報を特定の順序で取り出すことが頻繁にある場合は、リファレンス・データベースにインデックスを追加できます。

3. 必要なオプションを指定して ulinit ユーティリティを実行します。

   たとえば、2 つの異なるパブリケーションに含まれるテーブルを持つ *customer.udb* という Ultra Light データベースを初期化するには、コマンド・プロンプトで次のコマンド・ラインを入力します。Pub1 には、優先的に同期するテーブルのサブセットが含まれ、Pub2 には残りのデータが含まれます。

```
ulinit -a DBF=MySource.db
-c DBF=customer.udb -n Pub1 -n Pub2
```

参照

- 「Sybase Central からの Ultra Light データベースの作成」 55 ページ
- 「コマンド・プロンプトからの Ultra Light データベースの作成」 56 ページ
- 「XML からの Ultra Light データベースの作成」 58 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light で使用する作成時のデータベース・プロパティの選択」 61 ページ
- 「Ultra Light のアップグレード」 『SQL Anywhere 10 - 変更点とアップグレード』

XML からの Ultra Light データベースの作成

Ultra Light データベースを管理する中間のフォーマットとして XML を使用できます。このとき、このフォーマットは、Ultra Light で使用するための条件を満たしている必要があります。XML は次のように使用できます。

- データベースのプロパティやオプションが異なる新しいデータベースにデータをロードする。
- Ultra Lite の旧バージョンで作成されたデータベースからスキーマをアップグレードする。
- Ultra Light データベースのテキスト・バージョンを作成し、バージョン管理システムにチェック・インする。

Ultra Light では任意の XML ファイルは使用できません。 *<install-dir>¥win32* ディレクトリには *usm.xsd* ファイルがあります。このファイルを使用して、XML フォーマットを確認してください。

♦ **XML ファイルから Ultra Light データベースを作成するには、次の手順に従います。**

1. XML ファイルを任意のディレクトリに保存します。次のいずれかの操作を行うことができます。

   - データベースを XML ファイルにエクスポートまたはアンロードします。SQL Anywhere データベースをアンロードする場合は、サポートされている任意のエクスポート方法を使用します。「リレーショナル・データを XML としてエクスポートする」『SQL Anywhere サーバ - SQL の使用法』を参照してください。
   - 別のソースから XML 出力を取得します。ソースには、別のリレーショナル・データベース、またはフラット・ファイルにトランザクションが記録される Web サイトを使用できます。ただし、XML のフォーマットが Ultra Light の条件を満たしている必要があります。

2. コマンド・プロンプトを開きます。
3. 必要なパラメータを指定して ulload ユーティリティを実行します。

   たとえば、*sample.xml* 内のテーブル・フォーマットとデータから、新しい Ultra Light データベースを *sample.udb* ファイルに作成するには、次のように入力します。

   ```
   ulload -c DBF=sample.udb sample.xml
   ```

参照

- 「Sybase Central からの Ultra Light データベースの作成」 55 ページ
- 「コマンド・プロンプトからの Ultra Light データベースの作成」 56 ページ
- 「SQL Anywhere リファレンス・データベースからの Ultra Light データベースの作成」 57 ページ
- 「Ultra Light のアップグレード」『SQL Anywhere 10 - 変更点とアップグレード』
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- 「Ultra Light で使用する作成時のデータベース・プロパティの選択」 61 ページ

デバイス上での Ultra Light の作成

接続時に Ultra Light データベースを検出できなかった場合に新しいデータベースを作成するようにアプリケーションをプログラミングできます。プログラミングすると、アプリケーションは、SQL を使用してテーブル、インデックス、外部キーなどを作成できるようになります。これによって、データベースの移植は、統合データベースと同期させるだけでできるようになります。

考慮事項

この方法では、データベースの作成と SQL のコードがすべて追加されて、アプリケーションのサイズが大きくなる可能性があるので、データベースを作成する他の方法に比べると、あまり望ましくありません。ただし、アプリケーションをデバイスに配備するだけなので配備は簡単にな

ります。また、運用前の開発サイクルで、テストのためにデバイス上のデータベースを削除し、再構築する場合にも使用できます。

参照

- 「デスクトップでの Ultra Light の作成」 54 ページ
- Ultra Light for C/C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light.NET：「CreateDatabase メソッド」 『Ultra Light - .NET プログラミング』
- Ultra Light for AppForge：「CreateDatabase メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』

Ultra Light で使用する作成時のデータベース・プロパティの選択

Ultra Light データベース作成時プロパティは、管理ツールまたは CreateDatabase 関数でデータベースに書き込まれ、*name=value* の組み合わせで記録されます。これらのプロパティはシステム・テーブルに格納されるので、ユーザとアプリケーションが同じようにアクセスできます。「sysuldata システム・テーブル」 247 ページを参照してください。

プロパティへのアクセス

データベース作成後にプロパティを変更することはできません。ただし、Sybase Central でプロパティを表示することはできます。「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページを参照してください。

プロパティは、Ultra Light アプリケーションからプログラム的にアクセスすることもできます。通常は、アプリケーションがこれらのテーブル内のデータに直接アクセスしないようにします。代わりに、アプリケーションが、API に適切な GetDatabaseProperty 関数を呼び出すようにしてください。

API 固有の詳細情報については、次の項を参照してください。

- Ultra Light for C/C++ :「GetDatabaseProperty 関数」 『Ultra Light - C/C++ プログラミング』
- MobileVB :「GetDatabaseProperty メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET :「GetDatabaseProperty メソッド」 『Ultra Light - .NET プログラミング』
- M-Business :「 getDatabaseProperty メソッド」 『Ultra Light - M-Business Anywhere プログラミング』

これらのプロパティに加えて、データベース・オプションまたは接続パラメータを使用して、データベースの他の設定を行うことができます。次の項を参照してください。

- 「Ultra Light でのデータベース・オプションの設定」 73 ページ
- 「Ultra Light データベースへの接続」 77 ページ
- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ

プロパティ・リスト	説明
case	Ultra Light データベースの文字列を比較するときに大文字と小文字を区別するかどうかを設定します。「Ultra Light での大文字と小文字の区別の考慮事項」 65 ページと「Ultra Light case プロパティ」 136 ページを参照してください。
checksum_level	データベースのチェックサム検証のレベルを設定します。「チェックサムを使用したページの整合性の確認」 69 ページと「Ultra Light checksum_level プロパティ」 137 ページを参照してください。
collation	データベースが使用する照合を返します。「Ultra Light での文字の考慮事項」 63 ページと「Ultra Light collation プロパティ」 138 ページを参照してください。

プロパティ・リスト	説明
date_format	日付がデータベースから取り出されるときのデフォルトの文字列フォーマットを設定します。「Ultra Light での日付の考慮事項」 66 ページと「Ultra Light date_format プロパティ」 139 ページを参照してください。
date_order	年、月、日の日付の順序を解釈する方法を制御します。「Ultra Light での日付の考慮事項」 66 ページと「Ultra Light date_order プロパティ」 142 ページを参照してください。
fips	Certicom 認定暗号化アルゴリズムを使用した AES FIPS 準拠データの暗号化を制御します。「Ultra Light でのセキュリティの考慮事項」 70 ページと「Ultra Light fips プロパティ」 143 ページを参照してください。
max_hash_size	Ultra Light のインデックスのハッシュに使用する最大バイト数を設定します。「Ultra Light でのインデックス・パフォーマンスの考慮事項」 64 ページと「Ultra Light max_hash_size プロパティ」 145 ページを参照してください。
nearest_century	文字列から日付への変換で、2 桁の年の解釈を制御します。「Ultra Light での基準年の変換の考慮事項」 67 ページと「Ultra Light nearest_century プロパティ」 146 ページを参照してください。
obfuscate	データベース内のデータを難読化するかどうかを制御します。難読化は単純暗号化です。「Ultra Light でのセキュリティの考慮事項」 70 ページと「Ultra Light obfuscate プロパティ」 147 ページを参照してください。
page_size	データベース・ページ・サイズを定義します。「Ultra Light でのページ・サイズの考慮事項」 68 ページと「Ultra Light page_size プロパティ」 148 ページを参照してください。
precision	10 進法計算での結果の最大桁数を指定します。「Ultra Light での小数点の位置の考慮事項」 68 ページと「Ultra Light precision プロパティ」 150 ページを参照してください。
scale	計算結果が最大 precision にトランケートされる場合の、小数点以下の最小桁数を指定します。「Ultra Light での小数点の位置の考慮事項」 68 ページと「Ultra Light scale プロパティ」 151 ページを参照してください。
time_format	データベースから取り出した時刻のフォーマットを設定します。「Ultra Light での時刻の考慮事項」 66 ページと「Ultra Light time_format プロパティ」 153 ページを参照してください。
timestamp_format	Ultra Light でのタイムスタンプのフォーマットを指定します。「Ultra Light でのタイムスタンプの考慮事項」 66 ページと「Ultra Light timestamp_format プロパティ」 154 ページを参照してください。
timestamp_increment	Ultra Light でのタイムスタンプのトランケート方法を指定します。「Ultra Light でのタイムスタンプの考慮事項」 66 ページと「Ultra Light timestamp_increment プロパティ」 156 ページを参照してください。

プロパティ・リスト	説明
utf8_encoding	ユニコード用の 8 ビット・マルチバイト・エンコードである UTF-8 フォーマットでデータをエンコードします。「Ultra Light での文字の考慮事項」 63 ページと「Ultra Light utf8_encoding プロパティ」 158 ページを参照してください。

Ultra Light での文字の考慮事項

文字列の比較結果とソート順は、データベースの文字セット、照合、エンコードのプロパティによって異なります。

Ultra Light によってデータベース・ファイルが書き込まれた後でデータベースの照合を変更することはできません。既存のデータベースが使用している照合を変更するには、データベースをアンロードし、適切な照合を使用して新しいデータベースを作成してから、データベースを再ロードする必要があります。新しい照合に合わせてデータを変換する必要がある場合があります。

データベースに適切な文字セット、照合、エンコードのプロパティを決定するには、主に次の点を考慮します。

- 必要なソート順。一般に、データベースに格納する文字を適切にソートできる照合を選択してください。
- デバイスのプラットフォーム。サポートされているデバイスによって要件が異なり、一部のデバイスでは、UTF-8 で文字をエンコードする必要があります。複数のデバイスをサポートする必要がある場合は、データベースを共有できるかどうかを確認する必要があります。
- データを同期する場合は、統合データベースでサポートされている言語と文字セット。Ultra Light データベースと統合データベースで使用する文字セットに互換性があることを確認します。互換性がなく、一方のデータベースの文字セットにある文字が他方の文字セットになかった場合、データが失われたり、予期せず変更される可能性があります。Ultra Light を多言語環境に配備する場合も、Ultra Light データベースを UTF-8 でエンコードしてください。

  同期のときに、Mobile Link サーバで次のように文字が変換されます。

  1. Ultra Light データベースの文字がユニコードに変換されます。
  2. ユニコード文字が統合データベースの文字セットに変換されます。

参照

- 「Ultra Light での文字セットのエンコードに関するプラットフォーム要件」 64 ページ
- 「Ultra Light collation プロパティ」 138 ページ
- 「Ultra Light utf8_encoding プロパティ」 158 ページ
- 「文字セット、エンコード、照合の概要」『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「文字セットの考慮事項」『Mobile Link - サーバ管理』
- 「Ultra Light での大文字と小文字の区別の考慮事項」 65 ページ
- 「Ultra Light でのセキュリティの考慮事項」 70 ページ

Ultra Light での文字セットのエンコードに関するプラットフォーム要件

各プラットフォームには特定の文字セットとエンコードの要件があります。

Palm OS

UTF-8 エンコードは使用しないでください。Palm ではユニコード文字はサポートされません。常に対象デバイスのコード・ページと一致する照合を選択してください。

Symbian OS

常に UTF-8 エンコードを使用してください。これがデバイスの標準の文字セットです。Symbian では MBCS 文字セットはサポートされません。

Windows デスクトップと Windows CE

UTF-8 でエンコードされたデータベースを Windows で使用するときは、ワイド文字をデータベースに渡してください。これらのプラットフォームで UTF-8 エンコードを使用している場合、Ultra Light ではワイド文字以外の文字列パラメータが UTF-8 でエンコードされていると見なされますが、これは Windows で標準の文字セットではありません。接続文字列は例外であり、文字列パラメータはアクティブなコード・ページであると見なされます。ただし、ワイド文字を使用することで、このような複雑な状況を避けることができます。

参照

- 「Ultra Light utf8_encoding プロパティ」 158 ページ
- 「文字セット、エンコード、照合の概要」 『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「文字セットの考慮事項」 『Mobile Link - サーバ管理』
- 「Ultra Light での大文字と小文字の区別の考慮事項」 65 ページ
- 「Ultra Light でのセキュリティの考慮事項」 70 ページ

Ultra Light でのインデックス・パフォーマンスの考慮事項

ハッシュは、インデックス・エントリのオプション部分で、インデックスのページに格納されます。ハッシュによって、インデックス・カラムの実際のローの値が、インデックスの順序を保持したまま、それに相当する数値 (キー) に変換されます。キーのサイズと、実際の値がハッシュされる長さは、設定するハッシュのサイズで決まります。

Ultra Light データベースでは、自動的にデフォルトの最大ハッシュ・サイズ 4 バイトが使用されます。このデフォルトは、別のサイズに変更するか、0 に変更してインデックスのハッシュを無効にできます。このデータベースのデフォルトのハッシュ・サイズは、新しいインデックスを作成するときに変更できます。

ハッシュによるパフォーマンスの向上

Ultra Light では、ロー ID によって、テーブル内の実際のデータのローを見つけることができます。ロー ID は、常にインデックス・エントリの一部です。ハッシュ・サイズを 0 に設定する

(インデックスのハッシュを無効にする) と、インデックス・エントリにはこのロー ID だけが含まれます。ハッシュ・サイズを 0 以外に設定すると、そのローの変換されたデータすべてまたはその一部を含むハッシュ・キーがロー ID とともにインデックスのページに格納されます。その結果、Ultra Light で必ずしもデータを検索、ロード、アンパックしてから実際のローの値を比較する必要がないため、これらのインデックス・カラムに対するクエリのパフォーマンスを向上できます。

データベースのデフォルトのハッシュ・サイズを決定するには、クエリの効率とデータベースのサイズのトレードオフを評価する必要があります。最大ハッシュ値が大きいほど、データベースのサイズが大きくなります。

参照

- 「Ultra Light クエリ・パフォーマンスの最適化 」 38 ページ
- 「Ultra Light のインデックスの操作」 100 ページ
- 「Ultra Light でのインデックス・パフォーマンスの考慮事項」 64 ページ
- 「Ultra Light max_hash_size プロパティ」 145 ページ
- 「Ultra Light でのページ・サイズの考慮事項」 68 ページ

Ultra Light での大文字と小文字の区別の考慮事項

文字列の比較結果とソート順は、データベースで大文字と小文字を区別するかどうかによって異なります。Ultra Light データベースでの大文字と小文字の区別は、次の影響を及ぼします。

- **データ**　データの大文字と小文字を区別するかしないかは、インデックスや文字列の比較などに反映されます。デフォルトでは、比較時には常に大文字と小文字が区別されません。たとえば、Ultra Light では **short sleeve t** と **Short Sleeve T** は同じであると見なされます。
- **識別子**　識別子には、テーブル名やカラム名などがあります。ユーザ ID を含む識別子は、データベースの大文字と小文字の区別に関係なく、常に大文字と小文字が区別されません。たとえば、Ultra Light では **UID=DBA** と **UID=dba** は常に同じであると見なされます。
- **パスワード**　Ultra Light データベースのパスワードは常に大文字と小文字が区別されます。たとえば、Ultra Light では **PWD=sql** と **PWD=sQl** は同じであると見なされません。

ただし、一部の照合では、識別子の大文字と小文字の区別を前提とする場合、特別な注意が必要です。特に、トルコ語の照合では、予期できない複雑なエラーが発生するような大文字と小文字の変換動作があります。最も一般的なエラーは、I または i という文字を含むシステム・オブジェクトが見つからないというものです。

参照

- 「Ultra Light での文字の考慮事項」 63 ページ
- 「Ultra Light case プロパティ」 136 ページ

Ultra Light での日付の考慮事項

データ型が DATE の値は、date_format プロパティで設定されているフォーマットで表されます。日付の値は、文字列で表すこともできます。日付の値は、文字列変数に割り当ててから取り出します。

使用できる日付の単位

Ultra Light では**日付の単位**から日付が構築されます。日付の単位には、年、月、日、曜日、通し日数、時、分、秒 (その要素) があります。

これらの日付の単位を特定の順序に並べるには、date_order プロパティを使用します。

デフォルトの日付のフォーマットと順序は ISO (YYYY-MM-DD) です。たとえば、2006 年 1 月 7 日はこのフォーマットでは 2006-01-07 になります。デフォルトの ISO の日付のフォーマットと順序を使用しない場合は、これらの日付の単位に別のフォーマットと順序を指定します。

参照

- 「Ultra Light date_order プロパティ」 142 ページ
- 「Ultra Light date_format プロパティ」 139 ページ
- 「Ultra Light でのタイムスタンプの考慮事項」 66 ページ

Ultra Light での時刻の考慮事項

Ultra Light では time_format プロパティで設定した時間の単位を使用して時間が記述されます。時間の単位には、時、分、秒、ミリ秒があります。

時間の値は、文字列で表すこともできます。時間の値は、文字列変数に割り当ててから取り出します。

デフォルトの ISO フォーマット (HH:MM:SS) を使用しない場合は、これらの時間の単位のフォーマットを指定します。

ISO (HH:MM:SS) がデフォルトの時間フォーマットです。たとえば、真夜中はこのフォーマットでは 00:00:00 になります。デフォルトの ISO の時間フォーマットを使用しない場合は、これらの時間の単位に別のフォーマットと順序を指定します。

参照

- 「Ultra Light time_format プロパティ」 153 ページ
- 「Ultra Light でのタイムスタンプの考慮事項」 66 ページ

Ultra Light でのタイムスタンプの考慮事項

Ultra Light では、date_format プロパティと time_format プロパティで設定した日付と時間の単位からタイムスタンプが作成されます。日付と時間には合計 7 つの単位があります (年、月、日、時、分、秒、ミリ秒)。

タイムスタンプの値は、文字列で表すこともできます。タイムスタンプの値は、文字列変数に割り当ててから取り出します。

一般に、タイムスタンプ・カラムによって、統合データベースと同期させるときにデータの整合性が維持されます。タイムスタンプを使用して、各ユーザが最後に同期した時刻を追跡することで、複数のリモート・データベース間でデータの同時更新がいつ発生したかを特定できます。

ヒント
タイムスタンプとその増分は、統合データベースと Ultra Light データベースで同じ値になるようにしてください。これらのプロパティを統合データベースのものと合うように設定すると、タイムスタンプの不一致を回避できます。

参照

- 「Ultra Light での日付の考慮事項」 66 ページ
- 「Ultra Light での時刻の考慮事項」 66 ページ
- 「Ultra Light timestamp_increment プロパティ」 156 ページ
- 「Ultra Light timestamp_format プロパティ」 154 ページ
- 「タイムスタンプベースのダウンロード」 『Mobile Link - サーバ管理』
- 「Ultra Light での同時実行性」 13 ページ

Ultra Light での基準年の変換の考慮事項

Ultra Light では、期待値が日付の値であれば、年が文字列内で 2 桁だけで表されている場合でも、文字列を日付に自動的に変換します。年が 2 桁の場合は、適切なロールオーバ値を設定する必要があります。

適切なロールオーバ値を決定するには、次の点を考慮します。

- **2 桁の年の使用**　年が 2 桁ではない場合は、基準年への変換は適用されません。設定した nearest_century 値よりも小さい 2 桁の年は 20*yy* に変換され、nearest_century 値以上の年は 19*yy* に変換されます。

  ただし、間違った変換の問題を防ぐには、常に 4 桁の年を格納し、日付を明確にしてください。「あいまいさのない日付と時刻の使用」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

- **統合データベースの互換性**　たとえば、従来の SQL Anywhere では、年に 1900 を加算していました。Adaptive Server Enterprise では基準年を使用するので、*yy* が 50 より小さい場合は 20*yy* になります。

- **日付の意味：過去または未来**　誕生年は、過去のことなので、小さいロールオーバ値が必要です。したがって、*yy* が 20 未満の年は 20*yy* に設定してください。これに対して、日付が有効期限の場合は、未来のことなので、大きい値の方が理にかなっています。

このオプションを設定しないと、デフォルト設定の 50 が使用されます。したがって、2 桁の年文字列は 1950 ー 2049 として認識されます。

参照

- 「Ultra Light nearest_century プロパティ」 146 ページ
- 「あいまいな文字列から日付への変換」 『SQL Anywhere サーバ - SQL リファレンス』

Ultra Light での小数点の位置の考慮事項

小数点の位置は数値の精度 (precision) と位取り (scale) で決まります。精度は小数点の左右の合計桁数です。位取りは、計算結果が最大精度にトランケートされるときの小数点以下の最小桁数です。

適切な小数点の位置を決定するには、次の点を考慮します。

- **実行する計算のタイプ**　掛け算、割り算、足し算、引き算、集合関数はすべて結果が最大精度を超えることができます。

  たとえば、DECIMAL(8,2) と DECIMAL(9,2) を掛けると、結果には DECIMAL(17,4) が必要です。precision が 15 の場合、15 桁のみが結果に保持されます。scale が 4 の場合、結果は DECIMAL(15,4) になります。scale が 2 の場合、結果は DECIMAL(15.2) になります。どちらの場合も、オーバフロー・エラーの可能性があります。

- **scale と precision の値の関係**　scale は、小数点以下の桁数を設定し、負の数や precision より大きい値にはできません。

参照

- 「Ultra Light precision プロパティ」 150 ページ
- 「Ultra Light scale プロパティ」 151 ページ

Ultra Light でのページ・サイズの考慮事項

データベース内での格納の単位をページといい、データベースの入出力操作は常にページ単位で行われます。Ultra Light では次の情報を保持するようにページが割り付けられます。

- テーブルのロー (ユーザ・テーブルまたはシステム・テーブル)
- インデックス情報
- Mobile Link サーバの同期情報

データベースを暗号化する場合、データはページ・レベルで暗号化されます。

最適なページ・サイズの選択

Ultra Light データベースはページ単位で保管され、I/O 操作はすべてページ単位で行われます。ほとんどのアプリケーションでは、Ultra Light のデフォルトのページ・サイズ 4 KB が適切です。

ただし、データベースの配備に必要な場合は、別のサイズを選択できます。このとき、選択したページ・サイズは、データベースのパフォーマンスやサイズに影響する可能性があります。一般的に、大きいデータベースには、大きなページ・サイズが適しています。ページが小さいと保持

できる情報量が少なく、特にページの半分以上のサイズのローを挿入する場合は、領域の使用効率が低くなります。

- **ローの数** ロー (BLOB を除く) はページに収まる必要があるので、ページ・サイズによって、パックされたローの最大サイズと、各ページに格納できるローの数が決まります。「ローのパックとテーブル定義」 90 ページを参照してください。

  通常、ページ・サイズが小さいと、ランダムな場所から比較的少ない数のローを取り出す操作に有利な傾向があります。これに対して、ページ・サイズが大きいと、テーブルの逐次スキャンを実行するクエリに有利です。この場合、メモリに 1 ページを読み込んで 1 つのローの値を取得すると、次のいくつかのローの内容をメモリにロードできるという 2 次的な効果があります。

- **キャッシュ・サイズ** ページ・サイズが大きいと、同じ領域に収まるページ数が少ないので、大きいキャッシュ・サイズが必要になります。8 KB のような大きなページ・サイズを選択する場合は、データベースに接続するときにキャッシュのサイズを増やします。

  たとえば、1 MB のメモリには、1 ページを 1 KB とすると 1000 ページ格納できますが、1 ページ 4 KB であれば 250 ページしか格納できません。格納できるページ数は、使用しているデータベースと、アプリケーションが実行するクエリの性質によってまったく異なります。さまざまなキャッシュ・サイズで、パフォーマンス・テストを実行できます。キャッシュに十分なページを格納できない場合、Ultra Light は頻繁に使用されるページをディスクにスワップし始めるため、パフォーマンスが低下します。「Ultra Light CACHE_SIZE 接続パラメータ」 170 ページを参照してください。

- **インデックス・エントリ** ページ・サイズは、インデックスにも影響を与えます。データベースのページが大きいほど、保持できるインデックス・エントリ数が多くなります。「Ultra Light のインデックスの操作」 100 ページを参照してください。

ただし、小さいページ・サイズが必要な場合もあります。たとえば、ページ・サイズが小さいと、同じサイズのキャッシュにより多くのページを格納できるため、Ultra Light を少ないリソースで実行できます。特に、メモリが限られている小型のデバイスでデータベースを実行する場合に便利です。また、不特定のロケーションから少量の情報を取得するためにデータベースを使う場合にも便利です。

チェックサムを使用したページの整合性の確認

checksum_level データベース・プロパティを設定することで、ディスク、フラッシュ、またはメモリに保管されているページの妥当性を検査できます。選択するレベルによって、Ultra Light によって各データベース・ページの**チェックサム**が計算、記録されてから、ページが記憶領域に書き込まれます。「Ultra Light checksum_level プロパティ」 137 ページを参照してください。

計算されたチェックサムが、記憶領域から読み取られたページのチェックサムと一致しない場合は、ページの保管または取り出し中にページが変更されたか、壊れた可能性があります。チェックサムが一致しなかった場合、Ultra Light によってデータベースが停止され、致命的なエラーがレポートされます。このエラーは解決できません。Ultra Light データベースを再作成し、データベースのエラーを iAnywhere にレポートしてください。

参照

- 「Ultra Light のプラットフォーム別の最適化方法」 49 ページ
- 「Ultra Light クエリ・パフォーマンスの最適化 」 38 ページ
- 「Ultra Light page_size プロパティ」 148 ページ
- 「Ultra Light データベースへの接続」 77 ページ

Ultra Light でのセキュリティの考慮事項

デフォルトでは、Ultra Light データベースはディスク上で暗号化されません。データベース内のテキスト・カラムとバイナリ・カラムは、16 進エディタなどの表示ツールを使用すると、きちんと読むことができます。セキュリティ強化のためにデータを暗号化する必要がある場合は、次のオプションを検討します。

- **難読化** このオプションは、データベース内のデータに対する何気ないアクセスからデータを保護します。このオプションには、強力な暗号化ほどのセキュリティはありません。難読化は、パフォーマンスにほとんど影響しません。難読化は obfuscate プロパティで設定します。エンド・ユーザが対応する接続パラメータを指定する必要はありません。デバイスで使用するのが単純な難読化の場合、特別な設定は不要です。
- **AES 128 ビットの強力な暗号化** Ultra Light データベースは、AES 128 ビット・アルゴリズムを使用して強力に暗号化できます。このアルゴリズムは、SQL Anywhere データベースを暗号化するために使用しているアルゴリズムと同じです。強力な暗号化を使用すると、巧妙で容赦ないデータへのアクセス試行に対抗するセキュリティが提供されますが、パフォーマンスに大きな影響を与えます。暗号化を設定するには、ウィザードで [Encrypt Database] オプションを選択してから [AES Strong Encryption] を選択します。作成ユーティリティを使用して、key 接続パラメータでキーを設定します。データベースの作成後にエンド・ユーザがデータベースに接続するときには同じパラメータを使用します。デバイスで使用するのが AES 暗号化の場合、特別な設定は不要です。
- **AES FIPS 140-2 準拠の暗号化** Ultra Light には、米国政府とカナダ政府の FIPS 140-2 規格に準拠した暗号化ライブラリが用意されています (Certicom 認定の暗号化モジュールを使用)。このオプションは、この強度の暗号化が必要な政府機関だけが選択してください。FIPS 準拠の暗号化は FIPS プロパティで設定します。このプロパティを設定すると、エンド・ユーザが対応する接続パラメータで必要なキーを指定する必要があります。AES FIPS 暗号化を使用するには、デバイスを適切に設定する必要があります。「Ultra Light での AES FIPS データベース暗号化の設定 」 71 ページを参照してください。

ヒント
Mobile Link サーバの同期ストリームでは、パブリック・キーまたはプライベート・キーを使用してストリーム・データを暗号化できます。配備を容易にするには、これらの証明書を Ultra Light データベースの作成時に埋め込むことができます。「トランスポート・レイヤ・セキュリティを使用する Mobile Link クライアントの設定」『SQL Anywhere サーバ - データベース管理』を参照してください。

注意
FIPS と AES の両方のデータベース暗号化で 128 ビット AES が使用されます。したがって、同じ暗号化キーを使用すると、選択する規格に関係なく、データベースは同じ方法で暗号化されます。

警告
暗号化キーはデータベースの作成後に変更できますが、変更する場合は細心の注意が必要です。次の項を参照してください。

- Ultra Light for C++ :「ChangeEncryptionKey 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「ChangeEncryptionKey メソッド」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET :「ChangeEncryptionKey メソッド」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere :「ChangeEncryptionKey メソッド」『Ultra Light - M-Business Anywhere プログラミング』

この操作は負荷が高く、元に戻せません。操作が途中で停止すると、データベースが完全に失われます。
また、強力に暗号化されたデータベースの暗号化キーを忘れると、テクニカル・サポートに依頼してもデータにアクセスできなくなります。アクセスできなくなったデータベースは、廃棄して、新しくデータベースを作成する必要があります。

参照

- 「Ultra Light fips プロパティ」 143 ページ
- 「Ultra Light obfuscate プロパティ」 147 ページ
- 「Ultra Light DBKEY 接続パラメータ」 188 ページ
- 「Ultra Light での TLS が有効化された同期の設定」『Mobile Link - クライアント管理』
- Ultra Light for AppForge :「暗号化と難読化」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET :「暗号化と難読化」『Ultra Light - .NET プログラミング』
- Ultra Light for C++ :「データの暗号化」『Ultra Light - C/C++ プログラミング』
- Ultra Light for M-Business Anywhere :「データベースの暗号化と難読化」『Ultra Light - M-Business Anywhere プログラミング』

Ultra Light での AES FIPS データベース暗号化の設定

データベースの暗号化に AES FIPS 暗号化を使用した場合、デバイスをプラットフォーム別に設定して配備する必要があります。

♦ アプリケーションとデバイスを AES FIPS 暗号化された Ultra Light データベース用に設定するには、次の手順に従います。

1. Ultra Light データベースをプロパティ **fips=1** を使用して作成します。「Ultra Light fips プロパティ」 143 ページを参照してください。

2. アプリケーションの接続文字列で、接続パラメータとして **DBKEY=*key*** を使用します。「Ultra Light DBKEY 接続パラメータ」 188 ページを参照してください。
3. Palm OS では、ULEnableRsaFipsStrongEncryption を呼び出して、データベースの暗号化を有効にします。「ULEnableFIPSStrongEncryption 関数」 『Ultra Light - C/C++ プログラミング』を参照してください。
4. Palm OS では、*ulrt.lib* のほかに、次のファイルへリンクしていることを確認します。
   - *ulfips.lib*
   - *gse1st.lib*
5. 適切なファイルをデバイスに配備していることを確認します。
   - Windows デスクトップと Windows CE では、*ulfips10.dll* と *sbgse2.dll* が必要です。Windows CE コンポーネントには、コンポーネント DLL ファイルも必要です。
   - Palm OS では *libsbgse_4i.prc* が必要です。

参照

- 「Ultra Light での TLS が有効化された同期の設定」 『Mobile Link - クライアント管理』
- Ultra Light for AppForge：「暗号化と難読化」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「暗号化と難読化」 『Ultra Light - .NET プログラミング』
- Ultra Light for C++：「データの暗号化」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for M-Business Anywhere：「データベースの暗号化と難読化」 『Ultra Light - M-Business Anywhere プログラミング』

Ultra Light でのデータベース・オプションの設定

いつでも設定または変更できるデータベース・プロパティの設定にはオプションが使用されます。Ultra Light では、データベース・オプションとプロパティを同じ名前にすることでこれを実現しています。Ultra Light では、オプションは永続的または一時的です。一時的なオプションは、データベースの実行中のみ保持されます。永続的なオプションはデータベースの sysuldata システム・テーブルに格納されます。

設定可能なオプションは、それに対応するデータベース・プロパティを変更することで、いつでも表示したりアクセスしたりできます。

参照

- 「sysuldata システム・テーブル」 247 ページ
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ
- 「Ultra Light の設定可能なプロパティ」 136 ページ
- 「Ultra Light の読み込み専用プロパティ」 165 ページ

Ultra Light でのコミット・フラッシュの考慮事項

commit_flush_count と **commit_flush_timeout** は、どちらも一時的なデータベース・オプションです。つまり、これらのオプションはデータベースを起動するたびに設定する必要があります。オプションは、データベースが実行され続けている間は保持されます。これらは、**COMMIT_FLUSH=grouped** が接続文字列の一部として設定される場合にのみ必要です。

コミット・フラッシュ・オプションを設定するうえで重要な考慮事項は、コミットされたトランザクションがフラッシュされるまでの遅延が、データのリカバリ性に与えるリスクの大きさです。トランザクションがコミットされた後であっても、トランザクションが失われる可能性がわずかながら存在します。コミット後に深刻なハードウェア障害またはクラッシュが発生した場合、それがトランザクションが記憶領域にフラッシュされる前ならば、トランザクションはリカバリ時にロールバックされます。

一方、遅延を長くするほど、Ultra Light のパフォーマンスは向上します。したがって、適切なカウントまたはタイムアウトのスレッショルドの選択は慎重に行ってください。

参照

- 「単一のトランザクションまたはグループ化されたトランザクションのフラッシュ」 47 ページ
- 「Ultra Light commit_flush_count オプション [テンポラリ]」 160 ページ
- 「Ultra Light commit_flush_timeout オプション [テンポラリ]」 161 ページ

Ultra Light でのリモート ID の考慮事項

「**リモート ID**」は、Ultra Light リモートで使用されるユニークな識別子で、Mobile Link による同期に使用されます。Mobile Link のリモート ID は、最初は NULL に設定され、データベースの最

初の同期時 (またはリモート ID を再び NULL にリセットした場合) に Mobile Link サーバによって GUID に設定されます。ただし、リモート ID は、すべての Mobile Link クライアント間でユニークであれば、意味のある任意の文字列にできます。一意性の要件は常に適用されます。

リモート ID を使用して Mobile Link ユーザ名の同期の進行状況を格納します。ユニークなリモート ID を含めると、ユーザ名はユニークである必要がなくなります。ユーザ名は、認証のためにクライアントを識別する実際のユーザ名にできます。

リモート ID は、複数の Mobile Link ユーザが同じ Ultra Light クライアント・データベースを同期する場合に特に便利です。この場合、同期スクリプトでは、ユーザ名だけでなく、リモート ID を参照してください。

♦ **Sybase Central でリモート ID を設定またはリセットするには、次の手順に従います。**

1. [フォルダ] ビューで接続先の Ultra Light データベースを右クリックし、ポップアップ・メニューから [オプション] を選択します。

   [データベースのオプション] ダイアログが表示されます。

2. テーブルの ml_remote_id エントリを選択します。
3. [値] フィールドに ID の新しい値を入力します。
4. [すぐに設定] をクリックして変更を保存します。
5. [閉じる] をクリックします。

参照

- 「Ultra Light クライアントの概要」 『Mobile Link - クライアント管理』
- 「リモート ID」 『Mobile Link - クライアント管理』
- 「Ultra Light ユーザ認証」 『Mobile Link - クライアント管理』
- 「User Name 同期パラメータ」 『Mobile Link - クライアント管理』 および 「Password 同期パラメータ」 『Mobile Link - クライアント管理』
- 「Ultra Light ml_remote_id オプション」 163 ページ

Ultra Light でのグローバル・データベース ID の考慮事項

グローバル ID は、Mobile Link サーバとの同期時にプライマリ・キーの一意性を維持するために GLOBAL AUTOINCREMENT カラムの開始値を設定します。テーブルにローが追加され、まだ値が設定されていない場合は、Ultra Light によって、global_database_id の値と分割サイズを組み合わせてカラムの値が生成されます。「グローバル・オートインクリメントの使用」 『Mobile Link - サーバ管理』を参照してください。

♦ **Sybase Central でデータベースのグローバル ID を設定またはリセットするには、次の手順に従います。**

1. [フォルダ] ビューで接続先の Ultra Light データベースを右クリックし、ポップアップ・メニューから [オプション] を選択します。

[データベースのオプション] ダイアログが表示されます。

2. テーブルの global_database_id エントリを選択します。
3. [値] フィールドに ID の新しい値を入力します。
4. [すぐに設定] をクリックして変更を保存します。
5. [閉じる] をクリックします。

参照

- 「Ultra Light global_database_id オプション」 162 ページ

第 5 章

Ultra Light データベースへの接続

目次

Ultra Light データベース接続の概要

データベースを使用するアプリケーションでは、データベースへの接続を確立してから、トランザクションが行われます。アプリケーションには、Ultra Light のコマンド・ライン・ユーティリティ、Sybase Central ツールまたは Interactive SQL の接続ダイアログ、またはカスタム・アプリケーションがあります。

Ultra Light データベースに接続することで、アプリケーションのすべてのアクティビティが行われるチャネルが作成されます。接続が試行されるたびに、データベースに固有の SQL トランザクションが作成されます。

Ultra Light データベースの接続パラメータのリスト

設定する接続パラメータの一部は、データベースの作成時に定義するプロパティと重複します。一般に、作成時には必要なプロパティを定義します。接続時には、作成時に設定した値を指定します。これらの共有設定の違いと共通点を理解し、いつ、何を指定すればいいかがわかっていることが重要です。

パラメータ名	説明
CACHE_SIZE	データベース・キャッシュのサイズを定義します。「Ultra Light CACHE_SIZE 接続パラメータ」 170 ページを参照してください。
CON	現在の接続の名前を指定します。「Ultra Light CON 接続パラメータ」 174 ページを参照してください。
DBF と、CE_FILE、PALM_FILE、NT_FILE、または SYMBIAN_FILE	データベースの作成時には、これらのパラメータはデータベースのロケーションを設定します。その後の接続では、ファイルの場所を指定します。 単一プラットフォームのアプリケーションを作成している場合や、Ultra Light 管理ツールに接続している場合は、DBF を使用できます。さまざまなプラットフォーム固有データベースに接続する Ultra Light クライアントをプログラミングしている場合は、その他のプラットフォーム固有バージョンを使用します。次の項を参照してください。 ◆ 「Ultra Light DBF 接続パラメータ」 175 ページ ◆ 「Ultra Light CE_FILE 接続パラメータ」 177 ページ ◆ 「Ultra Light PALM_FILE 接続パラメータ」 181 ページ ◆ 「Ultra Light NT_FILE 接続パラメータ」 179 ページ ◆ 「Ultra Light SYMBIAN_FILE 接続パラメータ」 184 ページ
PALM_DB	AppForge 開発のみが対象です。AppForge の作成者 ID ではなく、Ultra Light の作成者 ID が使用されるように、正しい作成者 ID を設定します。このパラメータは、PALM_FILE パラメータと組み合わせて使用します。「Ultra Light PALM_DB 接続パラメータ」 183 ページを参照してください。

パラメータ名	説明
DBN	ファイル名ではなく名前で実行中のデータベースを指定します。「Ultra Light DBN 接続パラメータ」 186 ページを参照してください。
DBKEY	データベースの作成時には、このパラメータは使用する暗号化キーを設定します。その後の接続では、データベースの作成時に設定した暗号化キーを指定し、渡します。指定したキーが間違っていた場合は、接続に失敗します。「Ultra Light DBKEY 接続パラメータ」 188 ページを参照してください。
PALM_ALLOW_BACKUP	Palm デバイス上の HotSync のバックアップ動作を制御します。「Ultra Light PALM_ALLOW_BACKUP 接続パラメータ」 191 ページを参照してください。
PWD	データベースの作成時には、ユーザの*初期*パスワードを設定します。その後の接続では、ユーザ ID のパスワードを指定します。「Ultra Light PWD 接続パラメータ」 192 ページを参照してください。
RESERVE_SIZE	実際にデータを挿入することなく、Ultra Light データベースが必要とするファイル・システム領域を事前に割り付けます。「Ultra Light RESERVE_SIZE 接続パラメータ」 194 ページを参照してください。
START	Ultra Light エンジンの実行プログラムのロケーションを指定し、起動します。「Ultra Light START 接続パラメータ」 196 ページを参照してください。
UID	データベースの作成時には、*初期*ユーザ ID を設定します。その後の接続では、データベースにユーザを指定します。ユーザ ID は、Ultra Light データベースに格納されている最大 4 つのユーザ ID のうちの 1 つである必要があります。「Ultra Light UID 接続パラメータ」 197 ページを参照してください。

参照

- 「ユーザ ID とパスワードの組み合わせの解釈」 80 ページ
- 「Ultra Light page_size プロパティ」 148 ページ

ユーザ認証の役割

Ultra Light では、認証を無効にできません。正常に接続するには、ユーザが認証される必要があります。SQL Anywhere とは異なり、Ultra Light データベースでは、オブジェクトの所有権のためではなく、認証のためだけにユーザを作成し、管理します。ユーザが認証され、データベースに接続すると、ユーザはスキーマ・データを含むデータベース内のすべてのデータに無制限にアクセスできます。

Ultra Light のユーザは既存の接続からのみ追加または変更できます。したがって、Ultra Light のユーザベースを変更するには、その前に有効な UID と PWD で接続する必要があります。

初めて接続するときに必要な UID と PWD は、最初にデータベースを作成したときに設定した値です。初期ユーザを設定しなかった場合は、デフォルトの **UID=DBA** と **PWD=sql** を指定します。

認証の回避

認証を無効にすることはできませんが、データベースの作成時とデータベースへの接続時に Ultra Light のデフォルトを使用することで認証を回避できます。

UID と PWD の*両方*のパラメータを指定しなかった場合、Ultra Light では、接続方法に関係なく、常にデフォルトの **UID=DBA** と **PWD=sql** が想定されます。

♦ **Ultra Light で認証を回避するには、次の手順に従います。**

1. データベースの作成時に UID パラメータと PWD パラメータを設定しません。
2. Ultra Light データベース内のデフォルト・ユーザを削除または変更しません。
3. 作成したデータベースへの接続時に UID パラメータと PWD パラメータを設定しません。

参照

- 「考慮事項と制限事項」 110 ページ
- 「Ultra Light のユーザの操作」 110 ページ
- 「ユーザ ID とパスワードの組み合わせの解釈」 80 ページ

ユーザ ID とパスワードの組み合わせの解釈

Ultra Light では、UID パラメータと PWD パラメータのいずれか一方だけ設定するか、どちらも設定しないか、両方とも設定することができます。ただし、部分的な定義が原因でユーザが Ultra Light で識別できない場合があります。次の表は、不完全なユーザ定義が Ultra Light で解釈される方法を示します。

データベース作成時の設定	結果
ユーザ ID とパスワードの*両方*を設定しない	Ultra Light によって UID が **DBA** で PWD が **sql** のデフォルト・ユーザが作成されます。その後の接続時にこれらのパラメータを指定する必要はありません。
ユーザ ID パラメータのみ設定 例 ♦ UID=JaneD ♦ UID=JaneD;PWD= ♦ UID=JaneD;PWD=""	Ultra Light によって UID が **JaneD** で PWD が空のデフォルト・ユーザが作成されます。接続時には、常に UID パラメータを指定します。PWD パラメータは不要です。

データベース作成時の設定	結果
パスワード・パラメータのみ設定 例 ◆ PWD=3saBys ◆ UID=;PWD=3saBys ◆ UID="";PWD=3saBys	Ultra Light でエラーが発生します。Ultra Light では、ユーザ ID なしでパスワードを設定することはできません。

参照

- 「ユーザ認証の役割」 79 ページ
- 「考慮事項と制限事項」 110 ページ
- 「Ultra Light のユーザの操作」 110 ページ

接続文字列を使用した Ultra Light 接続の確立

接続文字列は、接続を定義し、最終的に確立できるようにアプリケーションからランタイムに渡す一連のパラメータです。

データベースへの接続の確立は、次の 3 つの手順で行われます。

1. **パラメータの定義**　サポートされているパラメータの任意の組み合わせで接続を定義します。一部の接続パラメータは、接続を確立するために必須です。その他のパラメータは、単一の接続のデータベース機能を調整するために使用します。

   パラメータを指定する方法は、Ultra Light 管理ツールから接続するか、Ultra Light アプリケーションから接続するかによって異なります。詳細については、「Ultra Light 接続パラメータの指定」 82 ページを参照してください。

2. **文字列のアセンブル**　開発者かまたはアプリケーションで、指定するパラメータを文字列にアセンブルします。接続文字列は、*keyword=value* をセミコロンで区切ったパラメータのリストです。詳細については、「パラメータの Ultra Light 接続文字列へのアセンブル」 84 ページを参照してください。

   たとえば、ファイル名、ユーザ ID、パスワードを指定する接続文字列フラグメントは次のようになります。

   ```
   DBF=myULdb.udb;UID=JDoe;PWD=token
   ```

3. **送信**　接続文字列をアセンブルしたら、Ultra Light ランタイムへの Ultra Light API を使用して、処理のためにデータベースに渡します。接続の試行が検証されると、接続が許可されます。接続は、次の場合に失敗します。

   - 指定したデータベース・ファイルが存在しない
   - 認証に失敗した

Ultra Light 接続パラメータの指定

接続情報の収集方法は、入力を体系化または自動化する程度によって異なります。入力を体系化するほど、接続情報の信頼性が高くなります。

接続の詳細は、Ultra Light のカスタム・アプリケーションから接続するか、SQL Anywhere の Ultra Light 用管理ツール (Interactive SQL、Ultra Light のコマンド・ライン・ユーティリティ、Sybase Central の Ultra Light のウィザード) から接続するかによって、異なる方法で収集できます。

方法	管理ツール	カスタム・アプリケーション
*接続時にエンド・ユーザにプロンプトを表示する：*サポートされている 4 つのデータベース・ユーザのうちの 1 つとしてユーザが認証される必要がある場合に使用します。Ultra Light のグラフィカル管理ツールでは接続オブジェクトが使用されます。 可能な場合は、ULConnectionParms オブジェクトまたは ConnectionParms オブジェクトを使用します。Open メソッドの引数である接続文字列を使用するよりも確認が簡単で、インタフェースがより体系化されます。次の項を参照してください。 ◆ Ultra Light for AppForge：「ユーザの認証」『Ultra Light - AppForge プログラミング』 ◆ Ultra Light.NET：「ユーザの認証」『Ultra Light - .NET プログラミング』 ◆ Ultra Light for C/C++：「ユーザの認証」『Ultra Light - C/C++ プログラミング』 ◆ Ultra Light for M-Business Anywhere：「ユーザの認証」『Ultra Light - M-Business Anywhere プログラミング』 ◆ Ultra Light for Embedded SQL：「ユーザの認証」『Ultra Light - C/C++ プログラミング』	X	X
*接続文字列を使用する：*配備先が単一ユーザのデバイスであるため認証が不要であるか、ユーザがアプリケーションを起動するたびにプロンプトを表示するのは不便である場合に使用します。Ultra Light のコマンド・ライン・ユーティリティでは、データベースへの接続が必要な場合に接続文字列が使用されます。格納されているファイルから値を読み取るように Ultra Light アプリケーションをプログラミングするか、アプリケーションに値をハードコードできます。次の項を参照してください。 ◆ Ultra Light for AppForge：「Ultra Light データベースへの接続」『Ultra Light - AppForge プログラミング』 ◆ Ultra Light.NET：「データベースへの接続」『Ultra Light - .NET プログラミング』 ◆ Ultra Light for C/C++：「データベースへの接続」『Ultra Light - C/C++ プログラミング』 ◆ Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」『Ultra Light - M-Business Anywhere プログラミング』 ◆ Ultra Light for Embedded SQL：「データベースへの接続」『Ultra Light - C/C++ プログラミング』	X[1]	X[3]

方法	管理ツール	カスタム・アプリケーション
*ULSQLCONNECT 環境変数を使用する：*繰り返し使用する接続パラメータを保管する場合に使用します。Ultra Light データベースをデスクトップで開発中に繰り返し指定する必要はありません。ULSQLCONNECT でパラメータとして指定されている値が、Ultra Light のデスクトップ・ツールのデフォルトになります。 すべての Ultra Light のデスクトップ管理ツールでは、パラメータの優先規則に従って、接続文字列に指定がないパラメータが ULSQLCONNECT の値で確認されます。これらの値を変更にするには、接続文字列で別の値を指定します。「ULSQLCONNECT 環境変数を使用した Ultra Light パラメータの保管」 88 ページを参照してください。	X[2]	なし

1 通常はユーザが指定

2 デスクトップ管理ツールのみ

3 通常はハードコードまたはファイルに保管

参照

- 「Ultra Light 管理ツールのパラメータの優先度」 84 ページ

パラメータの Ultra Light 接続文字列へのアセンブル

管理ツールでも、Ultra Light のカスタム・アプリケーションでも、アプリケーションの接続コードで指定する一連の接続パラメータを接続文字列と呼びます。場合によっては、アプリケーションで ConnectionParms オブジェクトのフィールドが解析され、文字列が作成されます。それ以外の場合は、接続文字列は、各パラメータ名と値をセミコロンで区切り、1 行で入力します。

```
parameter1=value1;parameter2=value2
```

Ultra Light ランタイムによって、パラメータが接続文字列にアセンブルされてから、その接続文字列を使用して接続が確立されます。たとえば、ulload ユーティリティを使用した場合、次の接続文字列を使用して新しい XML データが既存のデータベースにロードされます。次の文字列を指定するまで、指定のデータベース・ファイルに接続できません。

```
ulload -c "DBF=sample.udb;UID=DBA;PWD=sql" sample.xml
```

注意
Ultra Light では、認識できない接続文字列パラメータは無視されません。認識できない接続パラメータがあった場合は、エラーが発生します。

Ultra Light 管理ツールのパラメータの優先度

すべての Ultra Light 管理ツールでは、次の順位でパラメータが優先されます。

- CE_FILE、NT_FILE、SYMBIAN_FILE、PALM_FILE のいずれかのパラメータが指定されている場合は、常に DBF より優先されます。
- 2 つの DBF パラメータを同時に使用した場合は、最後に指定した方が優先します。
- 接続文字列で重複するパラメータを指定すると、最後に指定したパラメータが使用されます。他のパラメータはすべて無視されます。
- 接続文字列のパラメータが、ULSQLCONNECT または接続オブジェクトで指定されたパラメータより優先されます。
- 接続文字列で指定されていないパラメータは、ULSQLCONNECT 環境変数で確認されます。
- UID と PWD の*両方*の値が接続文字列でも ULSQLCONNECT でも指定されていない場合は、デフォルトの **UID=DBA** と **PWD=sql** が想定されます。

制限事項

接続文字列パラメータ値に含まれる前後のスペースはすべて無視されます。接続文字列パラメータに、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「ULSQLCONNECT 環境変数を使用した Ultra Light パラメータの保管」 88 ページ

Ultra Light 接続パラメータでのファイル・パスの指定

デバイスの物理記憶領域によって、データベースをファイルとして保存するかどうかと、サポートされている接続パラメータ (DBF、CE_FILE、NT_FILE、PALM_FILE、または SYMBIAN_FILE) でデータベースを指定するときの命名規則が決まります。

単一のプラットフォームに配備する場合や、Ultra Light デスクトップ管理ツールを使用する場合は、DBF パラメータが最適です。このマニュアルに示すほとんどのユーティリティ使用例では、DBF パラメータを使用しています。

```
ulload -c DBF=sample.udb sample.xml
```

Windows CE に関するヒント
Ultra Light 管理ツールを使用して、接続しているデバイスにすでに配備されているデータベースを管理できます。「Windows CE」 86 ページを参照してください。

プラットフォームを問わないアプリケーションを作成している場合は、プラットフォーム固有のファイル・パラメータ (CE_FILE、NT_FILE、PALM_FILE、または SYMBIAN_FILE) を使用して汎用の接続文字列を組み立ててください。たとえば、Windows CE と Palm OS 向けの AppForge コンポーネントを開発している場合、接続文字列は次のようになります。

```
Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;
CE_FILE=¥database¥MyCEDB.udb;PALM_FILE=MyPalmDB")
```


Windows デスクトップ

Windows デスクトップでは、ファイル名の制限はほとんどありません。デスクトップでは、絶対パスまたは相対パスを使用できます。

Windows CE

Windows CE デバイスでは、すべてのパスが絶対パスである必要があります。

Windows CE データベースはデスクトップまたは接続しているデバイスで管理できます。Windows CE デバイスにあるデータベースを管理するには、絶対パスの前に **wce:¥** を付ける必要があります。たとえば、次のように unload ユーティリティを使用します。

```
ulunload -c DBF=wce:¥UltraLite¥myULdb.udb c:¥out¥ce.xml
```

この例では、Ultra Light によってデータベースが CE デバイスから、Windows デスクトップの *c:¥out* フォルダにある *ce.xml* ファイルにアンロードされます。

Windows CE とバックアップ
[データベースのアンロード] ウィザードと [データベースのアップグレード] ウィザードを使用して、接続している CE デバイスにあるデータベースを管理する場合、データベースはアンロードまたはアップグレードの前に Ultra Light によってバックアップされません。したがって、これらの操作を手動で行ってから、これらのウィザードを実行してください。

Palm OS

Palm OS では、ファイル・パスの概念が使用されていません。したがって、定義方法は、ストアのタイプ (レコードベースまたは VFS) によって異なります。

ファイルベース・ストア (VFS)　VFS ボリュームにあるデータベースについては、次の構文でファイルを定義します。

vfs: [*volume-label***:** | *volume-ordinal***:**] *filename*

volume-label は、**INTERNAL** (組み込みのドライブ) または **CARD** (拡張カードまたはボリュームのラベル名) に設定できます。*volume-label* にデフォルトの文字列はありません。

volume-ordinal を設定してボリュームを指定することもできます。マウントされているボリュームの列挙はさまざまなので、選択する内部または外部のボリュームの正しい順序を設定する必要があります。デフォルト値は 0 です (プラットフォームで列挙されている最初のボリューム)。

filename の場合は、Palm OS のファイルとパスの命名規則に従って、常に絶対ファイル・パスを指定します。パスで指定したディレクトリが見つからない場合は、新しく作成されます。

レコードベースのデータ・ストア　レコードベースのデータ・ストアについては、データベース名は Palm OS のデータベース名のすべての規則に従う必要があります。たとえば、データベース名は 32 文字以内である必要があり、パスを含めることはできません。

また、データベースのロケーションに従って、DBF または PALM_FILE に適切な値を使用する必要があります。

- ulload などを使用して Palm OS のデータベースをデバイス以外の場所に保管する場合は、DBF に *.PDB* 拡張子を使用します。
- ファイルをデバイスに移動したら、*.PDB* 拡張子は HotSync コンジットによって削除されます。たとえば、デスクトップに作成したデータベースが *CustDB.pdb* という名前の場合、このデータベースをデバイスに配備するときにファイル名が *CustDB* に変わります。

Symbian OS

Symbian では、パスは絶対パスである必要があります。絶対パスではなかった場合、Symbian OS では、ファイルがデバイスのルート・ディレクトリにあると見なされます。

データベースは、ファイル・システムに保管されます。このファイル・システムは、Windows デスクトップや Windows CE デバイスで使用されているファイル・システムに似ています。たとえば、Symbian の電話では、電話のメモリに *C:¥* ドライブを使用し、フラッシュ・カードに *D:¥* ドライブまたは *E:¥* ドライブを使用できます。電話の RAM には *Z:¥* ドライブを使用できます。ドライブの文字はモデルによって異なる場合があります。

参照

- 「Ultra Light DBF 接続パラメータ」 175 ページ
- 「Ultra Light NT_FILE 接続パラメータ」 179 ページ
- 「Ultra Light CE_FILE 接続パラメータ」 177 ページ
- 「Ultra Light PALM_FILE 接続パラメータ」 181 ページ
- 「Ultra Light SYMBIAN_FILE 接続パラメータ」 184 ページ

ULSQLCONNECT 環境変数を使用した Ultra Light パラメータの保管

ULSQLCONNECT 環境変数はオプションです。インストール・プログラムでは設定を行いません。ULSQLCONNECT は、*keyword=value* をセミコロンで区切ったパラメータのリストです。

ULSQLCONNECT を使用すると、開発中に同じ接続パラメータを繰り返し Ultra Light 管理ツールに指定する必要がなくなります。カスタム・アプリケーションには ULSQLCONNECT を使用できません。

警告
等号の代わりにシャープ記号 (#) を使用しないでください。シャープ記号は Ultra Light では無視されます。Ultra Light でサポートされているすべてのプラットフォームで、環境変数の設定内で **=** を使用できます。

♦ Ultra Light デスクトップ・ツール用に ULSQLCONNECT を設定するには、次の手順に従います。

1. コマンド・プロンプトで次のように入力します。

   set ULSQLCONNECT="*parameter=value*; …**"**

2. 管理ツールに追加パラメータが必要な場合や、この環境変数で設定したデフォルト値を変更する必要がある場合は、これらの値を設定してください。ユーザが指定する値が常にこの変数より優先されます。

参照

- 「Ultra Light 管理ツールのパラメータの優先度」 84 ページ
- 「Ultra Light 接続パラメータの指定」 82 ページ
- 「Ultra Light 管理ツールのパラメータの優先度」 84 ページ

例

ULSQLCONNECT を使用して *c:¥database¥myfile.udb* というファイルに、ユーザ **demo** とパスワード **test** で接続するには、環境で次の変数を設定します。

```
set ULSQLCONNECT="DBF=c:¥database¥myfile.udb;UID=demo;PWD=test"
```

この環境変数を設定した場合は、これらのデフォルト値を変更する必要がなければ、-c 接続オプションを使用してこれらの値を指定する必要はありません。

たとえば、ulload を使用して *extra.xml* ファイルからデータベースに情報を追加する場合は、次のようにコマンドを実行します。

```
ulload -a extra.xml
```

第 6 章

Ultra Light データベースの操作

目次

Ultra Light のテーブルとカラムの操作

テーブルは、データを格納するため、またテーブル内のデータの関係を定義するために使用します。テーブルはローとカラムで構成されます。各カラムは電話番号や名前など情報の種類を特定し、各ローはデータのエントリを保持します。

Ultra Light データベースを初めて作成したときには、システム・テーブルだけが表示されます。システム・テーブルには Ultra Light のスキーマが保管されます。システム・テーブルは、必要に応じて Sybase Central で表示／非表示を切り替えることができます。

その後、アプリケーションに必要なテーブルを新しく追加できます。また、これらのテーブル内のデータをブラウズしたり、ソース・データベース内の既存のテーブル間や開いている宛先データベース間でデータをコピー・アンド・ペーストしたりできます。

ローのパックとテーブル定義

Ultra Light では、次の 2 つの形式のローを扱います。

- **アンパックされたロー**　非圧縮形式です。個々のカラム値を読み込んだり書き込んだりするには、その前に、該当するローをアンパックする必要があります。
- **パックされたロー**　アンパックされたローの圧縮形式です。各カラム値が圧縮され、ロー全体に必要なメモリが最小限に抑えられています。パックされたローのサイズは、各カラムに含まれている値だけに依存します。たとえば、同じテーブルに属する 2 つのローで、パックされたサイズが大きく異なる場合があります。LONG BINARY カラムと LONG VARCHAR カラムは、パックされたローとは別に格納されます。

Ultra Light には、パックされたローがデータベース・ページに収まらなければならないという制約があります。LONG BINARY カラムと LONG VARCHAR カラムは、パックされたローには格納されないので、ページ・サイズを超えてもかまいません。

テーブル定義は Ultra Light ランタイムがデータをパックする*前*のローについて記述していることを理解することが重要です。パックされたローのサイズは各カラムの値に依存しているので、パックされたローの要件が満たされているかどうかをテーブル定義を基に事前に判断することはできません。この理由から、Ultra Light ではアンパックされたローがページに収まらないテーブルを定義できるようになっています。ローがページに収まるかどうかを確認するには、ロー自体を挿入または更新する必要があります。ローが収まらなかった場合、Ultra Light はエラーを検出して通知します。

注意
テーブル・サイズを必要に応じて自由に大きく宣言することはできません。Ultra Light では、テーブル・サイズの上限は 64 KB に固定されています。したがって、アンパックされたローのサイズがこの上限を超えそうなテーブルを定義しようとすると、Ultra Light によって SQL エラー・コード SQLE_MAX_ROW_SIZE_EXCEEDED (-1132) が生成されます。

参照

- 「Ultra Light でのページ・サイズの考慮事項」 68 ページ

- 「データベース・テーブル」 『SQL Anywhere 10 - 紹介』
- 「データベースの設計」 『SQL Anywhere サーバ - SQL の使用法』
- 「Ultra Light のシステム・テーブル」 242 ページ

Ultra Light のテーブルの作成

Interactive SQL の SQL 文または Sybase Central のいずれかを使用して、リレーショナル・データを保持する新しいテーブルを作成できます。

Ultra Light では、ベース・テーブルだけを作成できます。ベース・テーブルは、永続的なデータを保管するテーブルです。テーブルとそのデータは、それらを明示的に削除しないかぎり存在し続けます。Ultra Light では、グローバル・テンポラリ・テーブルや宣言されたテンポラリ・テーブルはサポートされていません。

注意
Ultra Light アプリケーション内のテーブルには、プライマリ・キーが含まれます。プライマリ・キーは、Ultra Light データベースのローを統合データベースのローに関連付けるため、Mobile Link 同期を行うときにも必要です。

Sybase Central

Sybase Central では、選択したデータベースを操作しながら、これらのタスクを実行できます。

♦ **Ultra Light テーブルを作成するには、次の手順に従います (Sybase Central の場合)。**

1. Ultra Light データベースに接続します。
2. [テーブル] フォルダを開きます。
3. [ファイル] メニューから、[新規] – [テーブル] の順に選択します。

   [テーブル作成] ウィザードが表示されます。
4. [テーブル作成] ウィザードで新しいテーブルの名前を入力します。
5. [終了] をクリックします。
6. 右ウィンドウ枠の [カラム] タブで、テーブルにカラムを追加します。
7. 終了したら、[ファイル] – [保存] を選択します。

Interactive SQL

Interactive SQL では、新しいテーブルの作成時にカラムを宣言できます。

♦ **Ultra Light テーブルを作成するには、次の手順に従います (Interactive SQL の場合)。**

1. Ultra Light データベースに接続します。

2. CREATE TABLE 文を実行します。

   例 次の文は、会社内の従業員のさまざまなスキルや職業適性を記述するテーブルを新しく作成します。テーブルには、各スキルの ID 番号、名前、種別 (たとえば technical または administrative) のカラムが作成されます。

```
CREATE TABLE Skills (
  SkillID INTEGER PRIMARY KEY,
  SkillName CHAR( 20 ) NOT NULL,
  SkillType CHAR( 20 ) NOT NULL
)
```

参照

- 「Ultra Light CREATE TABLE 文」 390 ページ
- 「Ultra Light テーブルへのカラムの追加」 93 ページ

allsync と nosync の各サフィックスの使用

テーブル名に **_allsync** または **_nosync** を追加して、同期のデータ制限を制御できます。これらのサフィックスは、パブリケーションを使用してデータ制限を制御する方法の代わりに使用できます。データの優先度を制御するには、1 つまたは複数のパブリケーションを定義します。

- 名前の末尾が **_allsync** のテーブルを作成すると、最後の同期以降に変更されていない場合でも、同期時にこのテーブル内のすべてのローが同期されます。

  ヒント
  allsync テーブルには、ユーザ固有またはクライアント固有のデータを保管できます。同期するときに、統合データベースのテンポラリ・テーブルにこのテーブルのデータをアップロードすると、他のスクリプトで同期を制御するのにそのデータを使用できます。そのため、統合データベースにそのデータを保持する必要はありません。

- 名前の末尾が **_nosync** のテーブルを作成すると、このテーブルのローはすべて同期から除外されます。これらのテーブルは、統合データベースのテーブルに必要がない永続的なデータ用として使用できます。

例

CustDB.udb サンプル・データベースでは、**ULIdentifyEmployee_nosync** というテーブルが非同期のテーブルとして宣言されていることがそのテーブル名からわかります。したがって、このテーブルのデータが変更されても、Mobile Link で同期されず、情報は *CustDB.db* 統合データベースに表示されません。

参照

- 「Ultra Light のパブリケーションの操作」 105 ページ
- 「Ultra Light での同期の設計」 『Mobile Link - クライアント管理』
- 「Ultra Light の nosync テーブル」 『Mobile Link - クライアント管理』
- 「Ultra Light の allsync テーブル」 『Mobile Link - クライアント管理』
- 「Ultra Light CustDB サンプルの解説」 115 ページ

Ultra Light テーブルへのカラムの追加

テーブルが空の場合は、新しいカラムを簡単に追加できます。ただし、テーブルにデータがすでに格納されている場合は、カラムの定義にデフォルト値が含まれるか、カラムに NULL 値が許可される場合にのみ、カラムを追加できます。

Sybase Central を使用するか、直接 SQL 文 (Interactive SQL など) を実行して、このタスクを実行できます。

Sybase Central

Sybase Central では、選択したテーブルを操作しながら、これらのタスクを実行できます。

♦ 新しいカラムを Ultra Light テーブルに追加するには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. [テーブル] フォルダを開きます。
3. [カラム] タブで背景を右クリックし、[新規] – [カラム] を選択します。
4. カラムの新しい属性をすべて設定します。名前、データ型、制約、テーブルに NULL 値を許可するかどうか、などを宣言します。
5. 終了したら、[ファイル] – [保存] を選択します。

Interactive SQL

Interactive SQL では、テーブルの作成時または変更時にのみカラムを宣言できます。

♦ 新しい Ultra Light テーブルにカラムを追加するには、次の手順に従います (Interactive SQL の場合)。

1. Ultra Light データベースに接続します。
2. CREATE TABLE 文または ALTER TABLE 文を実行します。このとき、名前とその他の属性を宣言することでカラムを定義します。

   例 次の例では、図書データベース用にテーブルを作成して、借し出し図書の情報を保持します。date_borrowed のデフォルト値は、エントリが作成される日に本が貸し出されることを示します。date_returned カラムは、本が返却されるまでは NULL です。

   ```
   CREATE TABLE borrowed_book (
     loaner_name CHAR(100)     PRIMARY KEY,
     date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
     date_returned       DATE,
     book                CHAR(20)
     )
   ```

   次の例は、customer テーブルを変更して、最大 50 文字まで格納できる住所のカラムを追加します。


```
ALTER TABLE customer
ADD address CHAR(50)
```

参照

- 「カラム名の選択」 『SQL Anywhere サーバ - SQL の使用法』
- 「Ultra Light のデータ型」 262 ページ
- 「カラムのデータ型の選択」 『SQL Anywhere サーバ - SQL の使用法』
- 「Ultra Light CREATE TABLE 文」 390 ページ
- 「Ultra Light ALTER TABLE 文」 380 ページ

Ultra Light のカラム定義の変更

テーブルでは、さまざまなカラムの属性を変更するか、カラム全体を削除することで、カラム定義の構造を変更できます。変更したカラム定義は、カラムにすでに格納されているデータの条件を満たす必要があります。たとえば、カラムにすでに NULL のエントリがある場合は、カラムを変更して NULL を不許可にすることはできません。

Sybase Central または Interactive SQL のいずれかを使用して、このタスクを実行できます。

Sybase Central

Sybase Central では、選択したテーブルを操作しながら、これらのタスクを実行できます。

♦ **既存の Ultra Light カラムを変更するには、次の手順に従います (Sybase Central の場合)。**

1. Ultra Light データベースに接続します。
2. [テーブル] フォルダを開きます。
3. [カラム] タブで、必要に応じて属性を変更します。
4. 終了したら、[ファイル] – [保存] を選択します。

Interactive SQL

Interactive SQL では、ALTER TABLE 文を使用してこれらのタスクを実行できます。

♦ **既存の Ultra Light カラムを変更するには、次の手順に従います (Interactive SQL の場合)。**

1. Ultra Light データベースに接続します。
2. ALTER TABLE 文を実行します。

> **変更は慎重に行ってください**
> この後に示す例は、データベースの構造を変更する方法を示しています。いずれの場合も、文はすぐにコミットされます。したがって、変更を行うと、このテーブルを参照する項目が機能しなくなる可能性があります。

例 次の文は、SkillDescription カラムを最大 254 文字から最大 80 文字に変更します。

```
ALTER TABLE Skills
MODIFY SkillDescription CHAR( 80 )
```

次の文は、Classification カラムを削除します。

```
ALTER TABLE Skills
DROP Classification
```

次に示すコマンドは、テーブル全体の名前を変更します。

```
ALTER TABLE Skills
RENAME Qualification
```

参照

- 「カラム名の選択」 『SQL Anywhere サーバ - SQL の使用法』
- 「Ultra Light のデータ型」 262 ページ
- 「カラムのデータ型の選択」 『SQL Anywhere サーバ - SQL の使用法』
- 「Ultra Light ALTER TABLE 文」 380 ページ

Ultra Light テーブルの削除

テーブルを削除すると、データベースから削除されます。次の条件を満たすテーブルを削除できます。

- パブリケーションでアーティクルとして使用されていない
- 別のテーブルの外部キーで参照されているカラムがない

このような場合は、パブリケーションを変更するか、外部キーを削除してから、テーブルを削除します。

Sybase Central または Interactive SQL のいずれかを使用して、このタスクを実行できます。

Sybase Central

Sybase Central では、選択したテーブルを操作しながら、これらのタスクを実行できます。

♦ Ultra Light テーブルを削除するには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. [テーブル] フォルダを開きます。
3. テーブルを選択し、[編集] – [削除] を選択します。

Interactive SQL

Interactive SQL では、テーブルの削除をテーブルのドロップとも呼びます。テーブルを削除するには、DROP TABLE 文を使用します。

♦ **Ultra Light テーブルを削除するには、次の手順に従います (Interactive SQL の場合)。**

1. Ultra Light データベースに接続します。
2. DROP TABLE 文を実行します。

   例 次に示す DROP TABLE コマンドは、Skills テーブルからすべてのレコードを削除し、データベースから Skills テーブルの定義を削除します。

   ```
   DROP TABLE Skills
   ```

   CREATE 文と同様、DROP 文はテーブルを削除する前と後に COMMIT 文を自動的に実行します。したがって、最後に COMMIT または ROLLBACK を実行した後の変更はすべて確定されます。DROP 文では、テーブル上のインデックスもすべて削除されます。

参照

♦ 「Ultra Light DROP TABLE 文」 398 ページ

Ultra Light テーブル内の情報のブラウズ

Sybase Central または InteractiveSQL を使用して、Ultra Light データベースのテーブルに保持されているデータをブラウズできます。テーブルには、ユーザ・テーブルとシステム・テーブルがあります。現在のデータベースの表示でシステム・テーブルの表示と非表示を切り替えてテーブルをフィルタできます。Ultra Light には所有権の概念がないので、すべてのユーザがすべてのテーブルをブラウズできます。

Sybase Central

Sybase Central では、選択したデータベースを操作しながら、これらのタスクを実行できます。

♦ **Ultra Light テーブルをブラウズするには、次の手順に従います (Sybase Central の場合)。**

1. Ultra Light データベースに接続します。
2. システム・テーブルが非表示になっているときに 1 つまたは複数のテーブルのデータをブラウズするには、[コンテンツ] ウィンドウ枠の背景を右クリックし、[システム・オブジェクトを表示] を選択します。
3. テーブルを表示するには、[テーブル] をクリックします。
4. テーブルのデータをブラウズするには、テーブル名をダブルクリックします。
5. 右ウィンドウ枠で、[データ] タブをクリックします。

♦ **Ultra Light のシステム・テーブルをフィルタするには、次の手順に従います (Sybase Central の場合)。**

1. Ultra Light データベースに接続します。

2. 接続先のデータベース名を右クリックし、[システム・オブジェクトの非表示] または [システム・オブジェクトを表示] をクリックします。

Interactive SQL

Interactive SQL では、SELECT 文を使用してこれらのタスクを実行できます。

♦ Ultra Light のユーザ・テーブルをブラウズするには、次の手順に従います (Interactive SQL の場合)。

1. データベースに接続します。
2. SELECT 文を実行し、ブラウズするユーザ・テーブルを指定します。

♦ Ultra Light のシステム・テーブルをブラウズするには、次の手順に従います (Interactive SQL の場合)。

1. データベースに接続します。
2. SELECT 文を実行し、ブラウズするシステム・テーブルを指定します。

   例 たとえば、Interactive SQL で systable テーブルの内容を [結果] ウィンドウ枠の [結果] タブに表示するには、次のコマンドを実行します。

```
SELECT *
FROM SYSTABLE
```

参照

- 「Ultra Light システム・テーブルのリファレンス」 241 ページ

Ultra Light データベースのデータのコピー・アンド・ペースト

Sybase Central では、複数の方法でテーブルまたはカラムのコピー・アンド・ペーストとドラッグ・アンド・ドロップを行うことができます。したがって、データベース内または複数のデータベース間でオブジェクトを共有または移動できます。コピー・アンド・ペーストまたはドラッグ・アンド・ドロップで、次の表に示すデータを共有できます。

ターゲット	結果
別の Ultra Light／SQL Anywhere データベース	新しいオブジェクトが作成され、元のオブジェクトのコードが新しいオブジェクトにコピーされます。
同じ Ultra Light データベース	オブジェクトのコピーが作成されるので、新しいオブジェクトの名前を変更します。

注意
Mobile Link で開いているデータベースのデータをコピーし、Ultra Light データベースにペーストできます。ただし、Ultra Light のデータを、Mobile Link で開いているデータベースにペーストすることはできません。

Sybase Central

Sybase Central では、次にリストされているオブジェクトのいずれかをコピーすると、そのオブジェクトの SQL がクリップボードにコピーされるため、Interactive SQL やテキスト・エディタなど他のアプリケーションにペーストできます。たとえば、Sybase Central でインデックスをコピーし、テキスト・エディタにペーストすると、そのインデックスの CREATE INDEX 文が表示されます。

- アーティクル
- カラム
- 外部キー
- インデックス
- パブリケーション
- テーブル
- 一意性制約

Interactive SQL

Interactive SQL では、結果セットから別のオブジェクトにデータをコピーすることもできます。

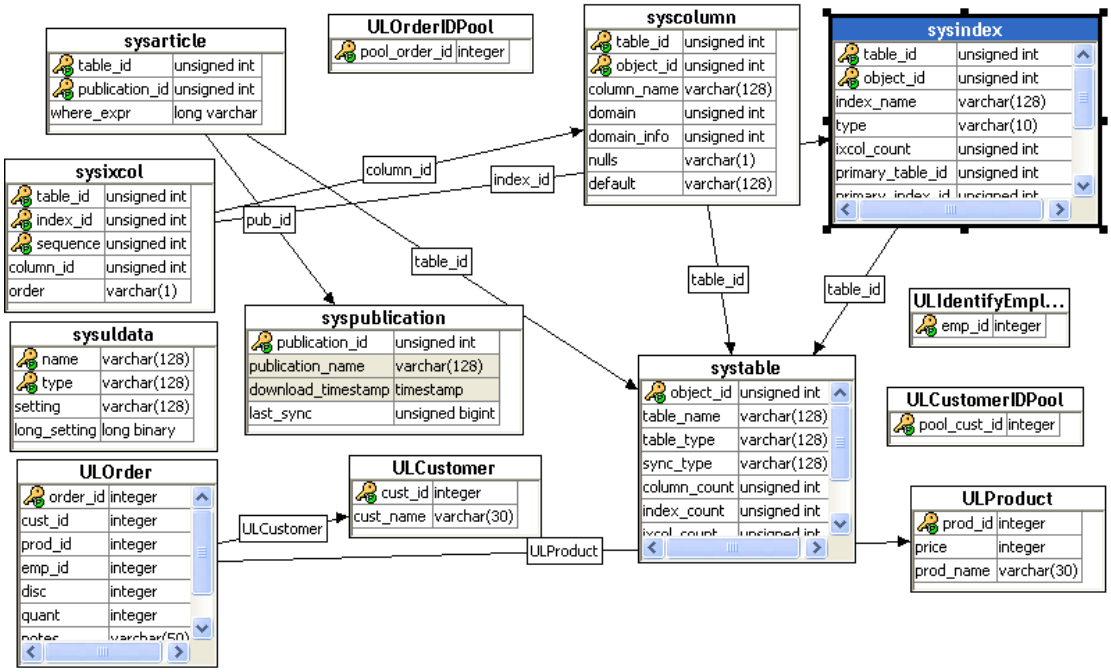
- 結果を指定のオブジェクトにコピーするには、SELECT 文を使用します。
- データベース内の別の場所にある 1 つのローまたは複数のローをテーブルに挿入するには、INSERT 文を使用します。

参照

- 「SQL Anywhere プラグインのデータベース・オブジェクトのコピー」 『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light INSERT 文」 399 ページ
- 「Ultra Light SELECT 文」 401 ページ

Ultra Light プラグインから ER 図を表示する

Ultra Light プラグインからデータベースに接続している場合は、データベースにあるテーブルの ER 図を表示できます。データベースを選択している状態で、右ウィンドウ枠の [ER 図] タブをクリックして ER 図を表示します。



ER 図でオブジェクトを並べ替えると、変更は Sybase Central セッション間で保持されます。テーブルをダブルクリックすると、そのテーブルのカラム定義を表示できます。

参照

- 「データベース設計の概念」 『SQL Anywhere サーバ - SQL の使用法』
- 「ER (実体関連) 図」 『SQL Anywhere サーバ - SQL の使用法』

Ultra Light のインデックスの操作

インデックスは、1 つまたは複数のカラムの値に基づいた、テーブルのローの順序 (昇順または降順) を示します。Ultra Light は、クエリを最適化するときに、既存のインデックスをスキャンして、クエリで指定されているテーブルにインデックスがあるかどうかを確認します。インデックスによってより短時間でローを返すことができる場合は、インデックスが使用されます。アプリケーションで Ultra Light のテーブル API を使用している場合は、ローをスキャンする順序を決定するインデックスを指定できます。

パフォーマンスに関するヒント
インデックスによって、特にテーブルが大きい場合は、クエリのパフォーマンスを向上できます。クエリで特定のインデックスが使用されているかどうかを確認するには、Interactive SQL. でクエリのアクセス・プランを確認できます。
また、プランを返すメソッドがある PreparedStatement オブジェクトを Ultra Light アプリケーションに含めることもできます。

複合インデックスについて

マルチカラムのインデックスは複合インデックスとも呼ばれます。インデックスにカラムを追加すると検索対象を限定できますが、2 カラムのインデックスを使用することと 2 つの別個のインデックスを使用することは異なります。たとえば、次の文では 2 カラムの複合インデックスが作成されます。

```
CREATE INDEX name
ON Employees ( Surname, GivenName )
```

複合インデックスは、最初のカラムだけでは高い選択性が得られない場合に役立ちます。たとえば、Surname と GivenName に対する複合インデックスは、従業員の姓が同じ場合に便利です。各従業員はユニークな ID を持っており、カラム Surname は追加の選択性を提供しないため、EmployeeID と Surname の複合インデックスは役に立ちません。

参照

- 「インデックス・スキャンの使用」 38 ページ
- 「Ultra Light のクエリ・アクセス・プラン」 291 ページ
- 「複合インデックス」 『SQL Anywhere サーバ - SQL の使用法』
- Ultra Light for AppForge :「テーブル API を使用したデータの使用」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「ULPreparedStatement クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET :「テーブル API によるデータ・アクセスと操作」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET :「Prepare メソッド」 『Ultra Light - .NET プログラミング』
- Ultra Light for C++ :「テーブル API を使用したデータへのアクセス」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++ :「UltraLite_PreparedStatement クラス」 『Ultra Light - C/C++ プログラミング』

- Ultra Light for M-Business Anywhere :「テーブル API を使用したデータの使用」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere :「 PreparedStatement クラス」 『Ultra Light - M-Business Anywhere プログラミング』

インデックスを使用する場合

インデックスは次の場合に使用します。

- **Ultra Light で参照整合性を保つ必要がある場合** インデックスは Ultra Light に、テーブル内のローに一意性制約を強制するための手段も提供します。よく似ているデータにインデックスを追加する必要はありません。
- **特定のクエリのパフォーマンスがアプリケーションにとって重要である場合** インデックスによってクエリのパフォーマンスが向上し、そのクエリのパフォーマンスがアプリケーションにとって重要であり、そのクエリを頻繁に使用する場合は、インデックスを使用します。テーブルが非常に小さい場合を除き、インデックスによって検索のパフォーマンスが大幅に向上するので、データを頻繁に検索する場合は一般にインデックスを使用することをおすすめします。
- **複雑なクエリがある場合** 複雑なクエリ (たとえば、JOIN 句、GROUP BY 句、ORDER BY 句を使用するもの) は、インデックスを使用するとパフォーマンスが大幅に向上する可能性があります。ただし、パフォーマンスの向上の程度を確認するのは困難である場合があります。したがって、インデックスを使用した場合と使用しなかった場合でクエリをテストし、パフォーマンスを確認することをおすすめします。
- **Ultra Light テーブルのサイズが大きい場合** ローを検索する平均時間は、テーブルのサイズに比例して長くなります。したがって、非常に大きなテーブルの検索効率を向上させるには、インデックスの使用を検討してください。インデックスを使用すると、インデックスが付いているカラムについては、Ultra Light でローが短時間で見つかります。インデックスがない場合、Ultra Light ではテーブル内のすべてのローを検索し、ローが探索条件を満たすかどうかを確認する必要があるので、大きなテーブルの場合は時間がかかります。
- **Ultra Light クライアント・アプリケーションで大量の挿入、更新、または削除の操作を行わない場合** Ultra Light ではインデックスがデータ自体とともに保持されるので、これらの操作では、インデックスによってパフォーマンスが下がります。したがって、インデックスの使用は、前述のように頻繁に問い合わせるデータだけに制限してください。Ultra Light のデフォルトのインデックス (プライマリ・キーと一意性制約のインデックス) で十分である可能性があります。
- **WHERE 句か ORDER BY 句またはその両方を使用する場合** これらの句の対象となるカラムにインデックスを使用すると、これらの句の評価を高速化できます。特にインデックスによってマルチカラムの ORDER BY 句を最適化できます。ただし、インデックスと ORDER BY 句でのカラムの配置がまったく同じである必要があります。

インデックス・タイプの選択

Ultra Light では、インデックスのタイプにユニーク・キー、ユニーク・インデックス、ユニークでないインデックスがあります。これらのタイプでは、インデックスで許可されることが異なります。

インデックスの特性	ユニーク・キー	ユニーク・インデックス	ユニークでないインデックス
インデックス・カラムに同じ値があるローの重複するインデックス・エントリを許可する	no	no	yes
インデックス・カラムに NULL 値を許可する	no	yes	yes

注意
ユニーク・キーには外部キーを作成できますが、ユニーク・インデックスには作成できません。また、自分でキー・カラムにインデックスを設定する必要はありません。Ultra Light によってユニーク・キーのインデックスが自動的に作成され、管理されます。

参照

- 「Ultra Light インデックスの追加」 102 ページ

Ultra Light インデックスの追加

Sybase Central または Interactive SQL のいずれかを使用して、このタスクを実行できます。

注意
Ultra Light では重複する、または冗長なインデックスは検出されません。インデックスはデータベース内のデータとともに保持されるので、インデックスは慎重に追加してください。

Sybase Central

Sybase Central では、選択したデータベースを操作しながら、これらのタスクを実行できます。

♦ 指定された Ultra Light テーブルに対する新しいインデックスを設定するには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. [インデックス] フォルダを開きます。
3. [ファイル] – [新規] – [インデックス] を選択します。

   [インデックス作成] ウィザードが表示されます。
4. インデックスに名前を付け、リストからテーブルを選択します。[次へ] をクリックします。

5. ウィザードの指示に従います。

   新しいインデックスが [インデックス] フォルダに表示されます。

Interactive SQL

Interactive SQL では、CREATE INDEX 文を使用してこれらのタスクを実行できます。

♦ 指定された Ultra Light テーブルに対する新しいインデックスを設定するには、次の手順に従います (Interactive SQL の場合)。

1. Ultra Light データベースに接続します。
2. CREATE INDEX 文を実行します。「Ultra Light CREATE INDEX 文」 387 ページを参照してください。

   設定したデフォルトの最大ハッシュ・サイズでインデックスが作成されます。デフォルト以外の値を使用するインデックスを作成するには、WITH MAX HASH SIZE *value* 句を使用してこのインデックス・インスタンスの新しい値を設定します。

   例 従業員情報を追跡するデータベースでの従業員の姓の検索を高速化し、このインデックスに対するクエリのパフォーマンスをチューニングするために、次の文で EmployeeNames というインデックスを作成し、ハッシュ・サイズを 20 バイトに増加します。

```
CREATE INDEX EmployeeNames
ON Employees (Surname, GivenName)
WITH MAX HASH SIZE 20
```

参照

- 「Ultra Light CREATE INDEX 文」 387 ページ

インデックスの削除

インデックスを削除すると、データベースから削除されます。

Sybase Central または Interactive SQL のいずれかを使用して、このタスクを実行できます。

Sybase Central

Sybase Central では、選択したデータベースを操作しながら、これらのタスクを実行できます。

♦ Ultra Light インデックスを削除するには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. [インデックス] フォルダを開きます。
3. インデックスを選択し、[編集] – [削除] を選択します。

Interactive SQL

Interactive SQL では、テーブルの削除をテーブルのドロップとも呼びます。これらのタスクは DROP INDEX 文で実行できます。

♦ **Ultra Light インデックスを削除するには、次の手順に従います (Interactive SQL の場合)。**

1. データベースに接続します。
2. DROP INDEX 文を実行します。

   例 次の文は、データベースから EmployeeNames インデックスを削除します。

   ```
   DROP INDEX EmployeeNames
   ```

参照

♦ 「Ultra Light DROP INDEX 文」 396 ページ

Ultra Light のパブリケーションの操作

パブリケーションは、Mobile Link サーバと一度に同期させる Ultra Light のデータのサブセットを示すデータベース・オブジェクトです。Ultra Light データベース内のすべてのテーブルと、そのテーブルのすべてのローを同期させる場合は、パブリケーションを作成しないでください。

パブリケーションはアーティクルのセットで構成されます。各アーティクルはテーブル全体、またはテーブル内の選択されたローです。WHERE 句を使用してこのローのセットを定義することができます （Palm OS 上の HotSync を除く）。

各データベースには、同期の論理によって、複数のパブリケーションを作成できます。たとえば、優先度の高いデータのパブリケーションを作成することができます。ユーザは高速無線ネットワークを経由してこのデータを同期できます。無線ネットワークには使用料がかかります。使用料を抑えるには、ビジネスに必須のデータのみを同期します。緊急でないデータは、後でクレードルから同期します。

Sybase Central または CREATE PUBLICATION 文を使用して、パブリケーションを作成します。Sybase Central では、[パブリケーション] フォルダにすべてのパブリケーションとアーティクルがあります。

使用上の注意

- Ultra Light パブリケーションでは、カラムのサブセットの定義と、SUBSCRIBE BY 句がサポートされていません。Ultra Light テーブルのカラムが SQL Anywhere 統合データベースのテーブルと正確に一致しない場合は、Mobile Link スクリプトを使用して、これらの違いを解消してください。
- パブリケーションはどのカラムが選択されているかは確認しますが、それらが送信される順序は確認しません。カラムは、CREATE TABLE 文で定義された順に常に送信されます。
- パブリケーションでテーブル同期順序を設定する必要はありません。配備においてテーブル順序が重要な場合は、Ultra Light データベースを同期するときに Table Order 同期パラメータを設定して、テーブル順序を設定できます。
- Ultra Light ではオブジェクトの所有権がサポートされていないので、すべてのユーザがパブリケーションを削除できます。

参照

- 「Ultra Light のテーブルの順序」 『Mobile Link - クライアント管理』
- 「データのパブリッシュ」 『Mobile Link - クライアント管理』
- 「Ultra Light での同期の設計」 『Mobile Link - クライアント管理』
- 「同期スクリプトの概要」 『Mobile Link - サーバ管理』

Ultra Light テーブル全体のパブリッシュ

作成できる最も簡単なパブリケーションは、単一のアーティクルで構成されます。このアーティクルは、テーブルのすべてのローとカラムで構成されます。

Sybase Central または Interactive SQL のいずれかを使用して、このタスクを実行できます。

Sybase Central

Sybase Central では、接続先のデータベースを操作しながら、このタスクを実行できます。

♦ Ultra Light のテーブル全体を 1 つ以上パブリッシュするには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. [パブリケーション] フォルダを開きます。
3. パブリケーションを作成します。

   [ファイル] – [新規] – [パブリケーション] を選択します。[パブリケーション作成] ウィザードが表示されます。
4. 新しいパブリケーションの名前を入力し、[次へ] をクリックします。
5. [テーブル] タブで、[使用可能なテーブル] のリストからテーブルを 1 つ選択します。[追加] をクリックします。選択したテーブルが、右側の [選択したテーブル] リストに表示されます。
6. オプションで、さらにテーブルを追加することもできます。テーブルの順序は重要ではありません。
7. [完了] をクリックします。

Interactive SQL

Interactive SQL では、CREATE PUBLICATION 文を使用してこのタスクを実行できます。

♦ Ultra Light のテーブル全体を 1 つ以上パブリッシュするには、次の手順に従います (Interactive SQL の場合)。

1. Ultra Light データベースに接続します。
2. 新しく作成するパブリケーションの名前とパブリッシュするテーブルの名前を指定して、CREATE PUBLICATION 文を実行します。

   例 次の文で、customer テーブル全体をパブリッシュするパブリケーションを作成します。

   ```
   CREATE PUBLICATION pub_customer (
     TABLE customer
   )
   ```

参照

- 「Ultra Light CREATE PUBLICATION 文」 389 ページ
- 「Ultra Light クライアント」 『Mobile Link - クライアント管理』

Ultra Light テーブルのローのサブセットのパブリッシュ

パブリケーションには、テーブルの特定のローだけが含まれます。Sybase Central と Interactive SQL のどちらを使用していても、WHERE 句はアップロードされるローを次のように限定します。

- 変更されたロー、かつ
- WHERE 句の検索条件を満たしているロー

すべての変更されたローをアップロードする場合は、WHERE 句を指定しないでください。

Palm OS
このプラットフォームでは、CREATE PUBLICATION 文で WHERE 句を使用できません。

WHERE 句で使用できない項目

WHERE 句内の探索条件では、アーティクルに含まれるカラムだけを参照できます。また、WHERE 句では次のいずれも使用できません。

- サブクエリ
- 変数
- 非決定的関数

これらの条件は強制ではありませんが、従わなかった場合、予期しない結果が発生します。WHERE 句に関連するエラーは、パブリケーションの定義時ではなく実行時に生成されます。

Sybase Central

Sybase Central では、接続先のデータベースを操作しながら、このタスクを実行できます。

♦ Ultra Light テーブル内の一部のローだけをパブリッシュするには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. [パブリケーション] フォルダを開きます。
3. 新しいパブリケーションを作成します。

   [ファイル] – [新規] – [パブリケーション] を選択します。[パブリケーション作成] ウィザードが表示されます。
4. 新しいパブリケーションの名前を入力します。[次へ] をクリックします。
5. [テーブル] タブで、[使用可能なテーブル] のリストからテーブルを 1 つ選択します。[追加] をクリックします。選択したテーブルが右側の [選択したテーブル] のリストに追加されます。

6. [WHERE 句] タブでテーブルを選択し、下側のボックスに探索条件を入力します。オプションで、[挿入] ダイアログを使用して探索条件をフォーマットできます。
7. [完了] をクリックします。

Interactive SQL

Interactive SQL では、CREATE PUBLICATION 文を使用してこのタスクを実行できます。

♦ Ultra Light で WHERE 句を使用してパブリケーションを作成するには、次の手順に従います (Interactive SQL の場合)。

1. Ultra Light データベースに接続します。
2. パブリケーション対象のテーブルと WHERE 条件を含む CREATE PUBLICATION 文を実行します。

   例 次の例は、単一のアーティクルで構成され、営業担当者 ID 番号 856 のすべての受注情報が含まれるパブリケーションを作成します。

```
CREATE PUBLICATION pub_orders_samuel_singer
( TABLE SalesOrders
    WHERE SalesRepresentative = 856 )
```

参照

- 「Ultra Light CREATE PUBLICATION 文」 389 ページ
- 「Ultra Light クライアント」 『Mobile Link - クライアント管理』

Ultra Light のパブリケーションの削除

パブリケーションは Sybase Central または Interactive SQL を使用して削除できます。

Sybase Central

Sybase Central では、接続先のデータベースを操作しながら、このタスクを実行できます。

♦ パブリケーションを削除するには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. [パブリケーション] フォルダを開きます。
3. 対象のパブリケーションを右クリックし、ポップアップ・メニューから [削除] を選択します。

Interactive SQL

Interactive SQL では、パブリケーションの削除をパブリケーションのドロップとも呼びます。このタスクは DROP PUBLICATION 文で実行できます。

- **パブリケーションを削除するには、次の手順に従います (Interactive SQL の場合)。**

1. Ultra Light データベースに接続します。
2. DROP PUBLICATION 文を実行します。

   例 次の文は、パブリケーション pub_orders を削除します。

   ```
   DROP PUBLICATION pub_orders
   ```

参照

- 「Ultra Light DROP PUBLICATION 文」 397 ページ
- 「Ultra Light クライアント」 『Mobile Link - クライアント管理』

Ultra Light のユーザの操作

Ultra Light データベースではユーザ ID とパスワードは暗号化されているので、定義されているユーザのリストは Sybase Central だけで確認できます。

考慮事項と制限事項

ユニークなユーザ ID を作成するときは、次の制限事項に注意してください。

- Ultra Light のユーザ ID は、Mobile Link のユーザ名やその他の SQL Anywhere のユーザ ID とは別です。
- Ultra Light では最大 4 つのユニークなユーザがサポートされます。
- ユーザ ID とパスワードの値はいずれも文字数制限が 31 文字です。
- パスワードは常に大文字と小文字が区別され、ユーザ ID は常に大文字と小文字が区別されません。パスワードは Sybase Central でいつでも変更できます。.
- ユーザ ID に含まれる前後のスペースはすべて無視されます。ユーザ ID に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。
- Ultra Light では、予防措置として、パスワードがハッシュされてから保存されます。したがって、パスワードは Sybase Central だけで変更できます。
- ユーザ ID は一度作成すると変更できません。変更する必要がある場合は、ユーザ ID を削除し、新しい ID を追加してください。

新しい Ultra Light ユーザの追加

Ultra Light では、Interactive SQL でのユーザの作成がサポートされていません。ただし、次の方法でユーザを追加できます。

- Sybase Central を使用して [ユーザ] フォルダにユーザを追加する。詳細については、この後に示す手順を参照してください。
- Connection オブジェクトに GrantConnectTo 関数を使用して Ultra Light アプリケーションから新しいユーザを追加する

♦ 新しい Ultra Light ユーザを作成するには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. [ユーザ] フォルダを開きます。
3. [ファイル] メニューから、[新規] – [ユーザ] を選択します。

   [ユーザ作成] ウィザードが表示されます。

4. ウィザードの指示に従います。さまざまなユーザ ID とパスワードの組み合わせが Ultra Light でどのように解釈されるかを理解している必要があります。詳細については、「ユーザ ID とパスワードの組み合わせの解釈」 80 ページを参照してください。

参照

- Ultra Light for AppForge：「ユーザの認証」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ユーザの認証」 『Ultra Light - .NET プログラミング』
- Ultra Light for C/C++：「ユーザの認証」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for M-Business Anywhere：「ユーザの認証」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for Embedded SQL：「ユーザの認証」 『Ultra Light - C/C++ プログラミング』

既存の Ultra Light ユーザの削除

Ultra Light では、Interactive SQL でのユーザの削除がサポートされていません。ただし、次の方法でユーザを削除できます。

- Sybase Central を使用して [ユーザ] フォルダからユーザを削除する。詳細については、この後に示す手順を参照してください。
- Connection オブジェクトに RevokeConnectFrom 関数を使用して Ultra Light アプリケーションからユーザを削除する

既存の Ultra Light ユーザを削除するには、次の手順に従います (Sybase Central の場合)。

1. Ultra Light データベースに接続します。
2. データベースをブラウズし、[ユーザ] をクリックします。
3. [ユーザ] ウィンドウ枠内でユーザを右クリックし、[削除] をクリックします。

参照

- Ultra Light for AppForge：「ユーザの認証」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ユーザの認証」 『Ultra Light - .NET プログラミング』
- Ultra Light for C/C++：「ユーザの認証」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for M-Business Anywhere：「ユーザの認証」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for Embedded SQL：「ユーザの認証」 『Ultra Light - C/C++ プログラミング』

Ultra Light データベース設定の表示とデータベース・オプションの変更

設定したデータベース・プロパティは Sybase Central だけで確認できます。データベース・プロパティはデータベースの作成後は変更できません。プロパティを変更する必要がある場合は、新しいデータベースを作成し、データをロードする必要があります。一方、データベース・オプションはいつでも変更できます。

♦ Ultra Light のデータベース・プロパティをブラウズするには、次の手順に従います (Sybase Central の場合)。

1. データベースに接続します。
2. 接続先のデータベースをブラウズし、データベースを右クリックして [プロパティ] を選択します。

   [データベースのプロパティ] ダイアログが表示されます。
3. [詳細情報] タブを選択します。

   データベース・プロパティはプロパティ名のアルファベット順で表示されます。データベース・プロパティを値の順にソートするには、[値] カラムをクリックします。
4. データベース・プロパティのブラウズ中にプロパティが変更された可能性がある場合は、[再表示] をクリックします。

♦ Ultra Light のデータベース・オプションをブラウズまたは変更するには、次の手順に従います (Sybase Central の場合)。

1. データベースに接続します。
2. 接続先のデータベースをブラウズし、データベースを右クリックして [オプション] を選択します。

   [データベースのオプション] ダイアログ・ボックスが表示されます。
3. オプションを設定またはリセットする場合は、[値] フィールドに新しいオプションを入力します。
4. [すぐに設定] または [すぐにリセット] をクリックして変更をコミットします。

♦ データベース・オプションを変更するには、次の手順に従います (Interactive SQL の場合)。

1. Ultra Light データベースに接続します。
2. SET OPTION 文を実行します。

   例 次の文を実行すると、global_database_id オプションが 100 に設定されます。

   ```
   SET OPTION global_database_id=100
   ```

参照

- 「Ultra Light データベースの作成」 54 ページ
- 「Ultra Light データベース設定のリファレンス」 135 ページ

第 7 章

Ultra Light CustDB サンプルの解説

目次

CustDB の概要

CustDB とは

CustDB (顧客データベース) を使用して、Mobile Link による SQL Anywhere 統合データベースとの同期など、多層データベース管理ソリューションの一部としての Ultra Light のさまざまな側面について学ぶことができます。この章のデスクトップベースのチュートリアルでは、CustDB の例を使用して Ultra Light の機能と動作を示します。

CustDB は、店舗販売時点顧客管理ソリューションの簡単な例であり、次の要素から構成されます。

- SQL Anywhere 統合データベース。このデータベースは販売管理データが事前に挿入されています。
- Ultra Light リモート・データベース。このデータベースは最初は空です。
- Ultra Light クライアント・アプリケーション。
- Mobile Link サーバ同期サンプル。同期スクリプトが事前に作成されています。

サポートされている各プログラミング・インタフェースとプラットフォーム向けに複数のバージョンのアプリケーション・コードがありますが、この後のチュートリアルでは、Windows デスクトップ用アプリケーションのコンパイル済みバージョンだけを使用します。各バージョンで Ultra Light の機能が実装されますが、各プラットフォームの規則に応じて機能が異なる場合があります。

CustDB の機能

CustDB を使用すると、販売担当者が取引を追跡、モニタできます。また、次の 2 種類のユーザからの情報が収集されます。

- ユーザ ID 51、52、 53 で認証された販売担当者
- ユーザ ID 50 で認証されたモバイル管理者

これらのユーザによって収集された情報は、統合データベースと同期できます。

Ultra Light の CustDB チュートリアルの目的

各レッスンを終了したら、次の操作を行う方法を理解できます。

- Mobile Link サーバを実行して、統合データベースと Ultra Light リモートの間でデータ同期を行う
- Sybase Central を使用して Ultra Light リモートのデータをブラウズする
- Ultra Light のコマンド・ライン・ユーティリティを使用して Ultra Light データベースを管理する

参照

- 「CustDB サンプル・ファイルの場所」 118 ページ

- 「シナリオ」『Mobile Link - クイック・スタート』
- 「CustDB サンプル内のユーザ」『Mobile Link - クイック・スタート』
- 「CustDB データベース内のテーブル」『Mobile Link - クイック・スタート』

CustDB サンプル・ファイルの場所

SQL Anywhere インストーラは、ソフトウェアのインストール時に CustDB を自動的にインストールします。次の表は、これらのファイルのロケーションと説明を示します。

SQL Anywhere CustDB データベース

統合データベースです。インストール中、このデータベース用に SQL Anywhere 10 CustDB という名前の ODBC データ・ソースが作成されます。

CustDB のインストール環境は、既存のサンプルを使用するか、新しいファイルを再作成するかによって異なります。

- 既存のサンプルを使用する場合：*samples-dir¥UltraLite¥CustDB¥custdb.db*
- 統合された *CustDB.db* ファイルに同期された変更内容を消去して、新しいバージョンを使用する場合：*samples-dir¥UltraLite¥CustDB¥newdb.bat*

このファイルのスキーマの詳細については、「Mobile Link CustDB サンプルの解説」 『Mobile Link - クイック・スタート』を参照してください。

Ultra Light CustDB データベース

情報のサブセットだけを含む、統合データベースのリモート・バージョンです。含まれる情報はデータベースを同期するユーザによって異なります。

ファイル名とロケーションは、プラットフォーム、プログラミング言語、またはデバイスによっても異なることがあります。

- AppForge の場合：*samples-dir¥UltraLiteforAppForge¥*。必要なデバイスの Crossfire または MobileVB に固有なファイル・バージョンがあります。ほとんどの場合、ファイル名は *ul_CustDB.udb* です。ただし、このファイルの Palm バージョンをデスクトップに移動すると、*ul_CustDB.udb.pdb* になります。
- Ultra Light.NET の場合：*samples-dir¥UltraLite.NET¥CustDB¥Common¥*
- その他のすべてのプラットフォームと API の場合：*samples-dir¥UltraLite¥CustDB¥custdb.udb*

RDBMS 固有の構築スクリプト

サポートされている任意の RDBMS 用に CustDB 統合データベースを再構築する SQL スクリプトです。

samples-dir¥MobiLink¥CustDB ディレクトリに次のファイルがあります。

- SQL Anywhere の場合：*custdb.sql*
- Adaptive Server Enterprise の場合：*custase.sql*
- Microsoft SQL Server の場合：*custmss.sql*
- Oracle の場合：*custora.sql*

- IBM DB2 の場合：*custdb2.sql*

統合データベースの設定の詳細については、「CustDB 統合データベースの設定」 『Mobile Link - クイック・スタート』を参照してください。

Ultra Light CustDB クライアント・アプリケーションと ReadMe ファイル

Ultra Light リモート・データベースへの使いやすいインタフェースを提供するエンド・ユーザ・ツールです。サポートされている API ごとにサンプル・クライアントがインストールされます。

各クライアント・アプリケーションには、*ReadMe.html* ファイルまたは *ReadMe.txt* ファイルが含まれています。ファイルの内容はそれぞれ異なります。ただし、一部のファイルには、サンプルの構築と実行に必要な手順の概略が記載されています。

アプリケーションと ReadMe のロケーションは、環境変数によって異なります。「レッスン 1：CustDB アプリケーションの構築と実行」 120 ページを参照してください。

SQL 同期論理

Ultra Light データベースの情報を問い合わせて修正するための SQL 文と、統合データベースとの同期を開始するために必要な呼び出しです。

samples-dir¥UltraLite¥CustDB¥custdb.sqc

参照

- 「Mobile Link 統合データベース」 『Mobile Link - サーバ管理』

レッスン 1：CustDB アプリケーションの構築と実行

CustDB アプリケーションは、数多くの開発環境用に構築されます。すべての環境に適用される一般的な手順については、次の項を参照してください。

♦ **CustDB アプリケーションを構築して実行するには、次の手順に従います。**

1. CustDB アプリケーションを構築します。
   a. 適切な環境で CustDB プロジェクト・ファイルを開きます。
   b. ソース・コードをコンパイルします。
2. CustDB アプリケーションを実行します。
   a. CustDB 実行ファイルをモバイル・デバイスに配備します。
   b. Ultra Light CustDB データベースをモバイル・デバイスに配備します。
   c. CustDB 実行ファイルを実行します。

各開発環境に関する詳細については、次の項を参照してください。

Windows 32 ビット・デスクトップ用 Ultra Light

CustDB アプリケーションを実行する前に構築する必要はありません。CustDB サンプルを Windows デスクトップで使用する場合についてこの章で説明している手順に従ってください。

CustDB 実行ファイルは *install-dir¥ultralite¥CustDB¥win32¥386* ディレクトリにあります。

Ultra Light for C/C++

♦ **C/C++ のすべてのバージョン** C/C++ には数多くの開発環境が存在するので、C/C++ CustDB プロジェクト・ファイルは複数のバージョンが用意されています。ほとんどのバージョンでは、汎用ファイルを使用しています。これらのファイルは *samples-dir¥UltraLite¥Custdb¥* ディレクトリにあります。

C/C++ CustDB アプリケーションのすべてのバージョンの詳細については、*samples-dir¥UltraLite¥Custdb¥readme.txt* を参照してください。

♦ **CodeWarrior for Palm OS** プロジェクト・ファイルは samples-dir¥UltraLite¥CustDB¥cwcommon¥ ディレクトリと samples-dir¥UltraLite¥CustDB¥cw¥ ディレクトリにあります。

C/C++ CustDB アプリケーションの構築の詳細については、「CodeWarrior を使用した CustDB サンプル・アプリケーションの構築」『Ultra Light - C/C++ プログラミング』を参照してください。

♦ **CodeWarrior for Symbian OS** プロジェクト・ファイルは、*samples-dir¥UltraLite¥CustDB¥Symbian* ディレクトリにあります。

サンプル・アプリケーションの構築の詳細については、「CodeWarrior for Symbian を使用したアプリケーションの開発」『Ultra Light - C/C++ プログラミング』を参照してください。C++

CustDB アプリケーション構築の手順の詳細については、*samples-dir¥UltraLite¥CustDB¥Symbian¥readme.txt* を参照してください。

- **Visual Studio**　プロジェクト・ファイルは、Visual Studio のバージョンに応じて、*samples-dir¥UltraLite¥CustDB¥vs7* ディレクトリまたは *samples-dir¥UltraLite¥CustDB¥vs8* ディレクトリにあります。CustDB アプリケーションを構築して実行するには、レッスン冒頭で説明している手順に従ってください。

Ultra Light for Embedded SQL

Embedded Visual C++ 用のプロジェクト・ファイルは、Embedded Visual C++ のバージョンに応じて、*samples-dir¥UltraLite¥CustDB¥EVC¥* ディレクトリまたは *samples-dir¥UltraLite¥CustDB¥EVC40¥* ディレクトリにあります。

Windows CE 用の Embedded SQL CustDB アプリケーションを Embedded Visual C++ を使用して構築する方法の詳細については、「CustDB サンプル・アプリケーションの構築」 『Ultra Light - C/C++ プログラミング』を参照してください。

Ultra Light for AppForge

CustDB プロジェクト・ファイルは、AppForge CrossFire と MobileVB 用に用意されています。どちらのプロジェクト・ファイル内にも、ファイルのサブセットがプラットフォーム別に存在しています。これにより、同じプロジェクト・ファイルを使用して CustDB アプリケーションをさまざまなプラットフォーム用に構築できます。どちらの開発環境にも *readme.txt* ファイルが用意されています。これらのファイルには、プロジェクトのロードやサポートされているプラットフォームへの配備の手順に関する開発環境固有の注意点が説明されています。

- **CrossFire**　プロジェクト・ファイルは、*samples-dir¥UltraLiteForAppForge¥CF_CustDB* ディレクトリにあります。

  CustDB アプリケーションを Palm OS、Windows CE、および Symbian OS 用に構築する手順については、*samples-dir¥UltraLiteForAppForge¥CF_CustDB¥readme.txt* を参照してください。

- **MobileVB**　プロジェクト・ファイルは、*samples-dir¥UltraLiteForAppForge¥MVB_CustDB* ディレクトリにあります。

  CustDB アプリケーションを Palm OS と Windows CE 用に構築する手順については、*samples-dir¥UltraLiteForAppForge¥MVB_CustDB¥readme.txt* を参照してください。

Ultra Light.NET

Visual Studio.NET 用のプロジェクト・ファイルは *samples-dir¥UltraLite.NET¥CustDB* ディレクトリにあります。

Visual Studio.NET を使用して CustDB アプリケーションを Windows CE 用に構築する手順については、*samples-dir¥UltraLite.NET¥CustDB¥ce¥ReadMe.html* を参照してください。

Windows デスクトップの配備ディレクトリ情報と、追加 Ultra Light.NET サンプルのダウンロード先に関する情報については、*samples-dir¥UltraLite.NET¥CustDB¥Desktop¥ReadMe.html* を参照してください。

Ultra Light for M-Business Anywhere

M-Business Anywhere 用のプロジェクト・ファイルは *samples-dir¥UltraLiteForMBusinessAnywhere¥CustDB* ディレクトリにあります。

M-Business Anywhere を使用して CustDB アプリケーションを構築する手順の詳細については、「Ultra Light for M-Business Anywhere クイック・スタート」『Ultra Light - M-Business Anywhere プログラミング』を参照してください。Windows CE、Windows、および Palm OS に適用される手順が用意されています。

レッスン 2：Ultra Light リモート・データベースへのログインとデータ移植

次の手順では、サンプルの Ultra Light クライアント・アプリケーションとサンプルの Mobile Link サーバを起動し、Ultra Light CustDB サンプル・リモート・データベースを同期して SQL Anywhere 統合データベースから初期のデータ・セットを取得することで Ultra Light リモート・データベースを移植します。ここでは、サンプル・アプリケーションは Mobile Link サーバと同じデスクトップ・コンピュータ上で動作しています。ただし、クライアント・アプリケーションをデバイスに配備しても結果は同じです。

ダウンロードするデータは、アプリケーションの起動時に入力したユーザ ID によって異なります。デフォルトでは、ユーザ ID 50 を使用して Ultra Light にログインします。

♦ **サンプル・アプリケーションを起動して同期するには、次の手順に従います。**

1. Mobile Link サーバのサンプルを起動します。

   コマンド・プロンプトで次のコマンドを入力します。

   ```
   mlsrv10 -c "DSN=SQL Anywhere 10 CustDB" -zu+ -vcrs
   ```

   または、[スタート] – [プログラム] – [SQL Anywhere 10] – [Mobile Link] – [Mobile Link サーバのサンプル] を選択します。

   サーバが起動したら、コンソール・ウィンドウが開き、Mobile Link サーバのステータスを示すメッセージが表示されます。

2. Ultra Light クライアント・サンプル・アプリケーションを起動します。

   [スタート] – [プログラム] – [SQL Anywhere 10] – [Ultra Light] – [Windows アプリケーションのサンプル] を選択します。

3. 従業員 ID を入力します。

   50 と入力し、[Enter] キーを押します。

   従業員 ID を入力したら、アプリケーションが同期されます。Mobile Link サーバのコンソール・ウィンドウに、同期が行われたことを示すメッセージが表示されます。

   デフォルトの同期スクリプトによって、ユーザ 50 がログインしたときにアプリケーションにダウンロードされる顧客、製品、注文のサブセットが決まります。ここでは、承認されていない注文のみがダウンロードされます。

4. アプリケーションにデータが含まれていることを確認します。

   アプリケーション・ウィンドウに会社名とサンプル注文情報が表示されています。

レッスン 3：CustDB クライアント・アプリケーションの使用

統合データベースとリモート・データベースの両方に ULOrder というテーブルがあります。統合データベースにはすべての注文 (承認済みと未承認) が含まれますが、Ultra Light リモートには、認証されたユーザに応じて、カラムのサブセットだけが表示されます。

テーブル内のカラムは、クライアント・アプリケーションでフィールドとして表示されます。注文を追加するときは、[Customer]、[Product]、[Quantity]、[Price]、[Discount] の各フィールドに入力する必要があります。[Status] や [Notes] などその他の情報を追加することもできます。タイムスタンプ・カラムを使用して、ローの同期が必要かどうかが判断されます。

参照

- 「CustDB データベース内のテーブル」 『Mobile Link - クイック・スタート』

注文のブラウズ

注文のブラウズは、Ultra Light クライアント・アプリケーションの各バージョンで同様の方法で行います。

注文をブラウズするときは、ローカルの Ultra Light データベースのデータをスクロールします。顧客はアルファベット順でソートされているので、簡単にリストをスクロールして顧客の名前を見つけることができます。

- **注文をブラウズするには、次の手順に従います。**

1. 顧客のリストを下にスクロールするには、[Next] をクリックします。
2. 顧客のリストを上にスクロールするには、[Previous] をクリックします。

注文の追加

注文の追加は、Ultra Light クライアント・アプリケーションの各バージョンで同様の方法で行います。

これで、ローカルの Ultra Light データベースでデータが修正されました。このデータは、同期が行われるまで統合データベースとは共有されません。

- **注文情報を追加するには、次の手順に従います。**

1. 新しい注文を作成します。

   [Order] – [New] を選択します。

   [Add New Order] ダイアログが表示されます。
2. 統合データベースからダウンロードされたリストから顧客を選択します。

[Customer] ドロップダウン・リストから [**Basements R Us**] を選択します。この顧客には現在注文がありません。

3. 統合データベースからダウンロードされたリストから製品を選択します。

   [Product] ドロップダウン・リストから [**Screwmaster Drill**] を選択します。この製品の価格は [Price] フィールドに自動的に設定されます。

4. 数量と値引き情報を入力します。

   [**Quantity**] フィールドに **20** と入力し、[**Discount**] フィールドに **5** (パーセント) と入力します。

5. [Enter] キーを押して、ULOrder テーブルのローとしてこの注文をリモート・データベースに追加します。

注文のステータスの変更

ユーザ ID 50 として認証されたユーザはマネージャで、販売担当者と同じ操作に加えて、注文を承認または拒否する権限があります。注文を承認または拒否するときには、注文のステータスを変更し、販売担当者へのメモを追加します。ただし、統合データベースのデータは、同期が行われるまで変更されません。

♦ **注文の承認、拒否、削除を行うには、次の手順に従います。**

1. **Apple Street Builders** からの注文を承認します。

   a. [Previous] をクリックしてこの顧客を探します。

   b. [Approve] をクリックして、注文を承認します。

   c. [Approve Order] ダイアログで、[Note] ドロップダウン・リストから [**Good work!**] を選択します。

   d. [Enter] を押します。

      注文情報には [Approved] のステータスが表示されます。

2. Art's Renovations からの注文を拒否します。

   a. リストの次の項目である Art's Renovations からの注文を表示します。

   b. [Deny] をクリックして、注文を拒否します。

   c. [Deny Order] ダイアログで、[Note] ドロップダウン・リストから [**Discount is too high**] を選択します。

   d. [Enter] を押します。

   注文情報には [Denied] のステータスが表示されます。

3. Awnings R Us からの注文を削除します。

a. リストの次の項目である Awnings R Us からの注文を表示します。

b. [Order] − [Delete] を選択してこの注文を削除します。

削除の確認を求めるメッセージが表示されます。[はい] をクリックします。

注文に削除済みのマークが付けられます。ただし、変更内容を統合データベースと同期するまでは、現在のデータが Ultra Light リモートに残ります。

レッスン 4：CustDB 統合データベースとの同期

同期するには、Mobile Link サーバが実行されている必要があります。Mobile Link サーバを停止した場合は、サーバを再起動してください。「レッスン 2：Ultra Light リモート・データベースへのログインとデータ移植」 123 ページを参照してください。

このサンプル・アプリケーションで同期を行うと、承認された注文情報はデータベースから削除されます。

♦ **Ultra Light リモートを同期するには、次の手順に従います。**

1. [File] – [Synchronize] を選択してデータを同期します。
2. 同期されたことを確認します。
   - リモート・データベースでは、**Apple Street Builders** の承認済み注文が削除されたことを確認することで、必要なトランザクションがすべて完了したことを確認できます。このエントリがないことを確認するには、注文をブラウズします。
   - 統合データベースでは、データを確認して、必要な処理がすべて行われたことを確認できます。

統合データベースでの同期の確認

Interactive SQL または Sybase Central を使用して統合データベースに接続し、変更内容が同期されたことを確認します。

♦ **同期を確認するには、次の手順に従います (Interactive SQL の場合)。**

1. Interactive SQL から統合データベースに接続します。
   a. [スタート] – [プログラム] – [SQL Anywhere 10] – [SQL Anywhere] – [Interactive SQL] を選択します。

      Interactive SQL の [接続] ダイアログが表示されます。

   b. [ODBC データ・ソース名] を選択し、ドロップダウン・リストから [SQL Anywhere 10 CustDB] を選択します。
2. 承認や拒否を行った注文情報のステータスが変更されていることを確認します。

   承認や拒否が同期されたことを確認するため、次の文を発行します。

   ```
   SELECT order_id, status
   FROM ULOrder
   WHERE status IS NOT NULL
   ```

   この文の結果として、注文 5100 は承認されており、5101 は拒否されたことがわかります。
3. 削除された注文情報がなくなっていることを確認します。

削除された注文情報の order_id は 5102 です。次のクエリを実行してもローは返りません。これは、その注文情報がシステムから削除されたことを示します。

```
SELECT *
FROM ULOrder
WHERE order_id = 5102
```

♦ **同期を確認するには、次の手順に従います (Sybase Central の場合)。**

1. Sybase Central を起動します。

   [プログラム] − [Sybase] − [SQL Anywhere 10] − [Sybase Central] を選択します。

2. CustDB 統合データベースに接続します。

   a. [接続] − [SQL Anywhere 10 に接続] を選択します。

      [接続] ダイアログが表示されます。

   b. [ODBC データ・ソース名] を選択し、ドロップダウン・リストから [SQL Anywhere 10 CustDB] を選択します。

3. ULOrder テーブルで変更内容を確認します。

   a. [テーブル] をダブルクリックしてすべてのテーブルを表示します。

   b. ULOrder テーブルをダブルクリックします。

   c. [データ] タブをクリックし、次のことを確認します。

      ♦ 注文 5100 が承認されており、5101 が拒否されている

      ♦ 注文 5102 が削除されている

レッスン 5：Mobile Link 同期スクリプトのブラウズ

CustDB の同期論理は、Mobile Link 同期スクリプトとして統合データベースに格納されています。同期論理によって、統合データベースでダウンロードまたはアップロードの対象となる容量を特定できます。タイムスタンプベースの同期やスナップショット同期などの方法で、テーブル全体またはテーブルの一部 (ローまたはカラムのサブセット) をダウンロードできます。

CustDB への同期の実装の詳細については、「同期の設計」 『Mobile Link - クイック・スタート』を参照してください。

参照

- 「同期スクリプトの作成」 『Mobile Link - サーバ管理』
- 「Ultra Light クライアントの概要」 『Mobile Link - クライアント管理』

同期スクリプトのブラウズ

Sybase Central を使用すると、テーブル、ユーザ、パブリケーションの他に、統合データベースに格納されている同期スクリプトをブラウズできます。Sybase Central は、データベースにこれらのスクリプトを追加するためのプライマリ・ツールです。

custdb.sql ファイルは、ml_add_connection_script または ml_add_table_script を呼び出して、各同期スクリプトを統合データベースに追加します。接続スクリプトは、特定のテーブルに関連付けられていない高いレベルのイベントを制御します。これらのイベントは、各同期の処理中に必要な全般的なタスクを実行するときに使用します。テーブル・スクリプトによって、ローのアップロードの開始や終了、競合の解決、ダウンロードするローの選択など、特定のテーブルの同期に関する特定のイベントでのアクションを実行できます。

CustDB で使用されている同期論理の詳細については、「同期論理のソース・コード」 『Mobile Link - クイック・スタート』を参照してください。

♦ **同期スクリプトをブラウズするには、次の手順に従います。**

1. Sybase Central を起動します。

   [プログラム] – [SQL Anywhere 10] – [Sybase Central] を選択します。

2. CustDB 統合データベースに接続します。

   a. [接続] – [Mobile Link 10 に接続] を選択します。

      [接続] ダイアログが表示されます。

   b. [ODBC データ・ソース名] を選択し、[SQL Anywhere 10 CustDB] を選択します。

      [OK] をクリックします。

3. [接続スクリプト] フォルダを開きます。

   右ウィンドウ枠には、同期スクリプトと、それらが関連付けられているイベント・セットがリストされています。Mobile Link サーバが同期処理を実行すると、一連のイベントがトリ

ガされます。このときに、イベントに関連付けられた同期スクリプトが実行されます。同期スクリプトを作成し、同期イベントを割り当てることによって、同期の際に実行されるアクションを制御できます。

4. [同期テーブル] フォルダを開き、[ULCustomer] テーブル・フォルダを開きます。

   右ウィンドウ枠には、このテーブル固有のスクリプト・セットと、それに対応するイベントがリストされています。これらのスクリプトは、ULCustomer テーブルのデータがリモート・データベースと同期される方法を制御します。

参照

- 「接続スクリプト」『Mobile Link - サーバ管理』
- 「テーブル・スクリプト」『Mobile Link - サーバ管理』

次の作業

サポートされている各インタフェースを使用した独自のアプリケーションを構築するためのチュートリアルが用意されています。詳細については、次の項を参照してください。

- **Ultra Light C++** 「チュートリアル：C++ API を使用したアプリケーションの構築」『Ultra Light - C/C++ プログラミング』.
- **Ultra Light Embedded SQL** 「チュートリアル：Embedded SQL を使用したアプリケーションの構築」『Ultra Light - C/C++ プログラミング』.
- **Ultra Light for AppForge** 「チュートリアル：AppForge MobileVB のサンプル・アプリケーション」『Ultra Light - AppForge プログラミング』.
- **Ultra Light.NET** 「チュートリアル：Ultra Light.NET アプリケーションの構築」『Ultra Light - .NET プログラミング』.
- **Ultra Light for M-Business Anywhere** 「チュートリアル：M-Business Anywhere 用のサンプル・アプリケーション」『Ultra Light - M-Business Anywhere プログラミング』

パート IV. Ultra Light データベースのリファレンス

パート IV では、Ultra Light データベースのプロパティ、オプション、接続パラメータ、ユーティリティのリファレンスです。

第 8 章

Ultra Light データベース設定のリファレンス

目次

Ultra Light の設定可能なプロパティ

Ultra Light データベースを新規作成する際、設定にプロパティを使用します。データベース・プロパティは変更できません。プロパティを変更する唯一の方法は、データベースを作成し直すことです。

ただし、設定可能なプロパティはいつでも表示またはアクセスできます。

参照

- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ
- 「Ultra Light の設定可能なオプション」 160 ページ
- 「Ultra Light の読み込み専用プロパティ」 165 ページ

Ultra Light case プロパティ

Ultra Light データベースの文字列を比較するときに大文字と小文字を区別するかどうかを設定します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

コンテキスト	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o case=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースの照合と文字セット] ページで [文字列の比較で大文字と小文字を区別する] オプションを選択します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

Ignore、Respect

デフォルト

Ignore

備考

既存のデータベースの大文字と小文字を区別するかどうかは変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

データの大文字と小文字を区別するかしないかは、テーブル、インデックスなどに反映されます。Ultra Light データベースは、デフォルトでは、データは常に入力されたとおりの大文字と小文字で保持されていますが、比較において大文字と小文字を区別しません。データベースの大文字と小文字の区別の設定に関係なく、パスワードは常に大文字と小文字が区別されます。

参照

- 「Ultra Light での大文字と小文字の区別の考慮事項」 65 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light checksum_level プロパティ

データベースのチェックサム検証のレベルを設定します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o checksum_level=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースの記憶領域設定] ページで [データベース・ページのチェックサム・レベル] オプションを選択します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

次の 3 つのチェックサム検証レベルを設定できます。

値	定義
0	データベース・ページにチェックサムを追加しない。
1	重要なデータベース・ページ (インデックスや同期ステータスのページ) にチェックサムを追加し、ロー・ページには追加しない。
2	すべてのデータベース・ページにチェックサムを追加する。

デフォルト

0

備考

チェックサムは、オフラインの破損を検出するために使用します。チェックサムによって、重要なページの破損が原因で他のデータが破損するのを防ぐことができます。チェックサムが一致しなかった場合、データベースでページがロードされるときに、Ultra Light によってデータベースが停止され、致命的なエラーがレポートされます。このエラーは解決できません。Ultra Light データベースを再作成し、データベースのエラーを iAnywhere にレポートしてください。

チェックサムが有効になっている Ultra Light データベースをアンロードしてから再ロードした場合、チェックサムのレベルは保持され、リストアされます。

参照

- 「チェックサムを使用したページの整合性の確認」 69 ページ
- 「単一のトランザクションまたはグループ化されたトランザクションのフラッシュ」 47 ページ
- 「サポートされている終了コード」 200 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「createDatabase メソッド」『Ultra Light - M-Business Anywhere プログラミング』

Ultra Light collation プロパティ

データベースが使用する照合を返します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o collation=***name*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースの照合と文字セット] ページで、デフォルトの照合 (1252Latin1) を選択するか、表示されるリストから別の照合を選択します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

Ultra Light でサポートされている任意の照合名。Ultra Light でサポートされている照合のリストを表示するには、データベース作成ユーティリティで -l オプションを使用します。[データベース作成] ウィザードでは、サポートされている照合は [新しいデータベースの照合と文字セット] ページに表示されます。

参照

- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- 「Ultra Light での文字の考慮事項」 63 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light date_format プロパティ

日付がデータベースから取り出されるときのデフォルトの文字列フォーマットを設定します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o date_format=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [日付形式] オプションを設定します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

次のいずれかの記号を使用する文字列を指定できます。

記号	説明
yy	2 桁の年度
yyyy	4 桁の年度
mm	2 桁の月、または 2 桁の分 (*hh*:*mm* のように、コロンの後ろに続く場合)
mmm[*m*…]	月を示す略式文字−"m" の数と同数の文字。大文字の M の場合、出力も大文字になります。
d	1 桁の曜日 (0 = 日曜日、6 = 土曜日)
dd	2 桁の指定月の日。先頭のゼロは必要ありません。
ddd[*d*…]	曜日を示す略式文字。大文字の D の場合、出力も大文字になります。
hh	2 桁の時間。先頭のゼロは必要ありません。
nn	2 桁の分。先頭のゼロは必要ありません。
ss[.*ss*..]	秒と秒の一部
aa	12 時間表記。正午前の時刻は AM で表します。
pp	12 時間表記。正午後の時刻は PM で表します。
jjj	指定年の日 (1 ー 366)

デフォルト

YYYY-MM-DD (ISO の日付フォーマット仕様に対応)

備考

既存のデータベースの日付フォーマットは変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

指定可能な値は、上の表に示す記号から構成されます。各記号は、フォーマットされる日付のデータで置き換えられます。

略式文字については、指定した文字数が数えられ、必要な場合は、A.M. または P.M. のインジケータ (ローカライズ可能) が、指定した文字数に対応するバイト数までトランケートされます。

出力の大文字と小文字の制御　文字データを表す記号 (*mmm* など) では、出力の文字を次のように制御できます。

- 記号を大文字で入力すると、フォーマットが大文字で表記されます。たとえば MMM と入力すると、JAN と表記されます。
- 記号を小文字で入力すると、フォーマットが小文字で表記されます。たとえば mmm と入力すると、jan と表記されます。
- 大文字と小文字を混ぜて入力すると、使用される言語に適切な文字が Ultra Light により選択されます。たとえば、Mmm と入力すると、英語では May、フランス語では Mai と表記されます。

0 埋め込みの制御　数値データを表す記号では、記号に大文字を使用するか小文字を使用するかで 0 埋め込みを制御できます。

- 記号をすべて大文字または小文字 (MM や mm など) で入力すると、0 埋め込みが行われます。たとえば、yyyy/mm/dd と入力すると、2002/01/01 と表記されます。
- 大文字と小文字を混ぜて入力すると (Mm など)、0 の埋め込みは行われません。たとえば、yyyy/Mm/Dd と入力すると、2002/1/1 と表記されます。

例

次の表は、date_format の設定と、2001 年 5 月 21 日 (木) に実行された SELECT CURRENT DATE 文からの出力を示します。

使用した date_format の構文	返された結果
yyyy/mm/dd/ddd	2001/05/21/thu
jjj	141
mmm yyyy	may 2001
mm-yyyy	05-2001

参照

- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ

- 「Ultra Light での日付の考慮事項」 66 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light date_order プロパティ

年、月、日の各日付の単位の順序を解釈する方法を制御します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o date_order=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [日付順] オプションを設定します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

MDY、YMD、DMY

デフォルト

YMD (ISO 日付フォーマット仕様に対応)

備考

既存のデータベースの日付順は変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

参照

- 「Ultra Light での日付の考慮事項」 66 ページ

- 「Ultra Light date_format プロパティ」 139 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

例

値によって、日付 10/11/12 の変換方法が異なります。

使用される構文	変換
MDY	1912 年 10 月 11 日
YMD	1910 年 11 月 12 日
DMY	1912 年 11 月 10 日

Ultra Light fips プロパティ

新しいデータベースで AES_FIPS 暗号化と AES 暗号化のどちらを使用するかを制御します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o fips=***boolean* ulcreate の接続文字列に **KEY** 接続パラメータを含める必要があります。これによって FIPS 暗号化に使用する暗号化キーが設定されます。
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースの記憶領域設定] ページで [AES FIPS] アルゴリズムのオプションを選択して、データベースを強力な暗号化で暗号化することを指定します。暗号化キーを設定し、確認する必要もあります。

方法	操作
クライアント・アプリケーションから データベース作成メソッドを使用します。	◆ Palm OS では、ULEnableFipsStrongEncryption メソッドを使用します。 ◆ その他すべてのプラットフォームでは、データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

データベースで AES_FIPS を使用して暗号化される場合は true、AES を使用して暗号化される場合は false。

デフォルト

0 (データベースは AES を使用して暗号化されます)

備考

すべてのブール値を指定できます。たとえば、true と false、yes と no、1 と 0 などです。

暗号化キーを変更するには、新しい暗号化キーを Connection オブジェクトで指定します。適切なメソッドを呼び出す前に、アプリケーションで既存の暗号化キーを使用して、暗号化されたデータベースに接続する必要があります。ただし、使用する暗号化タイプは変更できません。目的の fips または obfuscate プロパティを使用してデータベースを作成し直す必要があります。

FIPS 暗号化に使用するキーを設定したら、このデータベースに接続するユーザが、接続のたびにこのキーを指定する必要があります。

FIPS の配備

FIPS 対応のデータベースを配備するには、プラットフォームに適切なライブラリをすべてコピーします(例：ulfips.dll)。「Ultra Light での AES FIPS データベース暗号化の設定 」 71 ページを参照してください。

参照

- 「強力な暗号化」 『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light でのセキュリティの考慮事項」 70 ページ
- 「Ultra Light DBKEY 接続パラメータ」 188 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- 「Palm OS の暗号化キーの保存、取り出し、クリア」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「ULChangeEncryptionKey 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「ChangeEncryptionKey メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』

- Ultra Light.NET：「ChangeEncryptionKey メソッド」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「createDatabase メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「ChangeEncryptionKey メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light max_hash_size プロパティ

デフォルトの最大インデックス・ハッシュ・サイズをバイト単位で設定します。Ultra Light では、この最大値を上限として、カラムのデータ型に必要なバイト数だけが使用されます。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o max_hash_size=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースの記憶領域設定] ページで [インデックスの最大ハッシュ・サイズ] オプションを選択します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

0 ー 32 バイト

デフォルト

4 バイト

備考

デフォルトのハッシュ・サイズは、インデックスを作成するときにサイズを設定しなかった場合にのみ使用されます。

デフォルトのハッシュ・サイズを 0 に設定すると、ローの値はハッシュされません。

インデックスの作成後にインデックスのハッシュ・サイズを変更することはできません。ただし、Sybase Central で Ultra Light の [インデックス作成] ウィザードを使用して、または CREATE INDEX 文または CREATE TABLE 文で WITH MAX SIZE 句を使用して新しいインデックスを作成するときにデフォルトを変更できます。

カラムを DOUBLE、FLOAT、または REAL のデータ型として宣言した場合は、ハッシュは使用されません。ハッシュ・サイズは常に無視されます。

参照

- 「Ultra Light でのインデックス・パフォーマンスの考慮事項」 64 ページ
- 「Ultra Light クエリ・パフォーマンスの最適化 」 38 ページ
- 「Ultra Light のインデックスの操作」 100 ページ
- 「最適なハッシュ・サイズの選択」 42 ページ
- 「Ultra Light CREATE INDEX 文」 387 ページ
- 「Ultra Light CREATE TABLE 文」 390 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light nearest_century プロパティ

文字列から日付への変換で、2 桁の年の解釈を制御します。文字列から日付またはタイムスタンプに変換するときに使用します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o nearest_century=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [基準年] オプションを設定します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

整数 (0 ー 100)

デフォルト

50

備考

既存のデータベースの基準年は変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

nearest_century 設定は、ロールオーバ・ポイントとして動作する数値です。この値より小さい 2 桁の年は 20*yy* に変換され、この値以上の年は 19*yy* に変換されます。

参照

- 「Ultra Light での基準年の変換の考慮事項」 67 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light obfuscate プロパティ

データベース内のデータの難読化を制御します。難読化は単純暗号化です。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o obfuscate=***boolean*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースの記憶領域設定] ページで [簡易暗号化を使用 (obfuscation)] オプションを選択してデータベースを暗号化することを指定します。

方法	操作
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

すべてのブール値を指定できます。たとえば、true と false、yes と no、1 と 0 などです。

デフォルト

0 (データベースの難読化なし)

備考

単純暗号化は、難読化と同じです。これにより第三者は、ディスク・ユーティリティを使用してファイルを表示し、データベースのデータを解読することが困難になります。単純暗号化では、データベースの暗号化のためのキーは不要です。

正しい暗号化キーがなかった場合にデータベースにアクセスできないようにするには、データベースに強力な暗号化を設定する必要があります。

参照

- 「単純暗号化」『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light でのセキュリティの考慮事項」 70 ページ
- 「Ultra Light DBKEY 接続パラメータ」 188 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「createDatabase メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light page_size プロパティ

データベース・ページ・サイズを定義します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o page_size=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースの記憶領域設定] ページで、この後に示す指定可能な値に対応する適切なバイト値を選択します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

1 K、2 K、4 K、8 K、16 K

デフォルト

4 K

備考

既存のデータベースのページ・サイズは変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

k または K を使用してキロバイト単位を示します。前述の指定可能な値以外の値を使用すると、サイズはその値より大きい次の値に変更されます。単位を指定しない場合、デフォルトはバイトです。

プラットフォームの動的メモリの制約が大きい場合は、ページ・サイズを小さくして、同期メモリ要件による影響を低減させることを検討してください。

参照

- 「Ultra Light のプラットフォーム別の最適化方法」 49 ページ
- 「ローのパックとテーブル定義 」 90 ページ
- 「Ultra Light でのページ・サイズの考慮事項」 68 ページ
- 「Ultra Light CACHE_SIZE 接続パラメータ」 170 ページ
- 「Ultra Light RESERVE_SIZE 接続パラメータ」 194 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』

- Ultra Light for M-Business Anywhere：「createDatabase メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

例

データベースのページ・サイズを 8 KB に設定するには、次のように値を設定します。

page_size=8k

または

page_size=8192

Ultra Light precision プロパティ

計算結果が小数の場合の最大桁数を指定します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o precision=*value***
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [精度] オプションを設定します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

整数 (1 ー 127)

デフォルト

30

備考

精度は、小数点の左右の合計桁数です。このプロパティと scale プロパティを使用して、計算結果が最大 precision にトランケートされる場合の、小数点以下の最小桁数を指定します。

掛け算、割り算、足し算、引き算、集合関数はすべて結果が最大精度を超えることができます。たとえば、DECIMAL(8,2) と DECIMAL(9,2) を掛けると、結果には DECIMAL(17,4) が必要です。precision が 15 の場合、15 桁のみが結果に保持されます。scale が 4 の場合、結果は DECIMAL(15,4) になります。scale が 2 の場合、結果は DECIMAL(15.2) になります。どちらの場合も、オーバフローが発生する可能性があります。

既存のデータベースの精度は変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

統合データベースとして Oracle データベースを使用している場合、すべての Ultra Light リモートと Oracle 統合データベースが精度として同じ値を使用する必要があります。

参照

- 「Ultra Light での小数点の位置の考慮事項」 68 ページ
- 「Ultra Light scale プロパティ」 151 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light scale プロパティ

計算結果が最大 precision にトランケートされる場合の、小数点以下の最小桁数を指定します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o scale=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [位取り] オプションを設定します。

方法	操作
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

precision データベース・オプションに指定した値より小さい 0 ー 127 の整数

デフォルト

6

備考

既存のデータベースの位取りは変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

scale プロパティは、小数点以下の最小桁数を指定します。掛け算、割り算、足し算、引き算、集合関数はすべて結果が最大精度を超えることができます。

このプロパティは、計算結果の全体長を指定する precision プロパティと同時に使用します。

参照

- 「Ultra Light での小数点の位置の考慮事項」 68 ページ
- 「Ultra Light precision プロパティ」 150 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

例

DECIMAL(8,2) と DECIMAL(9,2) を掛けると、結果には DECIMAL(17,4) が必要です。precision が 15 の場合、15 桁のみが結果に保持されます。scale が 4 の場合、結果は DECIMAL(15,4) になります。scale が 2 の場合、結果は DECIMAL(15.2) になります。どちらの場合も、オーバフローが発生する可能性があります。

Ultra Light time_format プロパティ

データベースから取り出した時刻のフォーマットを設定します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o time_format=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [時間形式] オプションを設定します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

次のいずれかの記号を使用する文字列を指定できます。

シンボル	説明
hh	2 桁の時間 (24 時間表記)。
nn	2 桁の分。
mm	コロンの後の場合は、2 桁の分 (*hh*:*mm* など)。
ss[.s…]	2 桁の秒と、オプションで小数。

デフォルト

HH:*NN*:*SS.SSS*

備考

既存のデータベースの時間フォーマットは変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

各記号は、フォーマットしようとする日付のデータで置き換えられます。

0 埋め込みの制御　記号に大文字を使用するか小文字を使用するかで 0 埋め込みを制御できます。

- 記号をすべて大文字または小文字 (HH や hh など) で入力すると、0 埋め込みが行われます。たとえば HH:NN:SS と入力すると、01:01:01 と表記されます。

- 大文字と小文字を混ぜて入力すると (Hh など)、0 埋め込みは行われません。たとえば Hh:Nn:Ss と入力すると、1:1:1 と表記されます。

参照

- 「Ultra Light での時刻の考慮事項」 66 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

例

3:30 PM にトランザクションが実行された場合、デフォルトの time_format 構文 *HH*:*NN*:*SS*.*SSS* では、結果は次のようになります。

15:30:55.0

Ultra Light timestamp_format プロパティ

データベースから取り出したタイムスタンプのフォーマットを設定します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **-o timestamp_format=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [タイムスタンプ形式] オプションを設定します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

次のいずれかの記号を使用する文字列を指定できます。

シンボル	説明
yy	2 桁の年度
yyyy	4 桁の年度
mm	2 桁の月、または 2 桁の分 (*hh*:*mm* のように、コロンの後ろに続く場合)
mmm[*m*…]	月を示す略式文字–"m" の数と同数の文字。大文字の M の場合、出力も大文字になります。
d	1 桁の曜日 (0 = 日曜日、6 = 土曜日)
dd	2 桁の指定月の日。先頭のゼロは必要ありません。
ddd[*d*…]	曜日を示す略式文字。大文字の D の場合、出力も大文字になります。
hh	2 桁の時間。先頭のゼロは必要ありません。
nn	2 桁の分。先頭のゼロは必要ありません。
ss[.*ss*..]	秒と秒の一部
aa	12 時間表記。正午前の時刻は AM で表します。
pp	12 時間表記。正午後の時刻は PM で表します。
jjj	指定年の日 (1 ー 366)

デフォルト

YYYY-MM-DD HH:*NN*:*SS.SSS*

備考

既存のデータベースのタイムスタンプ・フォーマットは変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

指定可能な値は、上の表に示す記号から構成されます。各記号は、フォーマットされる日付のデータで置き換えられます。

略式文字については、指定した文字数が数えられ、必要な場合は、A.M. または P.M. のインジケータ (ローカライズ可能) が、指定した文字数に対応するバイト数までトランケートされます。

出力の大文字と小文字の制御　文字データを表す記号 (*mmm* など) では、出力の文字を次のように制御できます。

- 記号をすべて大文字で入力すると、フォーマットはすべて大文字で表記されます。たとえば、MMM と入力すると、JAN と表記されます。

- 記号をすべて小文字で入力すると、フォーマットはすべて小文字で表記されます。たとえば、mmm と入力すると、jan と表記されます。
- 大文字と小文字を混ぜて入力すると、使用される言語に適切な文字が Ultra Light により選択されます。たとえば、Mmm と入力すると、英語では May、フランス語では Mai と表記されます。

0 埋め込みの制御　数値データを表す記号では、記号に大文字を使用するか小文字を使用するかで 0 埋め込みを制御できます。

- 記号をすべて大文字または小文字 (MM や mm など) で入力すると、0 埋め込みが行われます。たとえば、yyyy/mm/dd と入力すると、2002/01/01 と表記されます。
- 大文字と小文字を混ぜて入力すると (Mm など)、0 の埋め込みは行われません。たとえば、yyyy/Mm/Dd と入力すると、2002/1/1 と表記されます。

参照

- 「Ultra Light でのタイムスタンプの考慮事項」 66 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

例

2006 年 5 月 12 日 (金) の 3:30 PM にトランザクションが実行された場合、デフォルトの timestamp_format 構文 *YYYY-MM-DD HH*:*NN*:*SS.SSS* では、結果は次のようになります。

2006-05-12 15:30:55.0

Ultra Light timestamp_increment プロパティ

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。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o timestamp_increment=***value*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [タイムスタンプ・インクリメント] オプションを設定します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

1 ー 60,000,000,000 マイクロ秒

デフォルト

1 マイクロ秒

備考

既存のデータベースのタイムスタンプ・インクリメントは変更できません。変更する必要がある場合は、新しいデータベースを作成してください。

DEFAULT TIMESTAMP カラムがプライマリ・キーまたはロー識別子として使用されている場合、このインクリメントが役に立ちます。

参照

- 「Ultra Light でのタイムスタンプの考慮事項」 66 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

例

'2000/12/05 10:50:53:700' などの値を格納するには、このプロパティを 100000 に設定します。この値に設定すると、秒の部分は小数点以下 1 桁の後でトランケートされます。

Ultra Light utf8_encoding プロパティ

ユニコード用の 8 ビット・マルチバイト・エンコードである UTF-8 フォーマットでデータをエンコードします。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから データベースを作成する任意のユーティリティを使用します。	プロパティの次の構文を使用します。 **- o utf8_encoding=***boolean*
Sybase Central から データベースを作成する任意のウィザードを使用します。	[新しいデータベースのオプション] ページで [UTF-8] オプションを設定します。
クライアント・アプリケーションから データベース作成メソッドを使用します。	データベース・マネージャ・クラスに対する API のデータベース作成メソッドの作成パラメータの 1 つとしてこのプロパティを設定します。

指定可能な値

すべてのブール値を指定できます。たとえば、true と false、yes と no、1 と 0 などです。

デフォルト

0 (データベースの UTF-8 エンコードなし)

備考

UTF-8 文字は 1 ー 4 バイトで表されます。他のマルチバイト照合の場合、1 または 2 バイトが使用されます。すべてのマルチバイト照合で、2 バイト以上の文字はアルファベットであると見なされます。したがって、二重引用符を使用しないでこれらの文字を識別子で使用できます。

データベースを UTF-8 でエンコードすると、Ultra Light では UTF8BIN 照合を使用して文字がソートされます。UTF8BIN 文字セットは特定の言語に固有ではないので、この文字セットに特定のコード・ページは関連付けられていません。したがって、複数の言語のデータを同じ統合データベースに同期できます。UTF-8 でエンコードされている文字を、ユニコードがサポートされていない統合テーブルに同期しようとすると、ユーザ・エラーがレポートされます。

参照

- 「Ultra Light での文字セットのエンコードに関するプラットフォーム要件」 64 ページ

- 「Ultra Light での文字の考慮事項」 63 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light データベース初期化ユーティリティ (ulinit)」 220 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- Ultra Light for Embedded SQL：「ULCreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C++：「CreateDatabase 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ULDatabaseManager クラス」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「ULDatabaseManager メンバ」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「 createDatabase メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ

Ultra Light の設定可能なオプション

いつでも設定または変更できるデータベース・プロパティの設定にはオプションが使用されます。Ultra Light では、データベース・オプションとプロパティを同じ名前にすることでこれを実現しています。Ultra Light では、オプションは永続的または一時的です。一時的なオプションは、データベースの実行中のみ保持されます。永続的なオプションはデータベースの sysuldata システム・テーブルに格納されます。

設定可能なオプションは、それに対応するデータベース・プロパティを変更することで、いつでも表示したりアクセスしたりできます。

参照

- 「sysuldata システム・テーブル」 247 ページ
- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ
- 「プロパティへのアクセス」 61 ページ
- 「Ultra Light の設定可能なプロパティ」 136 ページ
- 「Ultra Light の読み込み専用プロパティ」 165 ページ

Ultra Light commit_flush_count オプション [テンポラリ]

一時的にコミット・カウント・スレッショルドを設定します。このスレッショルドに達すると、コミット・フラッシュが実行されます。

適用対象

COMMIT_FLUSH=grouped として設定されている接続

設定方法

方法	操作
コマンド・ラインから ulinfo ユーティリティを使用します。	次のユーティリティ・オプションを使用します。 **-x**=*value*
SQL を使用 SET OPTION 文を使用します。	次の SQL 構文を使用します。 **SET OPTION commit_flush_count**=*value*
クライアント・アプリケーションから データベース・オプション設定メソッドを使用します。	このスレッショルド値の設定方法は、使用するプログラミング・インタフェースによって異なります。

指定可能な値

整数

備考

トランザクション・カウントを無効にする場合は、0 を使用します。これは、フラッシュがいつトリガされるかという意味において、コミット数に制限がないことを意味します。

このオプションの効力があるのは、指定したデータベースの実行が停止されるまでです。必要がある場合は、データベースを開始するたびにこのオプションを設定する必要があります。

このオプションを **commit_flush_timeout** オプションとともに設定しており、接続パラメータに **COMMIT_FLUSH=grouped** が含まれている場合は、どちらかのスレッショルドに達するとフラッシュがトリガされます。フラッシュが実行されると、Ultra Light によってカウンタとタイマはどちらも 0 に戻されます。その後、どちらか一方がスレッショルドに達するまで、両方ともモニタリングされます。

参照

- 「Ultra Light commit_flush_timeout オプション [テンポラリ]」 161 ページ
- 「Ultra Light COMMIT_FLUSH 接続パラメータ」 172 ページ
- 「Ultra Light SET OPTION 文」 406 ページ
- Ultra Light for C/C++ :「SetDatabaseOption 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「ULSetDatabaseOptionString 関数」『Ultra Light - C/C++ プログラミング』

Ultra Light commit_flush_timeout オプション [テンポラリ]

一時的に時間間隔スレッショルドを設定します。このスレッショルドに達すると、グループ化されたコミット・フラッシュが実行されます。

設定方法

方法	操作
コマンド・ラインから ulinfo ユーティリティを使用します。	次のユーティリティ・オプションを使用します。 **-x**=*value*
SQL を使用 SET OPTION 文を使用します。	次の SQL 構文を使用します。 **SET OPTION commit_flush_timeout**=*value*
クライアント・アプリケーションから データベース・オプション設定メソッドを使用します。	このタイムアウト値の設定方法は、使用するプログラミング・インタフェースによって異なります。

指定可能な値

整数 (ミリ秒単位)

デフォルト

10000 ミリ秒 (10 秒)

備考

時間スレッショルドを無効にする場合は、0 を使用します。

このオプションの効力があるのは、指定したデータベースの実行が停止されるまでです。必要がある場合は、データベースを開始するたびにこのオプションを設定する必要があります。

このオプションを commit_flush_timeout オプションとともに設定し、COMMIT_FLUSH 接続パラメータを **grouped** に設定した場合は、どちらかのスレッショルドに達するとフラッシュがトリガされます。フラッシュが実行されると、Ultra Light によってカウンタとタイマはどちらも 0 に戻されます。その後、どちらか一方がスレッショルドに達するまで、両方ともモニタリングされます。

参照

- 「Ultra Light commit_flush_count オプション [テンポラリ]」 160 ページ
- 「Ultra Light COMMIT_FLUSH 接続パラメータ」 172 ページ
- 「Ultra Light SET OPTION 文」 406 ページ
- Ultra Light for C/C++：「SetDatabaseOption 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「ULSetDatabaseOptionString 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「プロパティ」 『Ultra Light - AppForge プログラミング』

Ultra Light global_database_id オプション

データベースの ID 番号を設定します。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから ulinfo ユーティリティを使用します。	次のユーティリティ・オプションを使用します。 **-g** *ID*
Sybase Central から データベース名を右クリックして[オプション] を選択して、[データベース・オプション] ダイアログを使用します。	global_database_ID オプションを選択し、オプションの表の下にある [値] フィールドに新しい文字列を入力します。
SQL を使用 SET OPTION 文を使用します。	次の SQL 構文を使用します。 **SET OPTION global_database_id=***ID*
クライアント・アプリケーションから データベース ID 設定メソッドを使用します。	この ID 番号の設定方法は、使用するプログラミング・インタフェースによって異なります。

指定可能な値

配備された各 Ultra Light アプリケーションのグローバル・データベース識別子には、ユニークな正の整数を設定してから、デフォルト値を割り当てます。各データベースは、これらの ID 番号によってユニークに識別されます。

デフォルト

特定のグローバル・オートインクリメント・カラムのデフォルト値の範囲は、$pn + 1$ ～ $p(n + 1)$ です。ここで、p はカラムの分割サイズを、n はグローバル・データベース ID 番号を示します。

備考

アプリケーションを配備するときには、Mobile Link サーバとの同期のために、各データベースに異なる ID 番号を割り当てる必要があります。

既存のデータベースのグローバル ID はいつでも変更できます。新しいデータベースを作成する必要はありません。

参照

- 「Ultra Light でのグローバル・データベース ID の考慮事項」 74 ページ
- 「Ultra Light 情報ユーティリティ (ulinfo)」 217 ページ
- 「Ultra Light での GLOBAL AUTOINCREMENT の使用」『Mobile Link - クライアント管理』
- Ultra Light for C/C++ :「SetDatabaseID 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「ULSetDatabaseID 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「プロパティ」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET :「DatabaseID プロパティ」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere :「 setDatabaseID メソッド」『Ultra Light - M-Business Anywhere プログラミング』

例

Ultra Light データベースのカラムを 3001 から 4000 までオートインクリメントするには、グローバル ID を 3 に設定します。

Ultra Light ml_remote_id オプション

Mobile Link による同期に使用される Ultra Light のユニークな識別子です。

設定方法

Ultra Light では、次の方法でこのプロパティを設定できます。

方法	操作
コマンド・ラインから ulinfo ユーティリティを使用します。	次のユーティリティ・オプションを使用します。 **-r** *ID*

方法	操作
Sybase Central から [データベース・オプション] ダイアログを使用します。	ml_remote_ID オプションを選択し、オプションの表の下にある [値] フィールド下に新しい文字列を入力します。
SQL を使用 SET OPTION 文を使用します。	次の SQL 構文を使用します。 **SET OPTION ml_remote_id=***ID*
クライアント・アプリケーションから データベース・オプション設定メソッドを使用します。	この ID 値の設定方法は、使用するプログラミング・インタフェースによって異なります。

指定可能な値

Mobile Link による同期時にデータベースをユニークに識別する任意の値

デフォルト

NULL (リモート ID なし)

備考

各リモート・データベースには、統合データベースにそれを表す単一のユーザ ID が必要です。ユーザ ID とアドレスがパブリケーションのサブスクライバとして識別されるために、ユーザ ID には REMOTE パーミッションを付与する必要があります。

ml_remote_id が NULL の場合、最初の同期時に Mobile Link サーバによってユニークな値が自動的に割り当てられます。

参照

- 「Ultra Light でのリモート ID の考慮事項」 73 ページ
- 「REMOTE パーミッションの付与」 『SQL Remote』
- 「Ultra Light 情報ユーティリティ (ulinfo)」 217 ページ
- Ultra Light for C/C++：「SetDatabaseOption 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「ULSetDatabaseOptionString 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「SetDatabaseOption メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「SetDatabaseOption メソッド」 『Ultra Light - .NET プログラミング』

Ultra Light の読み込み専用プロパティ

アプリケーションの接続先データベースから取得できる読み込み専用プロパティがいくつかあります。これらのプロパティの値を確認する方法は、使用する API によって異なります。

参照

- 「Ultra Light データベース設定の表示とデータベース・オプションの変更」 112 ページ

Ultra Light char_set プロパティ

データベースの CHAR 文字セットを返します。

備考

データベースで使用されている文字セットは、データベースの照合順と、データが UTF-8 コード化されているかどうかとによって判断されます。

参照

- Ultra Light for C/C++：「GetDatabaseProperty 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「GetDatabaseProperty メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「GetDatabaseProperty メソッド」 『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「getDatabaseProperty メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light utf8_encoding プロパティ」 158 ページ
- 「Ultra Light collation プロパティ」 138 ページ

例

Ultra Light では、値のプロパティをチェックする方法がいくつか用意されています。

Sybase Central の場合　接続先データベースを右クリックして、[プロパティ] を選択します。[データベース・プロパティ] ダイアログ・ボックスが開き、すべての既存のデータベース・プロパティの設定値が表示されます。照合は、[一般] タブの下部にリストされます。

C/C++ の場合　GetDatabaseProperty(ul_database_property_id char_set) を呼び出します。

Ultra Light conn_count プロパティ

データベースとの接続の数を返します。

備考

返される値は動的です。現在存在している接続の数によって変わります。

Ultra Light では、最大 14 の同時データベース接続をサポートできます。

参照

- Ultra Light for C/C++：「GetDatabaseProperty 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「GetDatabaseProperty メソッド」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「GetDatabaseProperty メソッド」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「getDatabaseProperty メソッド」『Ultra Light - M-Business Anywhere プログラミング』

例

Ultra Light では、値のプロパティをチェックする方法がいくつか用意されています。

Sybase Central の場合　接続先データベースを右クリックして、[プロパティ] を選択します。[データベース・プロパティ] ダイアログ・ボックスが表示されるので、[詳細情報] タブをクリックします。このタブに、conn_count パラメータと値がアルファベット順に表示されます。

C/C++ の場合　GetDatabaseProperty(ul_database_property_id conn_count) を呼び出します。

Ultra Light encryption プロパティ

データベースまたはテーブルの暗号化に使用されている暗号化タイプを返します。

戻り値

None、Simple、AES、AES_FIPS のいずれか。

備考

データベースで使用される暗号化は、FIPS 暗号化 (AES か AES_FIPS) と DBKEY 接続パラメータ、または難読化 (単純暗号化) のどちらが設定されているかによって判断されます。

このプロパティを変更できるのは、最初の値が None (fips も obfuscation も使用されていない) の場合だけです。その場合は、使用している API の適切な関数またはメソッドを呼び出して、Connection オブジェクトに新しい暗号化キーを指定することで、暗号化キーを変更できます。この場合、値は AES に変更されます。これは、データベースの作成後に FIPS パラメータを設定することができないためです。

参照

- Ultra Light for C/C++：「GetDatabaseProperty 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「ULChangeEncryptionKey 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「GetDatabaseProperty メソッド」『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「ChangeEncryptionKey メソッド」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「GetDatabaseProperty メソッド」『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「ChangeEncryptionKey メソッド」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「getDatabaseProperty メソッド」『Ultra Light - M-Business Anywhere プログラミング』

- Ultra Light for M-Business Anywhere：「ChangeEncryptionKey メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light でのセキュリティの考慮事項」 70 ページ
- 「Ultra Light fips プロパティ」 143 ページ
- 「Ultra Light obfuscate プロパティ」 147 ページ
- 「Ultra Light DBKEY 接続パラメータ」 188 ページ

例

Ultra Light では、値のプロパティをチェックする方法がいくつか用意されています。

Sybase Central の場合　接続先データベースを右クリックして、[プロパティ] を選択します。[データベース・プロパティ] ダイアログ・ボックスが開き、すべての既存のデータベース・プロパティの設定値が表示されます。暗号化タイプは、[一般] タブの下部にリストされます。

C/C++ の場合　GetDatabaseProperty(ul_database_property_id encryption) を呼び出します。

Ultra Light file プロパティ

データベースまたはテーブルの暗号化に使用されている暗号化タイプを返します。

現在の接続のデータベース・ルート・ファイル名をパスを含めて返します。

備考

返される文字ファイルは、DBF 接続パラメータ値です。

参照

- Ultra Light for C/C++：「GetDatabaseProperty 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「GetDatabaseProperty メソッド」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「GetDatabaseProperty メソッド」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「getDatabaseProperty メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light DBF 接続パラメータ」 175 ページ

例

Ultra Light では、値のプロパティをチェックする方法がいくつか用意されています。

Sybase Central の場合　接続先データベースを右クリックして、[プロパティ] を選択します。[データベース・プロパティ] ダイアログ・ボックスが開き、すべての既存のデータベース・プロパティの設定値が表示されます。データベース・ファイルは、[一般] タブの上部にリストされます。

C/C++ の場合　GetDatabaseProperty(ul_database_property_id file) を呼び出します。

Ultra Light name プロパティ

現在の接続におけるデータベースの名前 (またはエイリアス) を返します。

備考

返される文字ファイルは、DBN 接続パラメータ値です。DBN 接続パラメータを使用しなかった場合、返される名前はパスと拡張子のないデータベース・ファイルになります。

参照

- Ultra Light for C/C++：「GetDatabaseProperty 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「GetDatabaseProperty メソッド」『Ultra Light - AppForge プログラミング』
- Ultra Light.NET：「GetDatabaseProperty メソッド」『Ultra Light - .NET プログラミング』
- Ultra Light for M-Business Anywhere：「getDatabaseProperty メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- 「Ultra Light DBN 接続パラメータ」 186 ページ
- 「Ultra Light DBF 接続パラメータ」 175 ページ

例

Ultra Light では、値のプロパティをチェックする方法がいくつか用意されています。

Sybase Central の場合　接続先データベースを右クリックして、[プロパティ] を選択します。[データベース・プロパティ] ダイアログ・ボックスが開き、すべての既存のデータベース・プロパティの設定値が表示されます。データベース名は、[一般] タブの上部にリストされます。

C/C++ の場合　GetDatabaseProperty(ul_database_property_id name) を呼び出します。

第 9 章

Ultra Light の接続文字列パラメータのリファレンス

目次

Ultra Light CACHE_SIZE 接続パラメータ

データベース・キャッシュのサイズを定義します。

構文

CACHE_SIZE=*number*{ **k** | **m** | **g** }

デフォルト

接続先のプラットフォームによって異なります。

- Windwos デスクトップの場合、デフォルトは 512 KB です。
- Windwos CE の場合、デフォルトは 256 KB です。
- Symbian OS の場合、デフォルトは 128 KB です。
- Palm OS の場合、デフォルトはデバイスで使用できるメモリ容量とメモリのタイプ (ファイルベースまたはレコードベース) によって異なります。

備考

キャッシュ・サイズを指定しなかった場合、またはサイズを 0 に設定した場合は、デフォルトのサイズが使用されます。デフォルトのキャッシュ・サイズは小さめの値です。動的メモリの制約が大きいプラットフォーム (Palm OS の初期のバージョンなど) では、小さめの値にしておくことが非常に重要です。ただし、テスト結果によりパフォーマンスの向上が必要と判断した場合は、キャッシュ・サイズを大きくしてください。

サイズはバイト単位で指定します。キロバイトの単位を示すにはサフィックス k または K を使用し、メガバイトの単位を示すにはサフィックス m または M を使用し、ギガバイトの単位を示すにはサフィックス g または G を使用します。単位を指定しない場合、デフォルトはバイトです。

最大キャッシュ・サイズを超える値を指定すると、自動的にプラットフォームのキャッシュ・サイズ上限が設定されます。たとえば、Symbian OS では、最大キャッシュ・サイズは 2 MB です。キャッシュ・サイズをデータベースのサイズより大きくしても、パフォーマンスは向上しません。

キャッシュ・サイズを大きくすると、使用できる他のアプリケーションの数に影響する場合があります。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「Ultra Light のプラットフォーム別の最適化方法」 49 ページ
- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light page_size プロパティ」 148 ページ
- 「Ultra Light RESERVE_SIZE 接続パラメータ」 194 ページ
- Ultra Light for C/C++ :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』

- Ultra Light for Embedded SQL：「データベースへの接続」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「データベースへの接続」『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」『Ultra Light - .NET プログラミング』

例

次の接続文字列フラグメントによって、キャッシュ・サイズが 128 KB に設定されます。

```
"CACHE_SIZE=128k"
```


Ultra Light COMMIT_FLUSH 接続パラメータ

コミットされたトランザクションが、コミット呼び出しの後にいつ記憶領域にフラッシュされるかを規定します。Ultra Light アプリケーションからコミットが呼び出されないと、フラッシュされません。

構文

COMMIT_FLUSH={ immediate | grouped | on_checkpoint }

指定可能な値

- **immediate** コミットされたトランザクションは、コミット操作が完了する前でも、コミット呼び出しが実行されるとすぐに記憶領域にフラッシュされます。
- **grouped** コミットされたトランザクションは、設定したスレッショルドに達している場合にかぎり、コミット呼び出しがあった時点で記憶領域にフラッシュされます。トランザクション・カウント・スレッショルドを commit_flush_count データベース・オプションを使用して設定するか、時間ベースのスレッショルドを **commit_flush_timeout** データベース・オプションを使用して設定できます。

  設定された場合、**commit_flush_count** オプションと **commit_flush_timeout** オプションはどちらもコミット・フラッシュのトリガとして動作し、先にスレッショルドに達した方がフラッシュをトリガします。フラッシュが実行されると、Ultra Light によってカウンタとタイマはどちらも 0 に戻されます。その後、どちらか一方がスレッショルドに達するまで、両方ともモニタリングされます。
- **on_checkpoint** コミットされたトランザクションは、チェックポイント操作の時点で記憶領域にフラッシュされます。次のいずれかを使用してチェックポイントを作成する必要があります。
  - CHECKPOINT 文。チェックポイント・メソッドのない API では、この SQL 文を使用する必要があります。
  - Ultra Light Embedded SQL 用の ULCheckpoint 関数。
  - C++ コンポーネントの接続オブジェクトの Checkpoint メソッド。

デフォルト

immediate (Ultra Light のデフォルト動作)

備考

このパラメータは、ハードウェア障害またはクラッシュの後に、どのトランザクションをリカバリするかを定義します。論理オートコミット操作を単一のリカバリ・ポイントとしてグループ化できます。

これらの操作をグループ化することで、Ultra Light のパフォーマンスを向上させることができますが、データのリカバリ性が低下します。トランザクションがコミットされた後であっても、トランザクションが記憶領域にフラッシュされる前にハードウェア障害またはクラッシュが発生すると、トランザクションが失われる可能性がわずかながらあります。

参照

- 「単一のトランザクションまたはグループ化されたトランザクションのフラッシュ」 47 ページ
- 「Ultra Light commit_flush_count オプション [テンポラリ]」 160 ページ
- 「Ultra Light commit_flush_timeout オプション [テンポラリ]」 161 ページ
- 「Ultra Light CHECKPOINT 文」 385 ページ
- Ultra Light for Embedded SQL：「ULCheckpoint 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「Checkpoint 関数」 『Ultra Light - C/C++ プログラミング』

Ultra Light CON 接続パラメータ

接続に名前を付け、マルチ接続アプリケーションで簡単に切り替えができるようにします。

適用対象

データベースへの同時接続が必要なアプリケーションで使用します。

構文

CON=*name*

デフォルト

接続名なし

備考

CON パラメータは、アプリケーション全体に対して設定されます。

複数の同時接続を確立し、その接続間で切り替えを行わない場合は、このパラメータは使用しないでください。

接続名はデータベース名とは異なります。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light DBN 接続パラメータ」 186 ページ
- Ultra Light for C/C++：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」 『Ultra Light - .NET プログラミング』

例

次の接続文字列フラグメントによって、最初の接続名が MyFirstCon に設定されます。

```
"CON=MyFirstCon"
```

Ultra Light DBF 接続パラメータ

次の 2 つの異なる機能があります。

- データベースの作成時に、このパラメータによって新しいデータベース・ファイルの名前が付けられます。
- 接続を確立するときに、このパラメータで、ロードして接続するデータベース・ファイルを指定します。

適用対象

一般に、ターゲット・プラットフォームが 1 つのとき、または Ultra Light 管理ツールを使用するときに使用します。

代替パラメータ

単一の接続文字列から、異なるデバイスにある複数のデータベースに接続する場合は、次のパラメータを使用して、プラットフォーム固有の代替パラメータを指定できます。

- CE_FILE
- PALM_FILE
- NT_FILE
- SYMBIAN_FILE

構文

DBF=*ul-db*

デフォルト

プラットフォームによって異なります。

- **デスクトップ・プラットフォームの場合**　NT_FILE または DBF の値を指定しなかった場合は、Ultra Light によってファイル名が *¥UltraLiteDB¥ulstore.udb* に設定されます。
- **Windows CE の場合**　CE_FILE または DBF の値を指定しなかった場合は、Ultra Light によってファイル名が *¥UltraLiteDB¥ulstore.udb* に設定されます。
- **Palm OS の場合**　PALM_FILE または DBF の値を指定しなかった場合は、Ultra Light によってファイル名が *ulstore.udb* に設定されます。
- **Symbian OS の場合**　SYMBIAN_FILE または DBF の値を指定しなかった場合は、Ultra Light によってファイル名が *ulstore.udb* に設定されます。

推奨事項
デフォルトの動作に頼らずに、必ず明示的にパラメータを指定してください。

備考

プラットフォーム固有のパラメータを指定した場合は、そのパラメータが DBF より優先されます。

これらのパラメータはエイリアスなので、DBF を同時に指定した場合は、最後に指定した方が優先します。

DBF の値は、対象プラットフォームのファイル名の要件を満たしている必要があります。

Windows CE デバイスに配備する場合は、Ultra Light のユーティリティやウィザードを使用して、接続している CE デバイス上の Ultra Light データベースを管理できます。CE デバイス上のファイルを指定するには、下記に示す Windows CE の例のように、必ず絶対パスを指定し、**wce:¥** プレフィクスを使用する必要があります。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light 接続パラメータでのファイル・パスの指定」 85 ページ
- 「Ultra Light 管理ツールのパラメータの優先度」 84 ページ
- 「Ultra Light DBN 接続パラメータ」 186 ページ
- 「Ultra Light CE_FILE 接続パラメータ」 177 ページ
- 「Ultra Light PALM_FILE 接続パラメータ」 181 ページ
- 「Ultra Light NT_FILE 接続パラメータ」 179 ページ
- 「Ultra Light SYMBIAN_FILE 接続パラメータ」 184 ページ
- Ultra Light for C/C++：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」 『Ultra Light - .NET プログラミング』

例

デスクトップのディレクトリ *c:¥mydb* にインストールされているデータベース *MyULdb.udb* に接続するには、次の接続文字列を使用します。

```
"DBF=c:¥mydb¥MyULdb.udb"
```

接続している Windows CE デバイスの *Ultralite* フォルダに配備されている同じデータベースに接続するには、次の接続文字列を使用します。

```
"DBF=wce:¥UltraLite¥MyULdb.udb"
```

Ultra Light CE_FILE 接続パラメータ

次の 2 つの異なる機能があります。

- データベースの作成時に、このパラメータによって新しいデータベース・ファイルの名前が付けられます。
- 既存のデータベースへの接続を確立するときに、このパラメータがデータベースを識別します。

適用対象

同じ接続文字列を使用して Windows CE デバイスと別のプラットフォームに接続する、Ultra Light クライアント・アプリケーションで使用します。

代替パラメータ

Ultra Light 管理ツールから接続する場合、または接続オブジェクトが Windows CE データベースに接続する場合は、DBF を使用します。

構文

CE_FILE=*path¥ce-db*

デフォルト

デフォルトは、設定によって異なります。

- この接続パラメータに値を設定しなかった場合は、DBF パラメータの値が使用されます (設定されている場合)。
- このパラメータの値と DBF パラメータの値をどちらも指定しなかった場合、デフォルト値は *¥UltraLiteDB¥ulstore.udb* です。

推奨事項
デフォルトの動作に頼らずに、必ず明示的にパラメータを指定してください。

備考

このパラメータは DBF パラメータより優先されます。

CE_FILE の値は、Windows CE のファイル名の要件を満たしている必要があります。

データベースへの絶対パスを含める場合は、ディレクトリをすべて作成してから、このファイルのパスを設定する必要があります。ディレクトリは自動的には作成されません。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light 接続パラメータでのファイル・パスの指定」 85 ページ

- 「Ultra Light 管理ツールのパラメータの優先度」 84 ページ
- 「Ultra Light DBF 接続パラメータ」 175 ページ
- Ultra Light for C/C++：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」 『Ultra Light - .NET プログラミング』

例

次の例では、新しい接続を作成し、デスクトップおよび Windows CE プラットフォームに異なるデータベース・ファイルを指定します。

```
Set Connection = DatabaseMgr.OpenConnection("DBF=d:¥Dbfile.udb;CE_FILE=¥myapp¥MyDB.udb")
```

Ultra Light NT_FILE 接続パラメータ

次の 2 つの異なる機能があります。

- データベースの作成時に、このパラメータによって新しいデータベース・ファイルの名前が付けられます。
- 既存のデータベースへの接続を確立するときに、このパラメータがデータベースを識別します。

適用対象

同じ接続文字列を使用してデスクトップのデータベースと別のプラットフォームに接続する、Ultra Light クライアント・アプリケーションで使用します。

代替パラメータ

Ultra Light 管理ツールから接続する場合、または接続オブジェクトがデスクトップ・データベースに接続する場合は、DBF を使用します。

構文

NT_FILE=path¥nt-db

デフォルト

デフォルトは、設定によって異なります。

- この接続パラメータに値を設定しなかった場合は、DBF パラメータの値が使用されます (設定されている場合)。
- このパラメータの値と DBF の値をどちらも指定しなかった場合、デフォルト値は *¥UltraLiteDB¥ulstore.udb* です。

推奨事項
デフォルトの動作に頼らずに、必ず明示的にパラメータを指定してください。

備考

このパラメータは DBF パラメータより優先されます。

NT_FILE の値は、Windows デスクトップ・プラットフォームのファイル名の要件を満たしている必要があります。

パスは絶対パスまたは相対パスのどちらでもかまいません。ファイル名にディレクトリを含める場合は、ディレクトリをすべて作成してから、このファイルのパスを設定する必要があります。ディレクトリは自動的には作成されません。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light 接続パラメータでのファイル・パスの指定」 85 ページ
- 「Ultra Light 管理ツールのパラメータの優先度」 84 ページ
- 「Ultra Light DBF 接続パラメータ」 175 ページ
- Ultra Light for C/C++：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」 『Ultra Light - .NET プログラミング』

例

次の例では、新しい接続を作成し、デスクトップ、Palm OS、Windows CE の各プラットフォームに異なるデータベース・ファイルを指定します。

```
Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;
NT_FILE=c:¥test¥MyTestDB.udb;CE_FILE=¥database
¥MyCEDB.udb;PALM_FILE=MyPalmDB_MyCreatorID")
```

Ultra Light PALM_FILE 接続パラメータ

次の 2 つの異なる機能があります。

- データベースの作成時に、このパラメータによって新しいデータベース・ファイルの名前が付けられます。
- 既存のデータベースへの接続を確立するときに、このパラメータがデータベースを識別します。

適用対象

同じ接続文字列を使用して Palm デバイスと別のプラットフォームに接続する、Ultra Light クライアント・アプリケーションで使用します。

代替パラメータ

Ultra Light 管理ツールから接続する場合、または接続オブジェクトが Palm OS データベースに接続する場合は、DBF を使用します。

構文 1：レコードベース・ストアの場合

PALM_FILE=*name*

構文 2：ファイルベース・ストアの場合

PALM_FILE=vfs: [*volume-label*: | *volume-ordinal*:] *filename*

デフォルト

デフォルトは、設定によって異なります。

- この接続パラメータに値を設定しなかった場合は、DBF パラメータの値が使用されます (設定されている場合)。
- このパラメータの値と DBF パラメータの値をどちらも指定しなかった場合、デフォルト値は *ulstore.udb* です。

推奨事項
デフォルトの動作に頼らずに、必ず明示的にパラメータを指定してください。

備考

デスクトップの管理ユーティリティでは、このパラメータを使用しないでください。Palm データベースの編集中は、代わりに DBF パラメータを使用してください。

PALM_FILE の値は、Palm OS プラットフォームのファイル名の要件を満たしている必要があります。

AppForge で開発している場合は、PALM_FILE パラメータも使用する必要があります。

このパラメータは DBF パラメータより優先されます。

ファイルベース・ストアでは、常に絶対ファイル・パスを指定します。フル・パス名でディレクトリを指定し、そのディレクトリが見つからない場合は、新しく作成されます。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light 接続パラメータでのファイル・パスの指定」 85 ページ
- 「Ultra Light 管理ツールのパラメータの優先度」 84 ページ
- 「Palm 作成者 ID を登録する」 『Ultra Light - C/C++ プログラミング』
- 「Ultra Light PALM_DB 接続パラメータ」 183 ページ
- 「Ultra Light DBF 接続パラメータ」 175 ページ
- Ultra Light for C/C++ :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere :「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere :「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET :「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET :「Open メソッド」 『Ultra Light - .NET プログラミング』

例

次の例では、新しい接続を作成し、デスクトップ、Palm OS、Windows CE の各プラットフォームに異なるデータベース・ファイルを指定します。

```
Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;
NT_FILE=c:¥test¥MyTestDB.udb;CE_FILE=¥database
¥MyCEDB.udb;PALM_FILE=MyPalmDB_MyCreatorID")
```

Ultra Light PALM_DB 接続パラメータ

AppForge の作成者 ID ではなく、Ultra Light の作成者 ID が使用されるように、正しい作成者 ID を設定します。

適用対象

AppForge で Palm OS 用の Ultra Light クライアント・アプリケーションを開発するときは、PALM_FILE 接続パラメータと同時に使用します。

構文

PALM_DB=*creator-ID*

デフォルト

AppForge の作成者 ID

備考

作成者 ID は、4 文字の文字列です。AppForge の作成者 ID とは別の作成者 ID を使用する場合は、デフォルト値を変更してください。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light PALM_FILE 接続パラメータ」 181 ページ
- PalmSource の作成者 ID のページ (英語)
- Ultra Light for C/C++ :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere :「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere :「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET :「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET :「Open メソッド」 『Ultra Light - .NET プログラミング』

例

次の例では、新しい接続を作成し、デスクトップ、Palm OS、Windows CE の各プラットフォームに異なるデータベース・ファイルを指定します。

```
Set Connection = DatabaseMgr.OpenConnection("file_name=d:¥MyDB.udb;
PALM_FILE=MyPOSdb;PALM_DB=Syb3;CE_FILE=¥myapp¥MyCEdb.udb")
```


Ultra Light SYMBIAN_FILE 接続パラメータ

次の 2 つの異なる機能があります。

- データベースの作成時に、このパラメータによって新しいデータベース・ファイルの名前が付けられます。
- 既存のデータベースへの接続を確立するときに、このパラメータがデータベースを識別します。

適用対象

同じ接続文字列を使用して Symbian デバイスと別のプラットフォームに接続する、Ultra Light クライアント・アプリケーションで使用します。

代替パラメータ

Ultra Light 管理ツールから接続する場合、または接続オブジェクトが Symbian OS データベースに接続する場合は、DBF を使用します。

構文

SYMBIAN_FILE=*symbian-db*

デフォルト

デフォルトは、設定によって異なります。

- この接続パラメータに値を設定しなかった場合は、DBF パラメータの値が使用されます (設定されている場合)。
- このパラメータの値と DBF パラメータの値をどちらも指定しなかった場合、デフォルト値は *ulstore.udb* です。

推奨事項
デフォルトの動作に頼らずに、必ず明示的にパラメータを指定してください。

備考

このパラメータは DBF パラメータより優先されます。

SYMBIAN_FILE の値は、Symbian OS プラットフォームのファイル名の要件を満たしている必要があります。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light 接続パラメータでのファイル・パスの指定」 85 ページ
- 「Ultra Light 管理ツールのパラメータの優先度」 84 ページ
- 「Ultra Light DBF 接続パラメータ」 175 ページ

- Ultra Light for C/C++：「データベースへの接続」『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「OpenConnection 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「データベースへの接続」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「データベースへの接続」『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」『Ultra Light - .NET プログラミング』

例

次の例では、新しい接続を作成し、デスクトップ、Symbian OS、Windows CE の各プラットフォームに異なるデータベース・ファイルを指定します。

```
Connection = DatabaseMgr.OpenConnection("UID=JDoe;PWD=ULdb;
NT_FILE=c:¥test¥MyTestDB.udb;CE_FILE=MyCEdb.udb;SYMBIAN_FILE=MySymbianDB.udb")
```


Ultra Light DBN 接続パラメータ

アプリケーションが複数のデータベースに接続した場合に、データベースを名前で識別します。

適用対象

すでに開いているデータベースへの接続に使用します。

構文

DBN=*db-name*

デフォルト

プラットフォームによって異なります。

- **Palm OS の場合**　デフォルト値は作成者 ID です。
- **その他すべてのプラットフォームの場合**　デフォルト値は、ファイル名の値から、パスと拡張子を削除したもの (ある場合) になります。長さは 16 文字以内です。

備考

Ultra Light では、データベースを開いてからデータベース名が設定されます。データベースが開いたら、クライアント・アプリケーションでファイルではなく名前を使用してデータベースに接続できます。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

例

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light DBF 接続パラメータ」 175 ページ
- Ultra Light for C/C++：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」 『Ultra Light - .NET プログラミング』

次のパラメータを使用して Kitchener という実行中の Ultra Light データベースに接続することもできます。

```
DBN=Kitchener;DBF=cities.udb
```


Ultra Light DBKEY 接続パラメータ

次の 2 つの異なる機能があります。

- データベースを作成するときに、このパラメータでデータベースの暗号化キーを指定します。
- 既存のデータベースへの接続を確立するときに、このパラメータで、データベースに設定されている暗号化キーを指定します。指定したキーが間違っていた場合は、接続に失敗します。

適用対象

暗号化されたデータベースに接続するアプリケーションで使用します。

構文

DBKEY=*string*

デフォルト

キーは指定されません。

備考

暗号化キーを使用してデータベースを作成すると、データベース・ファイルは FIPS または AES の 128 ビット・アルゴリズムを使用して強力に暗号化されます。AES は、SQL Anywhere データベースの暗号化に使用されているアルゴリズムと同じです。強力な暗号化を使用すると、巧妙で容赦ないデータへのアクセス試行に対抗するセキュリティが提供されますが、パフォーマンスに大きな影響を与える可能性があります。

Palm OS では、ユーザが別のアプリケーションに切り替えると、アプリケーションがシステムによって自動的に終了します。ただし、ユーザが同じアプリケーションに戻るたびにキーを再入力する必要がないように Ultra Light クライアントをプログラミングできます。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- Ultra Light for C/C++：「Palm OS の暗号化キーの保存、取り出し、クリア」『Ultra Light - C/C++ プログラミング』
- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light でのセキュリティの考慮事項」 70 ページ
- 「Ultra Light obfuscate プロパティ」 147 ページ
- Ultra Light for C/C++：「データベースへの接続」『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「OpenConnection 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「データベースへの接続」『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」『Ultra Light - AppForge プログラミング』

- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「データベースへの接続」『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」『Ultra Light - .NET プログラミング』

Ultra Light ORDERED_TABLE_SCAN 接続パラメータ

下位互換性のためのパラメータです。クエリによるローへのアクセスに、プライマリ・キー・インデックスを使用するか、またはデータベース・ページから直接アクセスするかを制御します。10.0.1 より前のバージョンの Ultra Light では、Ultra Light のオプティマイザによって他のインデックスが選択されていない場合には、結果を返すのにプライマリ・キー・インデックスが使用されます。

構文

ORDERED_TABLE_SCAN={ yes | no }

指定可能な値

すべてのブール値を指定できます。たとえば、true と false、yes と no、1 と 0 などです。

デフォルト

0。データベース・ページは直接スキャンされます (現状の Ultra Light の動作)。

備考

返される結果でローがプライマリ・キー・インデックスの順序になるようにする場合は、ORDER BY 句が最初に含まれるようにクエリを書き直してください。そうしないと、この接続でダイレクト・ページ・スキャンを使用できることによるパフォーマンス向上が実現されなくなります。このパラメータを使用するのは、ORDER BY 句を使用するようクエリを書き直すことが現実的でない場合に限定してください。

参照

- 「ダイレクト・ページ・スキャンの使用」 45 ページ
- Ultra Light for C/C++ :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere :「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere :「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET :「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET :「Open メソッド」 『Ultra Light - .NET プログラミング』

Ultra Light PALM_ALLOW_BACKUP 接続パラメータ

HotSync を使用したバックアップの動作を制御します。デフォルトでは Ultra Light によって無効になります。

適用対象

Palm OS デバイスの Ultra Light の HotSync コンジットを使用したデスクトップ・バックアップが対象です。

構文

PALM_ALLOW_BACKUP={ yes | no }

備考

Palm では、HotSync を使用してデータベースをデスクトップにバックアップできます。ほとんどの Ultra Light クライアント・アプリケーションでは、データは同期によってバックアップされるので、デスクトップへの非公式のバックアップを使用する必要はありません。このため、Ultra Light のランタイムでは、Palm のバックアップ動作が無効になります。ただし、同期時に HotSync で Ultra Light データベースをデスクトップにバックアップする必要がある場合は、このパラメータを使用して Ultra Light のデフォルトを変更できます。

HotSync を使用したバックアップを有効にしたら、ULDBUtil でバックアップを設定する必要はありません。バックアップを無効にするまで、同期のたびにバックアップが行われます。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light の Palm OS 用データ管理ユーティリティ (ULDBUtil)」 234 ページ
- Ultra Light for C/C++ :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere :「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere :「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET :「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET :「Open メソッド」 『Ultra Light - .NET プログラミング』

Ultra Light PWD 接続パラメータ

認証に使用するユーザ ID のパスワードを定義します。

構文

PWD=*password_string*

デフォルト

UID と PWD の*両方*を設定しなかった場合、Ultra Light では **UID=DBA** と **PWD=sql** で接続が確立されます。

備考

パスワードは NULL または空の文字列に設定できますが、最大長 31 文字以内で指定する必要があります。

すべてのデータベース・ユーザにはパスワードがあります。Ultra Light では、最大で 4 つのユーザ ID とパスワードの組み合わせがサポートされています。

この接続パラメータは暗号化されません。ただし、Ultra Light では、パスワードがハッシュされてから保存されます。したがって、パスワードは Sybase Central だけで変更できます。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「ユーザ認証の役割」 79 ページ
- 「ユーザ ID とパスワードの組み合わせの解釈」 80 ページ
- 「Ultra Light UID 接続パラメータ」 197 ページ
- Ultra Light for C/C++：「ユーザの認証」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「ユーザの認証」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL：「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge：「ユーザの認証」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge：「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「ユーザの認証」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「ユーザの認証」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」 『Ultra Light - .NET プログラミング』

例

次の接続文字列フラグメントによって、ユーザ ID **DBA** とパスワード **sql** が指定されます。

```
"UID=DBA;PWD=sql"
```

次の接続文字列フラグメントによって、ユーザ ID **DBA** と空のパスワードが指定されます。

```
"UID=DBA;PWD=""
```

Ultra Light RESERVE_SIZE 接続パラメータ

実際にデータを挿入することなく、Ultra Light データベースが必要とするファイル・システム領域を事前に割り付けます。

構文

RESERVE_SIZE= *number*{ **k** | **m** | **g** }

デフォルト

0 (予約サイズなし) です。

備考

0 からデータベースの最大サイズまでの任意の値を指定できます。サイズはバイト単位で指定します。キロバイトの単位を示すにはサフィックス k または K を使用し、メガバイトの単位を示すにはサフィックス m または M を使用し、ギガバイトの単位を示すにはサフィックス g または G を使用します。単位を指定しない場合、デフォルトはバイトです。

RESERVE_SIZE の値がデータベースのサイズよりも小さい場合、このパラメータは無視されます。

ファイル・システム領域を予約すると、パフォーマンスが多少向上します。また、メモリ不足障害を防ぐこともできます。Ultra Light データベースはデータとメタデータで構成されているので、データベース・サイズが大きくなるのは必要がある場合 (具体的には、アプリケーションがデータベースを更新する場合) のみです。RESERVE_SIZE パラメータでは、データベースの作成時または起動時にデータベースを指定された予約サイズまで拡大することで領域が予約されます。これにより、データベースはトランケートされなくなります。データベースをテスト・データを使用して実行することによってデータベース・サイズを確認し、Ultra Light の配備に適した予約サイズを選択してください。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「Ultra Light CACHE_SIZE 接続パラメータ」 170 ページ
- 「Ultra Light page_size プロパティ」 148 ページ
- Ultra Light for C/C++ :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere :「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere :「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET :「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET :「Open メソッド」 『Ultra Light - .NET プログラミング』

例

次の接続文字列フラグメントによって、予約サイズが 128 KB に設定され、起動時にシステムによってデータベース用にこの領域が予約されるようになります。

```
"RESERVE_SIZE=128K"
```

Ultra Light START 接続パラメータ

Ultra Light エンジンの実行プログラムのロケーションを指定し、起動します。

適用対象

Ultra Light エンジンのクライアント・アプリケーション

構文

START=path¥uleng10.exe

デフォルト

Ultra Light エンジンの startline は使用しません。

備考

現在実行中でないエンジンに接続するときにかぎり、StartLine (START) 接続パラメータを指定します。

参照

- 「Ultra Light エンジンユーティリティ (uleng10)」 216 ページ
- 「Ultra Light データ管理コンポーネントの選択」 32 ページ
- Ultra Light for C/C++ :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere :「Ultra Light データベースへの接続」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere :「 openConnection メソッド」 『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET :「データベースへの接続」 『Ultra Light - .NET プログラミング』
- Ultra Light.NET :「Open メソッド」 『Ultra Light - .NET プログラミング』

Ultra Light UID 接続パラメータ

データベースへの接続に使用するユーザ ID。値には、データベースで認証されたユーザを指定します。

構文

UID=*id_name*

デフォルト

UID と PWD の*両方*を設定しなかった場合、Ultra Light では **UID=DBA** と **PWD=sql** で接続が確立されます。

備考

UID は NULL または空の文字列に設定できません。ユーザ ID は最大長 31 文字以内で指定する必要があります。

すべてのデータベース・ユーザがユーザ ID を持ちます。Ultra Light では、最大で 4 つのユーザ ID とパスワードの組み合わせがサポートされています。

Ultra Light のユーザ ID は、Mobile Link のユーザ名やその他の SQL Anywhere のユーザ ID とは別です。

ユーザ ID は一度作成すると変更できません。変更する必要がある場合は、ユーザ ID を削除し、新しい ID を追加してください。

ユーザ ID では大文字と小文字は区別されません。

パラメータ値に含まれる前後のスペースはすべて無視されます。この接続パラメータの値に、先頭の一重引用符 (')、先頭の二重引用符 (")、またはセミコロン (;) を含めることはできません。

参照

- 「接続文字列を使用した Ultra Light 接続の確立」 82 ページ
- 「ユーザ認証の役割」 79 ページ
- 「ユーザ ID とパスワードの組み合わせの解釈」 80 ページ
- Ultra Light for C/C++ :「ユーザの認証」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++ :「OpenConnection 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「ユーザの認証」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for Embedded SQL :「データベースへの接続」 『Ultra Light - C/C++ プログラミング』
- Ultra Light for AppForge :「ユーザの認証」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「Ultra Light データベースへの接続」 『Ultra Light - AppForge プログラミング』
- Ultra Light for AppForge :「OpenConnection メソッド」 『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere :「ユーザの認証」 『Ultra Light - M-Business Anywhere プログラミング』

- Ultra Light for M-Business Anywhere：「Ultra Light データベースへの接続」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light for M-Business Anywhere：「 openConnection メソッド」『Ultra Light - M-Business Anywhere プログラミング』
- Ultra Light.NET：「ユーザの認証」『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「データベースへの接続」『Ultra Light - .NET プログラミング』
- Ultra Light.NET：「Open メソッド」『Ultra Light - .NET プログラミング』

例

次の接続文字列フラグメントによって、*c:¥MyOrders.udb* というデータベースにユーザ ID **DBA** とパスワード **sql** が指定されます。

```
"UID=DBA;PWD=sql"
```

第 10 章

Ultra Light ユーティリティ・リファレンス

目次

Ultra Light のユーティリティの概要

Ultra Light には、コマンド・プロンプトからデータベースの基本的な管理作業を行うことができる一連のユーティリティがあります。これらのユーティリティの多くは、SQL Anywhere のユーティリティと機能が似ています。ただし、オプションの使用方法が異なる場合があります。Ultra Light でのオプションの実装方法については、必ず Ultra Light のリファレンスを参照してください。

注意
この章に示すユーティリティのオプションでは、特に明記されていないかぎり、大文字と小文字が区別されます。オプションはここに示すとおり*正確に*入力してください。

サポートされている終了コード

ulcreate、ulload、ulsync、ulunload の各ユーティリティは、処理が正常に終了したかどうかを示す終了コードを返します。0 は同期が成功したことを示します。他の値は同期が失敗したことを示します。

終了コード	ステータス	説明
0	EXIT_OKAY	処理は成功した。
1	EXIT_FAIL	処理は失敗した。
3	EXIT_FILE_ERROR	データベースが見つからない。
4	EXIT_OUT_OF_MEMORY	デバイスの動的メモリが足りない。
6	EXIT_COMMUNICATIONS_FAIL	Ultra Light エンジンとの通信中に通信エラーが発生した。
9	EXIT_UNABLE_TO_CONNECT	指定された UID または PWD が無効であるため、データベースに接続できない。
12	EXIT_BAD_ENCRYPT_KEY	暗号化キーがないか、無効である。
13	EXIT_DB_VER_NEWER	データベースのバージョンに互換性がないことが検出された。データベースを新しいバージョンにアップグレードする必要がある。
255	EXIT_USAGE	コマンド・ライン・オプションが無効である。

Ultra Light 用 Interactive SQL ユーティリティ (dbisql)

SQL コマンドを実行するか、指定するコマンド・ファイルを実行します。

構文

この後に示す構文は、Ultra Light で使用するためのものです。SQL Anywhere で使用するための構文については、「Interactive SQL ユーティリティ (dbisql)」『SQL Anywhere サーバ - データベース管理』を参照してください。

dbisql -c "*connection-string*" [*options*] [*dbisql-command* | *command-file*]

オプション	説明
@*data*	指定された環境変数または設定ファイルからオプションを読み出します。指定された環境変数と設定ファイルが両方とも存在する場合は、環境変数が使用されます。 設定ファイル内のパスワードなどの情報を保護する場合は、ファイル非表示ユーティリティを使用して、設定ファイルの内容を難読化できます。
-c "*connection-string*"	必須。*connection-string* の DBF パラメータまたは file_name パラメータで指定するデータベースに接続します。ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-codepage *number*	ファイルの読み込みまたは書き込みをするときに使用するコード・ページを指定します。デフォルトのコード・ページは、実行しているプラットフォームのデフォルト・コード・ページです。
-d *delimiter*	コマンド・デリミタを指定します。デリミタを囲む引用符は省略可能ですが、コマンド・シェル自体がデリミタを特別な方法で解釈するときは必ず指定します。 コマンド・デリミタは、データベースに格納されている設定に関係なく、その Interactive SQL セッション中のすべての接続に使用されます。
-d1	ユーザが明示的に実行するすべての文をコマンド・ウィンドウ (STDOUT) にエコーします。これによって、SQL スクリプトのデバッグ、または Interactive SQL が長文の SQL スクリプトを処理しているときに有用なフィードバックが提供されます。
-f *filename*	*filename* というファイルを (実行せずに) 開きます。ファイル名にスペースが含まれる場合はファイル名を引用符で囲みます。 そのファイルが存在しない場合、またはファイルではなく実際にはディレクトリである場合は、Interactive SQL がエラー・メッセージをコンソールに出力して終了します。ファイル名に完全なドライブとパスの仕様が含まれていないときは、現在のディレクトリが基準として想定されます。

オプション	説明
-nogui	コマンド・プロンプト・モードで実行します。*dbisql-command* または *command-file* のいずれかを指定する場合、−nogui が使用されます。
-onerror *behavior*	コマンド・ファイルからデータを読み出し中にエラーが起こった場合の事象を制御します。*behavior* には次のいずれかの値を定義できます。 ♦ **Stop** Interactive SQL が文の実行を停止します。 ♦ **Prompt** Interactive SQL は、ユーザに継続したいかどうかを確認するプロンプトを表示します。 ♦ **Continue** エラーは無視され、Interactive SQL は文の実行を継続します。 ♦ **Exit** Interactive SQL は終了します。 ♦ **Notify_Continue** エラーがレポートされますが、ユーザは継続させるために [Enter] を押すか、[OK] をクリックするよう要求されます。 ♦ **Notify_Stop** エラーがレポートされますが、ユーザは実行を停止するために [Enter] を押すか、[OK] をクリックするよう要求されます。 ♦ **Notify_Exit** エラーがレポートされますが、ユーザは Interactive SQL を終了するために [Enter] を押すか、[OK] をクリックするよう要求されます。
-q	ユーティリティをクワイエット・モードで実行するように設定します。情報のバナー、バージョン番号、ステータス・メッセージが非表示になります。エラー・メッセージは引き続き表示されます。
-x	コマンドをスキャンしますが、実行しません。長いコマンド・ファイルの構文エラーをチェックする場合に有用です。
-ul	デフォルトで Ultra Light データベースに接続します。 Interactive SQL では、デフォルトで SQL Anywhere データベースに接続すると見なされます。-ul オプションを指定すると、デフォルトが Ultra Light への接続になります。デフォルトに設定されているデータベースのタイプに関係なく、[接続] ダイアログのドロップダウン・リストからデータベースのタイプを選択することで、SQL Anywhere または Ultra Light のデータベースに接続できます。
dbisql-command|*command-file*	指定する SQL コマンドまたは *command-file* 内で指定されている SQL コマンドを実行します。 *dbisql-command* または *command-file* を指定しなかった場合、Interactive SQL は対話型モードになり、コマンドをコマンド・ウィンドウに入力できます。

備考

Interactive SQL を使うと、SQL コマンドを入力したり、Interactive SQL コマンド・ファイルを実行したりできます。また、影響を受けたローの数、各コマンドに必要な時間、クエリの実行計画、エラー・メッセージに関するフィードバックも提供します。

Ultra Light データベースに接続している場合は、SQL Anywhere に固有の一部のメニュー項目がインタフェースに表示されません。たとえば、[ツール] – [プロシージャ名のルックアップ] や [ツール] – [インデックス・コンサルタント] は表示されません。

コード・ページについて　Ultra Light では、照合にコード・ページとソート順が含まれます。したがって、コード・ページの番号は、Ultra Light の照合名の一部として表示される番号に対応します。コマンド・プロンプトで ulcreate -l を実行すると、サポートされている照合と対応するコード・ページのリストが表示されます。

たとえば、英語版の 32 ビット Windows デスクトップ・コンピュータでは、ウィンドウを使用するプログラムは 1252 (ANSI) コード・ページを使用します。297 (IBM France) コード・ページを使用して作成されたファイルを Interactive SQL で読み込む場合には、**-codepage 297** オプションを指定します。

終了コードについて　終了コードは、0 (成功) または 0 以外の値 (失敗) です。0 以外の終了コードが設定されるのは、(SQL 文またはスクリプト・ファイル名を指定したコマンド・ラインによって) Interactive SQL をバッチ・モードで実行した場合だけです。

GUI がないモードのとき、Interactive SQL は、処理の成功または失敗をプログラム終了コードを設定することで示します。Windows オペレーティング・システムでは、プログラム終了コードに対して環境変数 ERRORLEVEL が設定されます。

参照

- 「設定ファイルの使用」『SQL Anywhere サーバ - データベース管理』
- 「ファイル非表示ユーティリティ (dbfhide)」『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「サポートされている終了コード」 200 ページ

例

次のコマンドをコマンド・プロンプトで入力すると、デフォルトのユーザ ID **DBA** とパスワード **sql** で、Ultra Light データベース *CustDB.udb* に対してコマンド・ファイル *mycom.sql* が実行されます。コマンド・ファイルにエラーがあった場合は、処理は終了します。

```
dbisql -ul -c DBF=CustDB.udb -onerror exit mycom.sql
```


Ultra Light SQL プリプロセッサ・ユーティリティ (sqlpp)

Embedded SQL (ESQL) を含む C/C++ のプログラムをプリプロセッサで処理し、コンパイラを実行する前に、このプログラムに必要なコードを生成できるようにします。

ヒント
SQL プリプロセッサ (sqlpp) には、Embedded SQL アプリケーション内の静的 SQL 文をコンパイル時に通知する機能があります。この機能は、Ultra Light アプリケーションを開発するときに特に便利です。この機能で、SQL 文に Ultra Light との互換性があることを確認できます。SQL Anywhere アプリケーションと Ultra Light アプリケーションの両方に対する SQL の互換性をテストするには、**-e** オプションか **-w** オプションまたはその両方を使用します。SQL FLAGGER の詳細については、「SQL FLAGGER を使用した SQL 準拠のテスト」『SQL Anywhere サーバ - SQL の使用法』を参照してください。

構文

この後に示す構文は、Ultra Light で使用するためのものです。

sqlpp [*options*] *esql-filename* [*output-filename*]

SQL Anywhere で使用するための構文については、「SQL プリプロセッサ」『SQL Anywhere サーバ - プログラミング』を参照してください。

オプション	説明
-d	データ領域サイズを減らし、コード・サイズを増加させるコードを生成します。データ構造体を再利用し、実行時に初期化してから使用します。

オプション	説明
-e *level*	このオプションは、指定した規格に含まれない静的 Embedded SQL をエラーとして通知します。*level* 値は、使用する規格を表します。たとえば sqlpp -e c03 ... は、コア SQL/2003 規格に含まれない構文を通知します。 *level* として使用される値は次のとおりです。 ♦ **c03** コア SQL/2003 構文でない構文を通知します。 ♦ **p03** 上級 SQL/2003 構文でない構文を通知します。 ♦ **c99** コア SQL/1999 構文でない構文を通知します。 ♦ **p99** 上級 SQL/1999 構文でない構文を通知します。 ♦ **e92** 初級レベル SQL/1992 構文でない構文を通知します。 ♦ **i92** 中級レベル SQL/1992 構文でない構文を通知します。 ♦ **f92** 上級 SQL/1992 構文でない構文を通知します。 ♦ **t** 標準ではないホスト変数型を通知します。 ♦ **u** Ultra Light がサポートしていない構文を通知します。 以前のバージョンの SQL Anywhere と互換性を保つために、e、i、f を指定することもできます。これらはそれぞれ e92、i92、f92 に対応します。
-h	sqlpp によって出力される *.c* ファイル内の分割された行の最大長を *width* に制限します。分割された行が C コンパイラで 1 行として解析されるように、分割された行の末尾には円記号が追加されます。デフォルト値はありません (出力の行はデフォルトで分割されません)。
-k	コンパイルされるプログラムが SQLCODE のユーザ宣言をインクルードすることをプリプロセッサに通知します。
-n	コード内の適切な場所で #line ディレクティブを使用して、C ファイル内に行番号情報を生成します。 このオプションを使用して、ソースのエラーをレポートし、また *output-filename* ファイルではなく、*esql-filename* ファイル内の行番号にあるソースをデバッグできます。
-o	Ultra Light に不適用
-q	ユーティリティをクワイエット・モードで実行するように設定します。情報のバナー、バージョン番号、ステータス・メッセージが非表示になります。エラー・メッセージは引き続き表示されます。
-r	Ultra Light に不適用

オプション	説明
-s *string-length*	プリプロセッサが C ファイルに出力する文字列の最大サイズを設定します。この値より長い文字列は、文字のリスト (*'a'*、*'b'*、*'c'* など) を使用して初期化されます。ほとんどの C コンパイラには、処理できる文字列リテラルのサイズに制限があります。このオプションを使用して上限を設定します。デフォルト値は 500 です。
-u	Ultra Light で必須です。Ultra Light データベースに必要な出力を生成します。
-w *level*	SQL 構文に準拠しないものを警告として通知します。*level* 値は、使用する規格を表します。たとえば sqlpp -w c03 ... は、コア SQL/2003 構文に含まれない構文を通知します。 *level* として使用される値は次のとおりです。 ♦ **c03** コア SQL/2003 構文でない構文を通知します。 ♦ **p03** 上級 SQL/2003 構文でない構文を通知します。 ♦ **c99** コア SQL/1999 構文でない構文を通知します。 ♦ **p99** 上級 SQL/1999 構文でない構文を通知します。 ♦ **e92** 初級レベル SQL/1992 構文でない構文を通知します。 ♦ **i92** 中級レベル SQL/1992 構文でない構文を通知します。 ♦ **f92** 上級 SQL/1992 構文でない構文を通知します。 ♦ **t** 標準ではないホスト変数型を通知します。 ♦ **u** Ultra Light がサポートしていない構文を通知します。 以前のバージョンの SQL Anywhere と互換性を保つために、e、i、f を指定することもできます。これらはそれぞれ e92、i92、f92 に対応します。
-x	マルチバイト文字列をエスケープ・シーケンスに変更し、コンパイラをパススルーできるようにします。
-z *collation-sequence*	照合順を指定します。

備考

出力ファイルについて このプリプロセッサは、入力ファイル内の SQL 文を C/C++ に変換します。結果は *output-filename* に書き込みます。Embedded SQL を含むソース・プログラムの拡張子は通常は *sqc* です。デフォルトの *output-filename* は拡張子 *c* が付いた *esql-filename* ベースの名前です。ただし、*esql-filename* に拡張子 *.c* がすでに付いている場合、デフォルトの出力ファイル拡張子は *.cc* になります。

照合について 照合順は、プリプロセッサにプログラムのソース・コードで使用されている文字を理解させるために使用します。たとえば、識別子に使用できるアルファベット文字の識別などに使用されます。Ultra Light では、照合にコード・ページとソート順が含まれます。-z を指定し

なかった場合は、プリプロセッサが、オペレーティング・システムに基づいて、使用する合理的な照合順を決定しようとします。

コマンド・プロンプトで ulcreate -l を実行すると、サポートされている照合と対応するコード・ページのリストが表示されます。

参照

- 「SQL Anywhere Embedded SQL」 『SQL Anywhere サーバ - プログラミング』
- 「Ultra Light での文字の考慮事項」 63 ページ

例

次のコマンドは、Ultra Light アプリケーション用に Embedded SQL ファイル *srcfile.sqc* をクワイエット・モードで前処理します。

```
sqlpp -u -q MyEsqlFile.sqc
```


Ultra Light AppForge レジストリ・ユーティリティ (ulafreg)

Ultra Light をインストールした後に AppForge 開発環境をインストールした場合、このユーティリティは、適切な Ultra Light ネームスペースへの参照が AppForge 開発環境に含まれるように、Ultra Light ランタイムまたは Ultra Light エンジン・クライアントのどちらかを登録します。この変更により、AppForge タイプのライブラリをロードできるようになります。該当する *.DLL ファイルは、このユーティリティを使用して登録解除することもできます。

構文

ulafreg [*options*]

オプション	説明
-d	現在インストールされているバージョンの AppForge のロケーションを表示します。
-q	ユーティリティをクワイエット・モードで実行するように設定します。情報のバナー、バージョン番号、ステータス・メッセージが非表示になります。エラー・メッセージは引き続き表示されます。
-r [*directory*]	指定された *directory* にある Ultra Light ランタイム *.DLL (たとえば *ulmvb10.dll*) を登録します。このファイルを参照できるアプリケーションは 1 つだけです。 *directory* を設定しなかった場合は、デフォルトの SQL Anywhere のインストール先ロケーション (*install-dir¥ultralite ¥UltraLiteForAppForge¥win32¥*) が想定されます。
-rc [*directory*]	指定された *directory* にある Ultra Light エンジン・クライアント・コンポーネント *.DLL (たとえば *ulmvc10.dll*) を登録します。複数のアプリケーションが、登録された同じエンジン・クライアント *.DLL ファイルを参照できます。 *directory* を設定しなかった場合は、デフォルトの SQL Anywhere のインストール先ロケーション (*install-dir¥ultralite ¥UltraLiteForAppForge¥win32¥*) が想定されます。
-u	Ultra Light AppForge コンポーネントの登録を解除します。バージョン 10.0 の Ultra Light for AppForge コンポーネントによって、Ultra Light for MobileVB の以前のバージョンが置き換えられました。同じマシン上で複数のバージョンのコンポーネントを使用することはできません。そのため、バージョン 10.0 の Ultra Light for Appforge コンポーネントを登録する前に、以前のバージョンを登録解除する必要があります。

備考

ulafreg は Windows 実行プログラムです。

♦ **ulafreg 出力を保存するには、次の手順に従います。**

1. 実行プログラムが終了したら、[編集] – [コピー]をクリックして、ウィンドウに表示された情報をコピーします。
2. 任意のテキスト・エディタに出力を貼り付けます。
3. 出力を目的の場所に保存します。

ユーティリティを実行する必要があるかどうかがわからない場合は、次の手順で確認できます。

♦ **Ultra Light を環境に追加する必要があるかどうかを確認するには、次の手順に従います。**

1. 次のいずれかを行います。
   - 新しいプロジェクトに、Ultra Light 固有のネームスペースへの参照があるかどうかを確認します。
   - 正しい接続クラス (たとえば ULConnectionParms) が AppForge の使用可能なクラスのリストに表示されるかどうかを確認します。
2. 上の条件を満たさない場合は、AppForge を終了します。
3. 適切なオプションを指定して ulafreg ユーティリティを実行します。
4. AppForge を再起動します。

ulafreg を管理権限を使用して定期的なセッションで実行する場合、Vista では出力は新しいコンソール・ウィンドウに表示されるので、ulafreg コマンドが終了すると出力はすぐに破棄されます。その場合、ユーティリティの結果を表示することができません。この動作を回避するには、次の手順に従います。

参照

- 「Ultra Light エンジンユーティリティ (uleng10)」 216 ページ
- MobileVB：「MobileVB 設計環境への Ultra Light の追加」 『Ultra Light - AppForge プログラミング』
- Crossfire：「Crossfire 設計環境への Ultra Light の追加」 『Ultra Light - AppForge プログラミング』

Palm OS 用 Ultra Light HotSync コンジットのインストール・ユーティリティ (ulcond10)

コンジットが各データベースの HotSync 同期操作を管理できるように、各 Palm OS データベースをインストールし、HotSync コンジットに登録してください。このユーティリティを使用して、コンジットをアンインストールすることもできます。

構文

ulcond10 -c "*connection-string*" [*options*] *creator-ID*

オプション	説明
creator-ID	必須。コンジットを使用するアプリケーションの作成者 ID を設定します。指定した作成者 ID のコンジットがすでに存在している場合は、新しいコンジットと置き換えられます。
-a	-c オプションで設定した接続文字列に、データベース接続文字列を追加します。このオプションを使用すると、コンジットに複数のデータベースを登録できます。
-c "*connection-string*"	必須。*connection-string* の DBF パラメータで指定するデバイス上の Palm データベースに接続します。定義する接続文字列によって配備された Palm データベースがコンジットに登録され、接続文字列はコンジットの設定情報の一部として格納されます。 ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-d *filename*	プラグインの *.dll* ファイルの名前を設定します。 ユーティリティによって、*PluginDLL* という値でレジストリ・キー (*Software¥Sybase¥SQL Anywhere¥version¥Conduit¥creator-ID*) が 作成されます。*PluginDLL* の値は、設定した *filename* です。 空の文字列 (**-d""**) を指定すると、レジストリから既存のファイル名がクリアされます。
-n *name*	HotSync マネージャが表示する名前を設定します。デフォルト値は **Conduit** です。 このオプションと -u オプションを同時に使用しないでください。
-q	ユーティリティをクワイエット・モードで実行するように設定します。情報のバナーやバージョン番号が非表示になります。 ulcond10 を Windows Vista で適切な権限を使用して実行していて、エラーがレポートされなかった場合、ウィンドウはすぐにシャットダウンされます。それ以外の場合は、5 秒の遅延が適用されます。

オプション	説明
-QQ	ユーティリティを超クワイエット・モードで実行するように設定します。情報のバナー、バージョン番号、ステータス・メッセージ、エラー・メッセージが非表示になります。 ulcond10 を Windows Vista で適切な権限を使用して実行していた場合、ウィンドウはすぐにシャットダウンされます。
-u	作成者 ID のコンジットをアンインストールします。-u を指定しなかった場合は、コンジットがインストールされます。
-x *sync-parms*	セミコロンで区切った keyword=value の組み合わせのリストで、必要な同期パラメータを設定します。キーワードは大文字と小文字が区別されません。「拡張同期パラメータ」 238 ページを参照してください。 空の文字列 (**-x ""**) を指定すると、既存のパラメータがクリアされます。

備考

ulcond10 はコマンド・ライン・ユーティリティです。Windows Vista では、ulcond10 は昇格されて実行されます。実行プログラムのウィンドウは 5 秒間だけ閉じるのが遅れるので、出力を見ることができます。

HotSync コンジット・インストーラを実行するには、HotSync マネージャがコンピュータにインストールされている必要があります。

HotSync は、各同期がいつ実行され、インストールされている各コンジットが正しく動作したかどうかを記録します。HotSync ログ・ファイルは、Palm デスクトップ・インストール・ディレクトリのサブディレクトリ *User* にあります。

参照

- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「Ultra Light の Palm OS 用データ管理ユーティリティ (ULDBUtil)」 234 ページ
- 「エンド・ユーザのデスクトップへの Ultra Light HotSync コンジットの配備」 『Mobile Link - クライアント管理』
- 「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」 『Mobile Link - クライアント管理』
- 「Palm OS の HotSync」 『Mobile Link - クライアント管理』
- 「Palm 作成者 ID を登録する」 『Ultra Light - C/C++ プログラミング』

例

次のコマンドは、作成者 ID が **Syb2** であるアプリケーションの **CustDB** というコンジットをインストールします。次の例は CustDB サンプル・アプリケーションの設定です。

```
ulcond10 -c "DBF=custdb.udb;UID=DBA;PWD=sql" -n CustDB Syb2
```

次のコマンドは、同じ CustDB サンプル・アプリケーションのコンジットをアンインストールします。


```
ulcond10 -u Syb2
```

Ultra Light データベース作成ユーティリティ (ulcreate)

定義するプロパティで Ultra Light データベースを作成します。

構文

ulcreate **-c** "*connection-string*" [*options*] *new-database-file*

オプション	説明
-c "*connection-string*"	必須。*connection-string* の DBF パラメータまたは file_name パラメータで指定するデータベースを作成します。 ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。 接続文字列のパラメータとしてファイル名を指定しなかった場合は、コマンドの末尾で *new-database-file* で指定したファイルが確認されます。
-g *global-ID*	初期データベース ID を、指定する INTEGER 値に設定します。この初期値と分割サイズが、グローバル・オートインクリメント・カラムがある新しいローに使用されます。アプリケーションを配備するときには、Mobile Link サーバとの同期のために、各データベースに対して異なる ID 番号の範囲を割り当てます。
-o [*extended-options*]	Ultra Light データベースの拡張作成オプションを設定します。「拡張作成時オプション」 236 ページを参照してください。
-l	使用可能な照合順をリストして終了する。
-ol	利用可能なデータベースの拡張作成オプションをリストして終了します。「拡張作成時オプション」 236 ページを参照してください。
-p *creator-ID*	Palm OS の場合は必須です。指定する Ultra Light クライアント・アプリケーションの 4 文字の *creator-ID* でデータベースを作成します。
-q	クワイエット・モードで実行し、メッセージを表示しません。
-t *file*	信用されたパブリック・ルート証明書を含むファイルを指定します。この証明書は、サーバ認証に必要です。
-v	冗長メッセージを表示する
-y	データベース・ファイルが存在する場合、それを上書きする
-z *collation-sequence*	使用する照合のラベルを指定します。

オプション	説明
new-database-file	指定された名前でファイルを作成します。ユーザ ID (UID) やパスワード (PWD) のような初期データベース・パラメータを設定するための接続文字列を設定していない場合にかぎり、このスタンドアロン・ファイル名を使用してください。設定するスタンドアロン・ファイル名はプラットフォームに適切である必要があります。

備考

データベース・プロパティを設定しなかった場合は、大文字と小文字が区別されず、ユニコードを使用せず、照合順は現在のロケールに依存するデータベースが作成されます。

大文字と小文字の区別　データベースの大文字と小文字の区別の設定に関係なく、データベースのパスワードは常に大文字と小文字が区別されます。データベースの大文字と小文字の区別は、拡張オプション設定 case=respect を使用するかどうかで決まります。

照合について　データベース中のすべての文字列比較で使用される照合順です。Ultra Light では、照合にコード・ページとソート順が含まれます。-z を指定しなかった場合は、ulcreate が、デスクトップの現在のロケールに基づいて、使用する合理的な照合順を決定しようとします。

コマンド・プロンプトで ulcreate -l を実行すると、サポートされている照合と対応するコード・ページのリストが表示されます。

UTF-8 エンコード　UTF-8 エンコードを使用するかどうかは、デバイスのオペレーティング・システムで決まります。

Palm OS のデータベース　デスクトップに作成された Palm のデータベースは *.pdb* 拡張子で識別できる必要があります。ただし、データベースをデバイスに配備すると、拡張子は削除されます。ファイル名形式の詳細については、「Palm OS」 86 ページを参照してください。

エラー　このユーティリティはエラー・コードを返します。0 以外の値は処理に失敗したことを示します。

参照

- 「Ultra Light データベースの作成と設定」 53 ページ
- 「Ultra Light 接続パラメータでのファイル・パスの指定」 85 ページ
- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「サポートされている終了コード」 200 ページ
- 「サポートされている照合と代替照合」 『SQL Anywhere サーバ - データベース管理』
- 「Palm 作成者 ID を登録する」 『Ultra Light - C/C++ プログラミング』
- 「Ultra Light case プロパティ」 136 ページ
- 「Ultra Light での文字セットのエンコードに関するプラットフォーム要件」 64 ページ

例

大文字と小文字を区別せず、ユニコードを使用せず、照合順が現在のロケールに依存する、*test.udb* という Ultra Light データベースを作成します。

```
ulcreate test.udb
```

次のコマンドを実行すると、大文字と小文字を区別し、日付フォーマットと日付順が ISO に準拠する *test.udb* というデータベースが作成されます。

```
ulcreate -c DBF=test.udb -o case=respect -o date_format=YYYY-MM-DD -o date_order=YMD
```

暗号化され、暗号化キーが **afvc_1835** の *test.udb* というデータベースを作成します。

```
ulcreate -c "DBF=test.udb;DBKEY=afvc_1835"
```

Ultra Light エンジンユーティリティ (uleng10)

説明

32 ビット Windows デスクトップと Windows CE 上のアプリケーションからの同時 Ultra Light データベース接続を管理します。

備考

エンジンがサポートする同時接続の最大数は次のとおりです。

- Palm OS と Symbian OS の場合はデータベース数が 8、同時接続数が 16
- その他のすべてのプラットフォームの場合はデータベース数が 32、同時接続数が 64

Ultra Light エンジンのオプションはありません。

Ultra Light エンジンでは、起動時にコンソール・ウィンドウが表示されません。

- **uleng を Windows CE デバイスに配備するには、次の手順に従います。**

1. *uleng10.exe* と関連 **.dll* ファイルをコピーします。コピーする必要のある **.dll* ファイルには、データベースの暗号化や同期に必要な dll も含まれます。たとえば、*ulfips10.dll* (FIPS データ暗号化用) や *mlcrsafips10.dll* (FIPS 同期暗号化用) などがあります。
2. 配備の要件に従って、ファイルを適切なディレクトリに保存します。ほとんどの場合、コピー先ディレクトリは次のどれかです。
   - Windows CE ルート・ディレクトリ
   - *¥windows* ディレクトリ
   - 他の Ultra Light アプリケーション・ファイルのディレクトリ
3. 作成したアプリケーションが Ultra Light データベースに接続するときに、接続文字列に START 接続パラメータを含めて Ultra Light エンジンを起動します。

   次に例を示します。

   ```
   "DBF=wce:¥MyULFiles¥MyDB.udb;START=¥MyULFiles¥uleng10.exe"
   ```

参照

- 「Ultra Light データ管理コンポーネントの選択」 32 ページ
- 「Ultra Light エンジン停止ユーティリティ (ulstop)」 226 ページ
- 「Ultra Light START 接続パラメータ」 196 ページ

Ultra Light 情報ユーティリティ (ulinfo)

このユーティリティには、次の機能があります。

- Ultra Light データベースに関する情報を表示する。
- global_id または ml_remote_id データベース・オプションを変更またはクリアする。

構文

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

オプション	説明
-c "*connection-string*"	必須。*connection-string* の DBF パラメータまたは file_name パラメータで指定するデータベースに接続します。ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-g *ID*	初期グローバル・データベース ID を、指定する値に設定します。この値は、データベースでグローバル・オートインクリメント・カラムがあるすべての新しいローに使用されます。データベースでは、この基本値を使用して、追加する各ローかカラムまたはその両方に関連付けられる ID がオートインクリメントされます。 アプリケーションを配備するときには、Mobile Link サーバとの同期のために、各データベースに異なる ID 番号を割り当てる必要があります。
-or	読み込み専用モードでデータベースを開きます。Ultra Light によって、元のファイルのコピーが作成されます。このコピーを使用すると、データベースを変更しないでスクリプトをテストできます。コピーされたファイルの変更内容は終了時に破棄されます。 すでに CE デバイスに配備されているデータベースにデスクトップから直接接続している場合は、このオプションはサポートされません。
-q	クワイエット・モードで実行し、メッセージを表示しません。
-r *ID*	初期 ml_remote_id を、指定する値に設定します。新しい Ultra Light データベースでは、デフォルトで Mobile Link のリモート ID が NULL に設定されます。このデフォルトを使用することもできます。Ultra Light と dbmlsync の両方で、同期の開始時に Mobile Link のリモート ID がユニークなユーザ ID (UUID) に設定されます。
-rc	既存のリモート・データベース ID をクリアし、Mobile Link のリモート ID を NULL に設定します。
-v	冗長メッセージを表示します。指定するデータベースの内部構成に加えて現在のデータベース・プロパティを表示します。

備考

Ultra Light データベースを開くときに生成される警告メッセージは、-q オプションを使用しないかぎり、常に表示されます。

参照

- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「Ultra Light global_database_id オプション」 162 ページ
- 「Ultra Light でのグローバル・データベース ID の考慮事項」 74 ページ
- 「Ultra Light ml_remote_id オプション」 163 ページ
- 「Ultra Light でのリモート ID の考慮事項」 73 ページ

例

すでに同期されている *sample.udb* というファイルの基本的なデータベース内部構成を表示します。

```
ulinfo -c DBF=sample.udb

SQL Anywhere UltraLite Database Attribute Utility Version 10.0.0.xxxx Database name: cv_dbattr Disk
file 'run¥script¥cv_dbattr.udb'
Collation: 1252LATIN1
Number of Users: 1
  1. User: 'DBA'
Page size: 4096
Remote ID: JohnD
Global database ID: 1000
Global autoincrement usage: 0%
Number of tables: 3
Number of columns: 7
Number of publications: 3
Number of tables that will always be uploaded: 0
Number of tables that are never synchronized: 0
Number of primary Keys: 3
Number of foreign Keys: 0
Number of indexes: 0 Last synchronization completed successfully
Download occurred: 2005-12-07 15:01:28.615000
Upload OK
Upload rows not ignored
Partial download retained for subsequent sync
Actual MobiLink Authentication value: 1000
Authentication valid
Synchronization by publication number:
  1. Publication cv_sync
     Mask: 0x01
     Number of rows in next upload: 0
     Last download: 1900-01-01 00:00:00.000
  2. Publication cv_syncPub2
     Mask: 0x02
     Number of rows in next upload: 0
     Last download: 1900-01-01 00:00:00.000
  3. Publication cv_syncPub3
     Mask: 0x04
     Number of rows in next upload: 0
     Last download: 1900-01-01 00:00:00.000
```

CustDB.udb というファイルのデータベース内部構成を表示し、冗長メッセージを有効にしてデータベース・プロパティを表示します。

```
ulinfo -c DBF=CustDB.udb -v
```

```
SQL Anywhere UltraLite Database Attribute Utility Version 10.0.0.xxxx
Database information
Database name: custdb
Disk file 'C:¥Documents and Settings¥All Users¥Shared Documents¥SQL Anywhere 10¥Samples
¥UltraLite¥CustDB¥custdb.udb'
Collation: 1252LATIN1
Number of Users: 1
  1. User: 'DBA'
Page size: 4096
Default index maximum hash size: 8
Checksum level: 0
MobiLink Remote ID: not set
Global database ID: not set
Encryption: None
Character encoding set: 1252LATIN1
Case sensitive: OFF
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: 6
Number of columns: 16
Number of publications: 1
Number of tables that will always be uploaded: 0
Number of tables that are never synchronized: 1
Number of primary Keys: 6
Number of foreign Keys: 2
Number of indexes: 3
This database has not yet been synchronized.
Synchronization by publication number:
  1. Publication test
     Mask: 0x01
     Number of rows in next upload: 0
     Last download: 1900-01-01 00:00:00.000
Done.
```

sample.udb というファイルの ml_remote_id をクリアします。

```
ulinfo -c DBF=sample.udb -rc
```


Ultra Light データベース初期化ユーティリティ (ulinit)

既存の SQL Anywhere 統合データベースから Ultra Light リモート・データベースを作成します。

構文

ulinit -a "*SAconnection-string*" **-c** "*ULconnection-string*" -n *pubname* [*options*]

オプション	説明
-a *"SAconnection-string"*	必須。*SAconnection-string* で指定する SQL Anywhere データベースに接続します。ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-c *"ULconnection-string"*	必須。*connection-string* の DBF パラメータまたは file_name パラメータで指定するデータベースに接続します。ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-n *pubname*	必須。テーブルを Ultra Light データベース・スキーマに追加する。 *pubname* は、リファレンス・データベース内のパブリケーションを指定します。パブリケーション内のテーブルは、Ultra Light データベースに追加されます。これらのテーブルはリファレンス・データベースに存在する必要があります。ulinit では作成されません。 複数のパブリケーションからのテーブルを Ultra Light データベースに追加するには、オプションを複数回指定します。リファレンス・データベース内のすべてのテーブルを Ultra Light データベースに追加するには、**-n*** を指定します。
-o [*extended-options*]	Ultra Light データベースの拡張作成オプションを設定します。「拡張作成時オプション」 236 ページを参照してください。
-p *creator-ID*	Palm OS の場合は必須です。指定する Ultra Light クライアント・アプリケーションの 4 文字の *creator-ID* でデータベースを作成します。
-q	クワイエット・モードで実行し、メッセージを表示しません。
-s *pubname*	リファレンス・データベース内の指定するパブリケーションを使用して、同期の動作を設定します。複数の同期パブリケーションを指定するには、複数の -s オプションを指定します。 -s を指定しないと、Ultra Light リモートにはパブリケーションが指定されません。**-s*** と入力すると、リファレンス・データベース内のすべてのパブリケーションを使用して同期されます。 Mobile Link との同期用のパブリケーションを作成する方法については、「Ultra Light のパブリケーション」 『Mobile Link - クライアント管理』を参照してください。

オプション	説明
-t *file*	信用されたルート証明書を含むファイルを指定します。この証明書は、サーバ認証に必要です。
-w	警告を表示しません。

備考

SQL Anywhere リファレンス・データベースは、同期スクリプト、データベース設定 (照合順など)、Ultra Light データベース・ファイルのテーブル定義のソースとして機能します。基本的には、新しい Ultra Light データベースの設定に使用したスキーマ関連情報として機能します。作成したデータベースは、最初は空です。

♦ ulinit を使用して作成した空の Ultra Light データベースにデータを移植するには、次の手順に従います。

· 情報ソースとして別のデータベースを使用します。次の方法が可能です。

- ulload を実行して、リファレンス・データベースからデータをロードします。「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページを参照してください。
- 統合データベースとデータを同期します。「Ultra Light 同期ユーティリティ (ulsync)」 227 ページと「Ultra Light クライアント」 『Mobile Link - クライアント管理』を参照してください。

他のデータベース作成方法 SQL Anywhere リファレンス・データベースを使用しないで Ultra Light データベースを作成するには、次のいずれかの方法を使用します。

- SQL Anywhere 以外の RDBMS から Ultra Light データベースを初期化するには、Sybase Central の [同期モデル作成] ウィザードを使用します。このウィザードを実行すると、スキーマ情報を取得するために統合データベースに接続するように求められます。
- リファレンス・データベースから独立して設定できる空の Ultra Light データベースを作成するには、ulcreate ユーティリティまたは Ultra Light の [データベースの作成] ウィザードを使用します。

UTF-8 エンコード 通常、Ultra Light では、リファレンス・データベースで定義されている照合順が使用されます。ただし、デバイスでエンコードされた文字が必要な場合は、*extended-options* リストで utf8_encoding プロパティを設定することで、データベースを UTF-8 を使用してエンコードできます。

ただし、ulint ユーティリティで照合順を Ultra Light がサポートする照合順に制限する場合があります。コマンド・プロンプトで ulcreate -l を実行すると、サポートされている照合と対応するコード・ページのリストが表示されます。選択した照合順が Ultra Light でサポートされていない場合は、サポートされている照合順に変更してください。

Palm OS のデータベース　デスクトップに作成された Palm のデータベースは、*.pdb* 拡張子で識別できます。「Palm OS」 86 ページを参照してください。

ただし、このファイルをデバイスに配備すると、拡張子は削除されます。

参照

- 「Mobile Link のモデルの概要」 『Mobile Link - クイック・スタート』
- 「Ultra Light データベース作成ユーティリティ (ulcreate)」 213 ページ
- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ

例

TestPublication で定義されているテーブルを含む *customer.udb* というファイルを作成します。次に例を示します。

```
ulinit -a "DSN=dbdsn;UID=JimmyB;PWD=secret" -c DBF=customer.udb -n TestPublication
```

2 つの異なるパブリケーションを含む *customer.udb* というファイルを作成します。Pub1 には、優先的に同期するデータのサブセットが含まれ、Pub2 には残りのデータが含まれます。次に例を示します。

```
ulinit -a "DSN=dbdsn;UID=JimmyB;PWD=secret" -c DBF=customer.udb -n Pub1 -n Pub2 -s Pub1 -s Pub2
```

登録された作成者 ID を使用して *customer.udb* という Palm OS 用のファイルを作成します。次に例を示します。

```
ulinit -a "DSN=dbdsn;UID=JimmyB;PWd=secret" -c DBF=customer.udb.pdb -n TutCustomersPub
-p creator-id
```

Ultra Light データベースへの XML のロード・ユーティリティ (ulload)

このユーティリティには、次の機能があります。

- XML ファイルからデータをロードして新しいデータベースを作成する。
- 既存のデータベースにデータをロードする。

構文

ulload -c "*connection-string*" [*options*] *xml-file*

オプション	説明
-a	データとスキーマの定義を既存のデータベースに追加します。 Palm OS 用 (拡張子が *.pdb*) の既存のレコードベースのデータベースにデータを追加する場合は、-p オプションを使用しないでください。
-c "*connection-string*"	必須。*connection-string* の DBF パラメータまたは file_name パラメータで指定するデータベースに接続します。ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-d	データだけをロードし、XML ファイル入力内のスキーマ・データは無視します。
-f *directory*	XML ファイルのディレクトリを外部パスに設定します。
-g *ID*	初期データベース ID を、指定する INTEGER 値に設定します。この値は、データベースでグローバル・オートインクリメント・カラムがあるすべての新しいローに使用されます。データベースでは、この基本値を使用して、追加する各ローかカラムまたはその両方に関連付けられる ID がオートインクリメントされます。 アプリケーションを配備するときには、Mobile Link サーバとの同期のために、各データベースに異なる ID 番号を割り当てる必要があります。
-i	挿入されたローを次の同期時にアップロードする。デフォルトでは、このユーティリティによって挿入されたローは、同期時にアップロードされません。
-n	スキーマ・データだけをロードし、XML ファイル入力内のデータは無視します。
-o [*extended-options*]	Ultra Light データベースの拡張作成オプションを設定します。「拡張作成時オプション」 236 ページを参照してください。
-ol	利用可能なデータベースの拡張作成オプションをリストして終了します。「拡張作成時オプション」 236 ページを参照してください。

オプション	説明
-onerror *behavior*	XML ファイルからデータを読み出し中にエラーが起こった場合の事象を制御します。*behavior* には次のいずれかの値を定義できます。 ♦ **continue** エラーは無視され、ulload によって XML のロードが継続されます。 ♦ **prompt** 継続するかどうかを確認するプロンプトが表示されます。 ♦ **quit** XML のロードが停止し、エラーで終了します。これはデフォルトの動作です。 ♦ **exit** ulload が終了します。
-or	読み込み専用モードでデータベースを開きます。Ultra Light によって、元のファイルのコピーが作成され、このコピーを使用して、データベースを変更しないでスクリプトがテストされます。コピーされたファイルの変更内容は終了時に破棄されます。 すでに CE デバイスに配備されているデータベースにデスクトップから直接接続している場合は、このパラメータはサポートされません。
-p *creator-ID*	ロードするファイルから新しい Palm OS データベースを作成するだけの場合は必須です。指定する Ultra Light クライアント・アプリケーションの 4 文字の *creator-ID* でデータベースを作成します。 Palm OS 用 (拡張子が *.pdb*) の既存のレコードベースのデータベースにデータを追加する場合は、-a オプションと同時にこのオプションを使用しないでください。
-q	クワイエット・モードで実行し、メッセージを表示しません。
-s *file*	データベースのロードに使用された SQL 文を、指定する *file* に記録します。
-t *file*	信用されたルート証明書を含むファイルを指定します。この証明書は、サーバ認証に必要です。
-v	冗長メッセージを表示します。
-y	確認メッセージを表示しないで *xml-file* を上書きする
xml-file	データのロード元の XML ファイルの名前を指定します。

備考

指定したファイルが存在しなかった場合は、ulload ユーティリティによって Ultra Light データベース・ファイルが作成されます。

コマンド・ラインでオプションを設定するか、証明書を指定すると、ulload で処理される *xml-file* 内の設定より優先されます。

このユーティリティはエラー・コードを返します。0 以外の値は処理に失敗したことを示します。

Palm OS のデータベース　デスクトップに作成された Palm のデータベースは、*.pdb* 拡張子で識別できます。「Palm OS」 86 ページを参照してください。

ただし、このファイルをデバイスに配備すると、拡張子は削除されます。

参照

- 「Palm 作成者 ID を登録する」 『Ultra Light - C/C++ プログラミング』
- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「Ultra Light データベースの XML へのアンロード・ユーティリティ (ulunload)」 229 ページ
- 「サポートされている終了コード」 200 ページ
- 「Ultra Light global_database_id オプション」 162 ページ
- 「Ultra Light でのグローバル・データベース ID の考慮事項」 74 ページ

例

新しい Ultra Light データベース・ファイル *sample.udb* を作成し、*sample.xml* 内のデータをロードします。

```
ulload -c DBF=sample.udb sample.xml
```

sample.xml から既存のデータベース *sample.udb* にデータをロードし、エラーが発生した場合はプロンプトを表示します。

```
ulload -d -c DBF=sample.udb -onerror prompt sample.xml
```

test_data.xml というファイルから、Ultra Light によって作成される *sample.udb* というデータベースのコピーに XML をロードします。変更内容は終了時に破棄します。この方法で、XML データにエラーがないかどうかを確認し、エラーがあった場合は修正できます。データが正常にロードされたら、-or オプションを指定しないでコマンドを実行し、XML の更新内容を保持できます。

```
ulload -or -c DBF=sample.udb -a test_data.xml
```


Ultra Light エンジン停止ユーティリティ (ulstop)

説明

32 ビット Windows デスクトップと Windows CE の Ultra Light エンジンを停止します。

構文

ulstop

備考

開発中にエンジンを手動でシャットダウンする場合は ulstop を使用してください。本配備で ulstop を使用する必要は一般的にはありません。

このユーティリティにはオプションはありません。

参照

- 「Ultra Light データ管理コンポーネントの選択」 32 ページ
- 「Ultra Light エンジンユーティリティ (uleng10)」 216 ページ

Ultra Light 同期ユーティリティ (ulsync)

Ultra Light データベースを Mobile Link サーバと同期させます。これは、アプリケーション開発中に同期をテストするためのツールです。

構文

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

オプション	説明
-a *authenticate-parameters*	Mobile Link サーバ *authenticate_parameters* スクリプトで必要な値を設定します。このような値には、ユーザ名やパスワードなどがあります。必要な場合は、このオプションを 20 回まで繰り返すことができます。
-c "*connection-string*"	必須。*connection-string* の DBF パラメータまたは file_name パラメータで指定するデータベースに接続します。ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-e *sync-parms*	単一の拡張同期オプションを設定します。複数の -e オプションを使用すると、複数の拡張同期オプションを設定できます。キーワードは大文字と小文字が区別されません。Username と Version は必ず設定する必要があります。次に例を示します。 **-e Username=***MyName* **-e Version=***MyNum* 「拡張同期パラメータ」 238 ページを参照してください。
-k *stream-type*	暗号化されているか、暗号化されていない同期ストリームを指定します。*stream-type* は *tcpip*、*tls*、*http*、*https* のいずれかである必要があります。デフォルトのストリームは *tcpip* です。 Palm OS の場合は、Ultra Light データベースの作成時に RSA 証明書を格納している必要があります。その他のシステムでは、**-x trusted certificate=***path* オプションを使用して、または暗号化コードの一部として証明書ルックアップ・メカニズムがある名前のないファイルとして、証明書ファイルを指定できます。
-n	同期を実行しません。前回の同期の結果を表示するには、このオプションを -r と組み合わせて使用します。
-or	読み込み専用モードでデータベースを同期させます。Ultra Light によって、元のファイルのコピーが作成され、このコピーを使用して、データベースを変更しないでスクリプトがテストされます。コピーされたファイルの変更内容は終了時に破棄されます。 すでに CE デバイスに配備されているデータベースにデスクトップから直接接続している場合は、このパラメータはサポートされません。
-r	前回の同期の結果を表示します。
-q	クワイエット・モードで実行し、メッセージを表示しません。

オプション	説明
-v	同期の進行状況のメッセージを表示します。
-x *netopt-string*	セミコロンで区切った同期ネットワーク・プロトコル・オプション (ストリーム・パラメータ) とそれぞれの値のリストです。「Ultra Light 同期ストリームのネットワーク・プロトコルのオプション」『Mobile Link - クライアント管理』を参照してください。 デフォルト値は **host=localhost** です。

備考

sync-parms オプション (同期パラメータを設定) を **-x** *network protocol options* オプション (同期ストリーム・パラメータを設定) と混同しないでください。

このユーティリティはエラー・コードを返します。0 以外の値は処理に失敗したことを示します。

参照

- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「Ultra Light クライアント」 『Mobile Link - クライアント管理』
- 「サポートされている終了コード」 200 ページ
- 「Mobile Link ファイル転送ユーティリティ [mlfiletransfer]」 『Mobile Link - クライアント管理』

例

次のコマンドは、*myuldb.udb* というデータベース・ファイルを同期させます。Mobile Link ユーザ名は **remoteA** です。

```
ulsync -c DBF=myuldb.udb -k http -e "Username=remoteA;Version=2"
```

次のコマンドは、データベース・ファイル *myuldb.udb* の変更内容を、HTTPS 経由で、*c:¥certs¥rsa.crt* 証明書を使用してアップロードします。Mobile Link ユーザ名は **remoteB** です。

```
ulsync -c DBF=myuldb.udb -k https -x trusted_certificate=c:¥certs¥rsa.crt
-e "Username=remoteB;Version=2;UploadOnly=ON"
```

次のコマンドは、*synced.udb* というデータベース・ファイルについて、前回の同期の結果を表示します。

```
ulsync -n -r -c dbf=synced.udb
```

前回の同期の結果は、次のようにリストされます。

```
SQL Anywhere UltraLite Database Synchronize Utility Version 10.0
  Results of last synchronization:
  Succeeded
    Download timestamp: 2006-07-25 16:39:36.708000
    Upload OK
    No ignored rows
    Partial download retained
    Authentication value: 1000 (0x3e8)
```

Ultra Light データベースの XML へのアンロード・ユーティリティ (ulunload)

使用するオプションによって、次のいずれかをアンロードします。

- Ultra Light データベース全体を XML にアンロードする。
- Ultra Light データのすべてまたは一部だけを XML にアンロードする。
- Ultra Light スキーマだけを XML にアンロードする。
- Ultra Light スキーマだけを SQL 文としてアンロードする。

構文

ulunload -c "*connection-string*" [*options*] *output-file*

オプション	説明
-b *max-size*	XML ファイルに格納するカラム・データの最大サイズを設定します。デフォルトは 10 K です。**-b -1** を使用すると、すべてのデータが XML ファイルに保存されます (最大サイズが設定されません)。
-c "*connection-string*"	必須。*connection-string* の DBF パラメータまたは file_name パラメータで指定するデータベースに接続します。ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-d	データだけをアンロードし、データベース内のスキーマ情報は無視します。 ulload ユーティリティを使用して XML ファイルを新しいデータベースに再ロードする場合は、このオプションを使用しないでください。
-e *table*,...	指定する *table* 内のデータを除外します。複数のテーブルをカンマで区切って指定できます。次に例を示します。 -e mydbtable1,mydbtable5
-f *directory*	-b で指定した最大サイズよりも大きいデータを格納するディレクトリを設定します。デフォルトは、XML ファイルと同じディレクトリです。
-n	スキーマだけをアンロードし、データベース内のデータは無視します。

オプション	説明
-or	読み込み専用モードでデータベースを開きます。Ultra Light によって、元のファイルのコピーが作成されます。このコピーを使用すると、データベースを変更しないでスクリプトをテストできます。コピーされたファイルの変更内容は終了時に破棄されます。 すでに CE デバイスに配備されているデータベースにデスクトップから直接接続している場合は、このパラメータはサポートされません。
-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。
-s	スキーマだけを SQL Anywhere 互換の SQL 文としてアンロードします。データベース内のデータは無視します。
-t *table*,...	指定する *table* 内のデータだけをアンロードします。複数のテーブルをカンマで区切って指定できます。次に例を示します。 -t mydbtable2,mydbtable6
-v	冗長メッセージを表示します。
-x *owner*	テーブルが特定のユーザ ID の所有になるように出力します。このオプションは、-s オプションと同時に使用できます。
-y	確認メッセージを表示しないで *output-file* を上書きします。
output-file	必須。データベースのアンロード先のファイルの名前を設定します。-s オプションを使用すると、データベースは SQL 文としてアンロードされます。-s オプションを使用しなかった場合は、データベースは XML としてアンロードされます。

備考

デフォルトでは、データベース内のスキーマとデータを表す XML が出力されます。出力は、アーカイブに使用するか、Ultra Light データベースをすべてのリリース間で移植するために使用できます。

保持されるデータ　データベースのアンロード時に次のデータは保持されません。

- 同期ステータス、同期数、ロー削除。アンロードの前にデータベースを同期させる必要があります。
- Ultra Light ユーザのエントリ。

保持されたデータベース・オプションまたはプロパティを確認するには、ulload ユーティリティでデータベースを再ロードした後に ulinfo を実行します。

カラム・データのオーバフロー　カラム・データが、-b で指定した最大サイズを超えていた場合、オーバフローは **.bin* ファイルに保存されます。このファイルは、XML ファイルと同じディレクトリ、または -f で指定したディレクトリにあります。ファイルは次の命名規則に従います。

tablename-columname-rownumber.**bin**

テーブルの所有権の割り当て　-x オプションで Ultra Light テーブルに所有権を割り当てることができます。テーブルに所有者を割り当てるのは、出力の SQL 文を、SQL Anywhere データベースの作成または変更に使用する場合だけです。出力の SQL を、Ultra Light データベースの作成または変更に使用する場合は、このオプションを使用しないでください。

エラー　このユーティリティはエラー・コードを返します。0 以外の値は処理に失敗したことを示します。

参照

- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「サポートされている終了コード」 200 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- 「Ultra Light 情報ユーティリティ (ulinfo)」 217 ページ

例

sample.udb データベースを *sample.xml* ファイルにアンロードします。

```
ulunload -c DBF=sample.udb sample.xml
```

sample.udb データベースのデータを *sample.xml* ファイルにアンロードします。データベースが存在する場合は、上書きします。

```
ulunload -c DBF=sample.udb -d -y sample.xml
```


Ultra Light 古いデータベースのアンロード・ユーティリティ (ulunloadold)

Ultra Light バージョン 8.0.2 のデータベースを 9.0.x データベースにアンロードするか、スキーマ・ファイル (*.*usm*) を XML ファイルにアンロードするか、両方を実行します。

構文

ulunloadold -c "*connection-string*" [*options*] *xml-file*

オプション	説明
-b *max-size*	XML ファイルに格納するカラム・データの最大サイズを設定します。デフォルトは 10 K です。**-b -1** を使用すると、すべてのデータが XML ファイルに保存されます (最大サイズが設定されません)。
-c "*connection-string*"	必須。*connection-string* の DBF パラメータまたは file_name パラメータで指定するデータベースに接続します。ユーザ ID とパスワードの両方を指定しなかった場合は、デフォルトの UID **DBA** と PWD **sql** が使用されます。
-f *directory*	-b で指定した最大サイズよりも大きいデータを格納するディレクトリを設定します。デフォルトは、XML ファイルと同じディレクトリです。
-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	確認メッセージを表示しないで *xml-file* を上書きします。
xml-file	データのアンロード先の XML ファイルの名前を設定します。

備考

このユーティリティで Ultra Light バージョン 10 のデータベースはアンロードしないでください。Ultra Light バージョン 10 のデータベースには ulunload ユーティリティを使用します。データベースの Ultra Light 10 バージョンを作成するには、ulload で *xml-file* をロードします。

保持されるデータ　データベースのアンロード時に次のデータは保持されません。

- 同期ステータス、同期数、ロー削除。アンロードの前にデータベースを同期させる必要があります。
- Ultra Light ユーザのエントリ。

保持されたデータベース・オプションまたはプロパティを確認するには、ulload ユーティリティでデータベースを再ロードした後に ulinfo を実行します。

カラム・データのオーバフロー　カラム・データが、-b で指定した最大サイズを超えていた場合、オーバフローは **.bin* ファイルに保存されます。このファイルは、XML ファイルと同じディレクトリ、または -f で指定したディレクトリにあります。ファイルは次の命名規則に従います。

tablename-columnname-rownumber.**bin**

エラー　このユーティリティはエラー・コードを返します。0 以外の値は処理に失敗したことを示します。

参照

- 「Ultra Light の接続文字列パラメータのリファレンス」 169 ページ
- 「Ultra Light データベースへの XML のロード・ユーティリティ (ulload)」 223 ページ
- 「Ultra Light データベースの XML へのアンロード・ユーティリティ (ulunload)」 229 ページ
- 「Ultra Light 情報ユーティリティ (ulinfo)」 217 ページ

例

Ultra Light 8.0.x のスキーマ・ファイル *dbschema8.usm* を Ultra Light バージョン 10 のデータベース *db.udb* にアップグレードするには、次の 2 つのコマンドが必要です。

```
ulunloadold -c SCHEMA_FILE=dbschema8.usm dbschema.xml
```

```
ulload -c DBF=db.udb dbschema.xml
```

Palm OS 用の Ultra Light バージョン 9.0.x のデータベース *palm9db.pdb* を Ultra Light バージョン 10 のデータベース *palm10db.pdb* にアップグレードするには、次の 2 つのコマンドが必要です。

```
ulunloadold -c DBF=palm9db.pdb dbdata.xml
```

```
ulload -c DBF=palm10db.pdb -p Syb dbdata.xml
```


Ultra Light の Palm OS 用データ管理ユーティリティ (ULDBUtil)

説明

ULDBUtil をデバイスにインストールしたら、Palm OS 用の Ultra Light データベースでこのユーティリティを使用して次の機能を実行できます。

- デバイスを複数のユーザで共有している場合にデータベースをデバイスから削除する。ファイルを削除すると、領域を節約し、プライバシーを守ることができます。このデータベースを再インストールするか、このデータベースから新しい空のデータベースを作成できます。
- 次の同期時にデータベースをバックアップする。この機能を使用すると、初期同期を実行してから、データベースをバックアップできます。このデータベースを他のデバイスに配備すると、各デバイスで初期同期を実行する必要がなくなります。

備考

このユーティリティは次のファイルとしてインストールされます。

install-dir¥UltraLite¥Palm¥68k¥ULDBUtil.prc

♦ Palm OS デバイスから Ultra Light アプリケーションのデータを削除するには、次の手順に従います。

1. ULDBUtil に切り替えます。
2. デバイスに拡張カードがある場合は、アプリケーション・データベース・ファイルを削除するメディアを選択します (内部／外部ボリュームまたはレコードベース)。
3. Ultra Light バージョン 10 アプリケーションとデータベースのリストから、アプリケーションを選択します。
4. Palm デバイスで [削除] をタップして、データを削除します。

♦ Palm OS デバイスからデスクトップにデータベースをバックアップするには、次の手順に従います。

1. ULDBUtil に切り替えます。
2. [バックアップ] オプションを選択して、次の同期時にデータベースをバックアップすることを HotSync に指定します。このオプションは、バックアップのたびに選択する必要があります。

ヒント
自動バックアップを有効にして、同期のたびにこのオプションを選択しないで済むようにするには、PALM_ALLOW_BACKUP パラメータを使用します。

参照

- 「Palm OS 用 Ultra Light HotSync コンジットのインストール・ユーティリティ (ulcond10)」 210 ページ
- 「Ultra Light PALM_ALLOW_BACKUP 接続パラメータ」 191 ページ

サポートされている拡張オプション

Ultra Light のコマンド・ライン・ユーティリティにはパラメータを指定できます。パラメータには、次の 2 種類があります。

- データベース作成オプション。
- 同期オプション。

参照

- 「拡張作成時オプション」 236 ページ
- 「拡張同期パラメータ」 238 ページ

拡張作成時オプション

ulcreate、ulinit、ulload などのデータベース作成ユーティリティでサポートされている拡張オプションは、新しいデータベースを設定するときに使用できる作成時のプロパティと同じです。データベースの拡張作成時オプションは、次の 2 つの方法のいずれかで定義できます。

- **拡張オプションの前に -o を付ける**　たとえば、ulcreate を使用して、大文字と小文字を区別し、ISO 互換の日付フォーマットと日付順を使用するデータベースを作成するには、次のようにコマンドを実行します。

  ```
  ulcreate -o case=respect -o date_format=YYYY-MM-DD
   -o date_order=YMD
  ```

- **オプションから作成文字列をアセンブルする**　作成パラメータのアセンブリを「**作成文字列**」といいます。作成文字列は、各パラメータ名と値をセミコロンで区切り、1 行で入力します。

  ```
  parameter1=value1;parameter2=value2
  ```

  Ultra Light ランタイムによって、作成時パラメータが作成文字列にアセンブルされていることが確認されてから、その作成文字列を使用してデータベースが作成されます。たとえば、ulcreate ユーティリティを使用する場合は、次の接続文字列を使用して、大文字と小文字を区別し、ISO 互換の日付フォーマットと日付順を使用する新しいデータベースを作成します。

  ```
  ulcreate - o "case=respect;date_format=YYYY-MM-DD;
  date_order=YMD"
  ```

データベース作成プロパティは常に作成時に設定する必要があります。ファイルの作成後にこれらのプロパティを変更することはできません。

プロパティ・リスト	説明
case	Ultra Light データベースの文字列を比較するときに大文字と小文字を区別するかどうかを設定します。「Ultra Light での大文字と小文字の区別の考慮事項」 65 ページと「Ultra Light case プロパティ」 136 ページを参照してください。

プロパティ・リスト	説明
checksum_level	データベースのチェックサム検証のレベルを設定します。「チェックサムを使用したページの整合性の確認」 69 ページと「Ultra Light checksum_level プロパティ」 137 ページを参照してください。
collation	Ultra Light データベースで使用される照合順を設定します。このプロパティを UTF-8 プロパティと組み合わせて設定するかどうかで、データベースの文字セットが決まります。「Ultra Light での文字の考慮事項」 63 ページ、「Ultra Light collation プロパティ」 138 ページ、および「Ultra Light utf8_encoding プロパティ」 158 ページを参照してください。
date_format	日付がデータベースから取り出されるときのデフォルトの文字列フォーマットを設定します。「Ultra Light での日付の考慮事項」 66 ページと「Ultra Light date_format プロパティ」 139 ページを参照してください。
date_order	年、月、日の日付の順序を解釈する方法を制御します。「Ultra Light での日付の考慮事項」 66 ページと「Ultra Light date_order プロパティ」 142 ページを参照してください。
fips	Certicom 認定暗号化アルゴリズムを使用して AES FIPS 準拠データの暗号化を制御します。FIPS エンコードは、強力な暗号化の一種です。「Ultra Light でのセキュリティの考慮事項」 70 ページと「Ultra Light fips プロパティ」 143 ページを参照してください。
max_hash_size	デフォルトのインデックス・ハッシュ・サイズをバイト単位で設定します。「Ultra Light でのインデックス・パフォーマンスの考慮事項」 64 ページと「Ultra Light max_hash_size プロパティ」 145 ページを参照してください。
nearest_century	文字列から日付への変換で、2 桁の年の解釈を制御します。「Ultra Light での基準年の変換の考慮事項」 67 ページと「Ultra Light nearest_century プロパティ」 146 ページを参照してください。
obfuscate	データベース内のデータを難読化するかどうかを制御します。難読化は単純暗号化です。「Ultra Light でのセキュリティの考慮事項」 70 ページと「Ultra Light obfuscate プロパティ」 147 ページを参照してください。
page_size	データベース・ページ・サイズを定義します。「Ultra Light でのページ・サイズの考慮事項」 68 ページと「Ultra Light page_size プロパティ」 148 ページを参照してください。
precision	計算結果が小数の場合の最大桁数を指定します。「Ultra Light での小数点の位置の考慮事項」 68 ページと「Ultra Light precision プロパティ」 150 ページを参照してください。
scale	計算結果が最大 precision にトランケートされる場合の、小数点以下の最小桁数を指定します。「Ultra Light での小数点の位置の考慮事項」 68 ページと「Ultra Light scale プロパティ」 151 ページを参照してください。
time_format	データベースから取り出した時刻のフォーマットを設定します。「Ultra Light での時刻の考慮事項」 66 ページと「Ultra Light time_format プロパティ」 153 ページを参照してください。

プロパティ・リスト	説明
timestamp_format	データベースから取り出したタイムスタンプのフォーマットを設定します。「Ultra Light でのタイムスタンプの考慮事項」 66 ページと「Ultra Light timestamp_format プロパティ」 154 ページを参照してください。
timestamp_increment	Ultra Light でのタイムスタンプのトランケート方法を指定します。「Ultra Light でのタイムスタンプの考慮事項」 66 ページと「Ultra Light timestamp_increment プロパティ」 156 ページを参照してください。
utf8_encoding	ユニコード用の 8 ビット・マルチバイト・エンコードである UTF-8 フォーマットでデータをエンコードします。「Ultra Light での文字の考慮事項」 63 ページと「Ultra Light utf8_encoding プロパティ」 158 ページを参照してください。

参照

- 「Ultra Light で使用する作成時のデータベース・プロパティの選択」 61 ページ

拡張同期パラメータ

拡張同期パラメータは、ulsync ユーティリティのコマンド・ラインで、使用する他のすべてのコマンド・ライン・オプションを定義した後に指定します。複数の -e オプションを使用すると、複数のパラメータを定義できます。セミコロンで区切られたキーワード文字列は構築できません。キーワードは大文字と小文字が区別されません。

オプション名	説明
DownloadOnly (ブール値)	同期がダウンロード専用であることを指定します。「Download Only 同期パラメータ」 『Mobile Link - クライアント管理』を参照してください。
Newpassword (文字列)	新しいパスワードを指定します。「New Password 同期パラメータ」 『Mobile Link - クライアント管理』を参照してください。
Password (文字列)	サブスクリプションが同期されている Mobile Link クライアントのパスワードを指定します。「Password 同期パラメータ」 『Mobile Link - クライアント管理』を参照してください。
Ping (ブール値)	Ultra Light クライアントと Mobile Link サーバ間の通信を確認します。「Ping 同期パラメータ」 『Mobile Link - クライアント管理』を参照してください。
Publication (文字列)	同期させるパブリケーションを指定します。「publication 同期パラメータ」 『Mobile Link - クライアント管理』を参照してください。
PublicationMask (整数)	最終ダウンロード時間を取得するパブリケーションのセットを指定します。このセットはマスクとして提供されます。「PublicationMask 同期パラメータ」 『Mobile Link - クライアント管理』を参照してください。

オプション名	説明
SendColumnNames (ブール値)	ローを直接処理します。「Send Column Names 同期パラメータ」『Mobile Link - クライアント管理』を参照してください。
SendDownloadACK	クライアントからサーバにダウンロード確認が送信されるように指定する。「Send Download Acknowledgment 同期パラメータ」『Mobile Link - クライアント管理』を参照してください。
TableOrder (文字列)	デフォルトのテーブルの順序 (外部キーに基づく) が適切でない場合に、優先同期に必要なテーブルの順序を設定します。「Table Order 同期パラメータ」『Mobile Link - クライアント管理』を参照してください。
UploadOnly (ブール値)	同期がアップロード専用であることを指定します。「Upload Only 同期パラメータ」『Mobile Link - クライアント管理』を参照してください。
Username (文字列)	必須。統合データベースのユーザを定義します。「User Name 同期パラメータ」『Mobile Link - クライアント管理』を参照してください。
Version (文字列)	必須。統合データベースのバージョンを定義します。「Version 同期パラメータ」『Mobile Link - クライアント管理』を参照してください。

第 11 章

Ultra Light システム・テーブルのリファレンス

目次

Ultra Light のシステム・テーブル

すべての Ultra Light データベースのスキーマは、多数のシステム・テーブルに記述されています。Ultra Light ではテーブルの所有権がサポートされていないので、Ultra Light の 4 つすべてのユーザ ID でシステム・テーブルを使用できます。

これらのテーブルの内容は、Ultra Light によってのみ変更できます。したがって、テーブルの内容の変更に UPDATE、DELETE、INSERT コマンドは使用できません。また、ALTER TABLE と DROP コマンドを使って、これらのテーブルの構造を変更することもできません。

これらのテーブルの内容は Sybase Central でブラウズできます。

♦ Sybase Central でシステム・オブジェクトを表示または非表示にするには、次の手順に従います。

1. データベースに接続します。
2. データベースのオブジェクトをブラウズします。
3. コンテンツのウィンドウ枠を右クリックし、[システム・オブジェクトを表示] または [システム・オブジェクトの非表示] を選択します。
4. [テーブル] フォルダをクリックし、使用可能なテーブルをブラウズします。

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 1 つのテーブルに関する記述です。

カラム

カラム名	説明	型	データ制約	整合性制約
column_count	テーブル内のカラム数	UNSIGNED INT	NOT NULL	
index_count	テーブル内のインデックス数	UNSIGNED INT	NOT NULL	
indexcol_count	テーブル内のすべてのインデックスのカラムの合計数	UNSIGNED INT	NOT NULL	
map_handle	内部でのみ使用	UNSIGNED INT	NOT NULL	
table_name	テーブルの名前	VARCHAR (128)	NOT NULL	
object_id	テーブルのユニークな識別子	UNSIGNED INT	NOT NULL	プライマリ・キー

カラム名	説明	型	データ制約	整合性制約
sync_type	Mobile Link による同期に使用。**no_sync** (非同期)、**all_sync** (すべてのローの同期)、**normal_sync** (変更されたローのみ同期) のいずれか	VARCHAR(128)	NOT NULL	
table_name	テーブルの名前	VARCHAR(128)	NOT NULL	
table_type	**sys** (システム・テーブル) または **user** (通常のテーブル) のいずれか	TINYINT	NOT NULL	
tpd_handle	内部でのみ使用	UNSIGNED INT	NOT NULL	

syscolumn システム・テーブル

syscolumn システム・テーブルの各ローは 1 つのカラムに関する記述です。

カラム

カラム名	説明	型	データ制約	整合性制約
column_name	カラムのユニークな識別子	VARCHAR(128)	NOT NULL	
default	このカラムのデフォルト値。例：autoincrement	VARCHAR(128)		
domain	カラムのドメインを示す列挙値	UNSIGNED INT	NOT NULL	
domain_info	サイズが可変のドメインで使用	SMALLINT	NOT NULL	
nulls	カラムのデフォルトに NULL が許可されるかどうか	CHAR(1)	NOT NULL	
object_id	カラムのユニークな識別子	UNSIGNED INT	NOT NULL	プライマリ・キー
table_id	カラムが属するテーブルの識別子	UNSIGNED INT	NOT NULL	プライマリ・キー 外部キーは systable 内の object_id を参照する。

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 1 つのインデックスに関する記述です。

カラム

カラム名	説明	型	データ制約	整合性制約
check_on_commit	すべての外部キーに一致するプライマリ・ローがあることを確認するために参照整合性をいつチェックするかを示す。型が **foreign** の場合にのみ必要	UNSIGNED INT		
index_name	インデックスのユニークな識別子	UNSIGNED INT	NOT NULL	プライマリ・キー 外部キーは sysindex を参照する。
ixcol_count	インデックス内のカラム数	UNSIGNED INT	NOT NULL	
nullable	型が **foreign** の場合にのみ必要。NULL が許可されるかどうかを示す	BIT		
object_id	インデックスのユニークな識別子	UNSIGNED INT	NOT NULL	
primary_index_id	型が **foreign** の場合にのみ必要。プライマリ・インデックスの識別子のリスト	UNSIGNED INT		
primary_table_id	型が **foreign** の場合にのみ必要。プライマリ・テーブルの識別子のリスト	UNSIGNED INT		
root_handle	内部でのみ使用	UNSIGNED INT	NOT NULL	
table_id	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	UNSIGNED INT	NOT NULL	外部キーは systable を参照する

カラム名	説明	型	データ制約	整合性制約
type	インデックスの型。次のいずれかです。 ◆ **primary** ◆ **foreign** ◆ **key** ◆ **unique** ◆ **index**	SMALLINT (10)	NOT NULL	
hash_size	インデックスのハッシュに使用されるハッシュ・サイズ	SHORT		

参照

- 「sysixcol システム・テーブル」 245 ページ

sysixcol システム・テーブル

sysixcol システム・テーブルの各ローは、sysindex に含まれるインデックスの 1 カラムに関する記述です。

カラム

カラム名	説明	型	データ制約	整合性制約
column_id	インデックス対象カラムのユニークな識別子	UNSIGNED INT	NOT NULL	外部キーは syscolumn を参照する
index_id	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	UNSIGNED INT	NOT NULL	プライマリ・キー 外部キーは sysindex を参照する。
order	インデックス内のカラムが昇順 (A) か、降順 (D) かを示す。	CHAR(1)	NOT NULL	
sequence	インデックス内のカラムの順序	SMALLINT	NOT NULL	プライマリ・キー
table_id	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	UNSIGNED INT	NOT NULL	プライマリ・キー

参照

- 「sysindex システム・テーブル」 244 ページ

syspublication システム・テーブル

syspublication システム・テーブルの各ローはパブリケーションに関する記述です。

カラム

カラム名	説明	型	データ制約	整合性制約
download_timestamp	最後のダウンロードの時刻	UNSIGNED INT	NOT NULL	
last_sync	アップロードの進行状況の追跡に使用	UNSIGNED BIGINT	NOT NULL	
publication_id	パブリケーションのユニークな識別子	UNSIGNED INT	NOT NULL	プライマリ・キー
publication_name	パブリケーションの名前	CHAR(128)	NOT NULL	

参照

- 「sysarticle システム・テーブル」 246 ページ

sysarticle システム・テーブル

sysarticle 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に関する記述です。

カラム

カラム名	説明	型	データ制約	整合性制約
publication _id	このアーティクルが属するパブリケーションの識別子	UNSIGNED INT	NOT NULL	プライマリ・キー 外部キーは syspublication を参照する。
table_id	パブリケーションに属するテーブルの識別子	UNSIGNED INT	NOT NULL	プライマリ・キー 外部キーは systable 内の object_id を参照する。

カラム名	説明	型	データ制約	整合性制約
where_expr	ローをフィルタする述部 (オプション)	TINY INT		

参照

- 「syspublication システム・テーブル」 246 ページ

sysuldata システム・テーブル

sysuldata システム・テーブルの各ローは、オプションとプロパティの名前と値の組み合わせを示します。

カラム

カラム名	説明	型	データ制約	整合性制約
long_setting	長い値用の BLOB	LONGBINARY		
name	プロパティ名	VARCHAR(128)	NOT NULL	プライマリ・キー
setting	プロパティの値	VARCHAR(128)		
type	**sys** (内部)、**opt** (オプション)、または **prop** (プロパティ) のいずれか	VARCHAR(128)	NOT NULL	プライマリ・キー

参照

- 「Ultra Light データベース設定のリファレンス」 135 ページ

パート V. Ultra Light SQL リファレンス

パート では、Ultra Light SQL のリファレンスです。Ultra Light SQL は、SQL Anywhere データベースでサポートされている SQL のユニークなサブセットです。

第 12 章

Ultra Light の SQL 要素のリファレンス

目次

Ultra Light のキーワード

各 SQL 文には 1 つまたは複数のキーワードが含まれています。SQL 文のキーワードでは大文字と小文字を区別しませんが、このマニュアルではキーワードを大文字で表記します。キーワードの中には、二重引用符で囲まないと識別子として使用できないものがあります。このようなキーワードを、「**予約語**」と呼びます。「予約語」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

注意
Ultra Light では、ここに示す予約語の一部だけがサポートされています。ただし、念のため、SQL Anywhere のすべての予約語が常に Ultra Light の予約語でもあると考えてください。

Ultra Light の識別子

「**識別子**」は、ユーザ ID、テーブル、カラムなど、データベースのオブジェクト名を表します。識別子の最大長は 128 バイトです。

次のいずれかの条件と一致した場合に、識別子を二重引用符で囲みます。

- 識別子にスペースが含まれる。
- 識別子の先頭文字がアルファベット文字ではない。データベースの照合順は、どの文字がアルファベットまたは数字として扱われるかを指定する。
- 識別子に予約語が含まれる。「予約語」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
- 識別子にアルファベット文字や数字以外の文字が含まれる。

1 つの円記号は、エスケープ文字として使用される場合のみ識別子で使用できます。

Ultra Light の文字列

文字列を使用して、文字データをデータベースに格納します。Ultra Light は、SQL Anywhere と同じ文字列の規則をサポートしています。文字列の比較結果や文字列のソート順は、データベース、文字セット、照合順における大文字と小文字の区別によって決まります。これらのプロパティは、データベースの作成時に設定されます。

参照

- 「文字列」『SQL Anywhere サーバ - SQL リファレンス』
- 「Ultra Light での文字の考慮事項」 63 ページ

Ultra Light のコメント

コメントは、SQL 文または文ブロックに説明テキストを付加するために使用します。Ultra Light ランタイムは、コメントを実行しません。

Ultra Light では次のコメント・インジケータを使用できます。

- **-- (二重ハイフン)** データベース・サーバは、この行の残りの文字を無視します。これは、SQL/2003 のコメント・インジケータです。
- **// (二重スラッシュ)** 二重スラッシュは、二重ハイフンと同じ意味です。
- **/* … */ (スラッシュ – アスタリスク)** 2 つのコメント・マーカの間にある文字は、すべて無視されます。2 つのコメント・マーカは、同じ行にあっても、別の行にあってもかまいません。このスタイルで示されたコメントは、ネストできます。このスタイルは、C スタイル・コメントとも呼ばれます。

> **注意**
> Ultra Light では、SQL ブロックにコメントを追加する方法としてパーセント記号 (%) を使用することはサポートしていません。

例

- 次に、二重ハイフンのコメントの使用例を示します。

```
CREATE TABLE borrowed_book (
  loaner_name CHAR(100)     PRIMARY KEY,
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  date_returned       DATE,
  book              CHAR(20)
  FOREIGN KEY book REFERENCES library_books (isbn),
)
--This creates a table for a library database to hold information on borrowed books.
--The default value for date_borrowed indicates that the book is borrowed on the day the entry is
made.
--The date_returned column is NULL until the book is returned.
```

- 次に、C スタイル・コメントの使用例を示します。

```
CREATE TABLE borrowed_book (
  loaner_name CHAR(100)     PRIMARY KEY,
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  date_returned       DATE,
  book              CHAR(20)
  FOREIGN KEY book REFERENCES library_books (isbn),
)
/* This creates a table for a library database to hold information on borrowed books.
The default value for date_borrowed indicates that the book is borrowed on the day the entry is
made.
The date_returned column is NULL until the book is returned. */
```


Ultra Light の数値

数値を使用して、数値データをデータベースに格納します。数値は次のようなデータです。

- 任意の数字の並び
- 小数部が続く
- オプションでマイナス記号 (-) またはプラス記号 (+) を含む
- E の後に指数値が続く

たとえば、次に示す数値はすべて Ultra Light でサポートされています。

42

-4.038

.001

3.4e10

1e-10

Ultra Light の NULL 値

SQL Anywhere と同様に、NULL はすべてのデータ型のすべての有効な値とは異なる特殊な値です。ただし、NULL 値はすべてのデータ型で使用できます。NULL は、情報が不明 (値なし) であるか、該当しないことを表すために使用します。「NULL 値」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

Ultra Light の特別値

特別値は、式や、テーブル作成時のカラムのデフォルトに使用できます。

CURRENT DATE 特別値

現在の年、月、日を返します。

データ型

DATE

備考

返される日付は、SQL 文が Ultra Light ランタイムで実行されたときのシステム・クロックの読み取り値が基準になります。CURRENT DATE を次のいずれかの状態で使用すると、すべての値は、個別のクロック読み取り値が基準になります。

- 同一文で CURRENT DATE を複数回使用
- 同一文で CURRENT DATE を CURRENT TIME または CURRENT TIMESTAMP と組み合わせて使用
- 同一文で CURRENT DATE を NOW 関数または GETDATE 関数と組み合わせて使用

参照

- 「Ultra Light の式」 274 ページ
- 「GETDATE 関数 [日付と時刻]」 328 ページ
- 「NOW 関数 [日付と時刻]」 348 ページ

CURRENT TIME 特別値

現在の時、分、秒 (小数位あり) で構成される時刻を返します。

データ型

TIME

備考

秒は小数第 6 位まで格納されます。現在時刻の精度はシステム・クロックの精度によって制限されます。

返される日付は、SQL 文が Ultra Light ランタイムで実行されたときのシステム・クロックの読み取り値が基準になります。CURRENT TIME を次のいずれかと組み合わせて使用すると、すべての値は、個別のクロック読み取り値が基準になります。

- 同一文で CURRENT TIME を複数回使用

- 同一文で CURRENT TIME を CURRENT DATE または CURRENT TIMESTAMP と組み合わせて使用
- 同一文で CURRENT TIME を NOW 関数または GETDATE 関数と組み合わせて使用

参照

- 「Ultra Light の式」 274 ページ
- 「GETDATE 関数 [日付と時刻]」 328 ページ
- 「NOW 関数 [日付と時刻]」 348 ページ

CURRENT TIMESTAMP 特別値

CURRENT DATE と CURRENT TIME を結合して、TIMESTAMP 値を形成します。年、月、日、時、分、秒、秒の小数位で構成されます。

データ型

TIMESTAMP

備考

秒は小数第 3 位まで格納されます。精度はシステム・クロックの精度によって制限されます。

DEFAULT CURRENT TIMESTAMP で宣言されたカラムには、ユニークな値が入るとはかぎりません。

CURRENT TIMESTAMP が返す情報は、GETDATE 関数と NOW 関数が返す情報と同じです。

CURRENT_TIMESTAMP は、CURRENT TIMESTAMP と同じです。

返される日付は、SQL 文が Ultra Light ランタイムで実行されたときのシステム・クロックの読み取り値が基準になります。CURRENT TIMESTAMP を次のいずれかと組み合わせて使用すると、すべての値は、個別のクロック読み取り値が基準になります。

- 同一文で CURRENT TIMESTAMP を複数回使用
- 同一文で CURRENT TIMESTAMP を CURRENT DATE または CURRENT TIME と組み合わせて使用
- 同一文で CURRENT TIMESTAMP を NOW 関数または GETDATE 関数と組み合わせて使用

参照

- 「CURRENT TIME 特別値」 258 ページ
- 「Ultra Light の式」 274 ページ
- 「NOW 関数 [日付と時刻]」 348 ページ
- 「GETDATE 関数 [日付と時刻]」 328 ページ
- 「NOW 関数 [日付と時刻]」 348 ページ

SQLCODE 特別値

SQLCODE 特別値が評価された時点の値です。

データ型

String

備考

SQLCODE 値は各文の後に設定されます。SQLCODE をチェックして、文の実行が成功したかどうかを確認することができます。

参照

- 「Ultra Light の式」 274 ページ
- SQL Anywhere 10 - エラー・メッセージ『SQL Anywhere 10 - エラー・メッセージ』.

例

SELECT 文を使用して、結果セットから新しいローをフェッチしようとするたびにエラー・コードを生成します。例：SELECT a, b, SQLCODE FROM MyTable

Ultra Light の日付と時刻

日付と時刻の関数の多くは、日付と時刻の単位で構成される日付を使用します。Ultra Light と SQL Anywhere では同じ日付の単位がサポートされています。「日付の単位」 298 ページを参照してください。

Ultra Light のデータ型

Ultra Light SQL では、次のデータ型を使用できます。

- integer データ型
- decimal データ型
- 浮動小数点データ型
- 文字データ型
- バイナリ・データ型
- 日付／時刻データ型

注意
Ultra Light SQL では、ドメイン (ユーザ定義のデータ型) はサポートされていません。

注意
LONGVARCHAR データ型と LONGBINARY データ型は連結できません。「文字列演算子」 288 ページを参照してください。

ホスト変数は、サポートされる任意のデータ型について作成できます。Ultra Light は、SQL Anywhere で使用できるデータ型のサブセットをサポートします。

次のデータ型は、Ultra Light データベースでサポートされている SQL データ型です。

データ型	説明
BIT	ブール値 (0 または 1)。「BIT データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
{ **CHAR**|**CHARACTER**} **(** *max-length* **)**	*max-length* の文字データです。範囲は 1 ー 32767 バイトです。「CHAR データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。 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。
VARCHAR(*max-length* **)**	VARCHAR は、最大長 *max-length* の可変長の文字データに使用します。「VARCHAR データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

データ型	説明
LONG VARCHAR	任意の長さの文字データです。SQL 文の条件 (WHERE 句の条件など) は、LONG VARCHAR カラムでは実行できません。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-list* へのこれらの指定のみです。「LONG VARCHAR データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。 LONGVARCHAR データに、または LONGVARCHAR データから文字列をキャストできます。
[**UNSIGNED**] **BIGINT**	8 バイトの記憶領域を必要とする整数です。「BIGINT データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
{ **DECIMAL** | **DEC**|**NUMERIC**} **(** *precision* ,*scale*] **)**]	*precision* (合計桁数) と *scale* (小数点以下の桁数) の 2 つの部分で小数を表します。「DECIMAL データ型」『SQL Anywhere サーバ - SQL リファレンス』、「NUMERIC データ型」『SQL Anywhere サーバ - SQL リファレンス』、「Ultra Light での小数点の位置の考慮事項」 68 ページを参照してください。
DOUBLE [**PRECISION**]	倍精度の浮動小数点数です。このデータ型では、PRECISION は DOUBLE データ型名のオプション部分です。「DOUBLE データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
FLOAT [**(** *precision* **)**]	単精度または倍精度の浮動小数点数です。「FLOAT データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
[**UNSIGNED**] {**INT**|**INTEGER**}	4 バイトの記憶領域を必要とする符号なし整数です。「INTEGER データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
REAL	4 バイトで格納される単精度浮動小数点数。「REAL データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
[**UNSIGNED**] **SMALLINT**	2 バイトの記憶領域を必要とする整数です。「SMALLINT データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
[**UNSIGNED**] **TINYINT**	1 バイトの記憶領域を必要とする整数です。「TINYINT データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
DATE	年、月、日などの暦日です。「DATE データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
TIME	時、分、秒、秒以下で構成される時間です。「TIME データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

データ型	説明
DATETIME	TIMESTAMP と同じです。「DATETIME データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
TIMESTAMP	年、月、日、時、分、秒、秒以下で構成される時刻です。「TIMESTAMP データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
VARBINARY (*max-length* **)**	BINARY と同じです。「VARBINARY データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
BINARY (*max-length* **)**	最大長が *max-length* バイトのバイナリ・データです。最大長は 2048 バイト以内である必要があります。「BINARY データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
LONG BINARY	任意の長さのバイナリ・データです。SQL 文の条件 (WHERE 句の条件など) では、LONG BINARY カラムは使用できません。LONG BINARY カラムで実行可能な演算は、挿入、更新、削除、またはクエリの *select-list* への指定のみです。「LONG BINARY データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。 LONGBINARY データに、または LONGBINARY データから値をキャストできます。
UNIQUEIDENTIFIER	通常は、ローを一意に識別する UUID (ユニバーサル・ユニーク識別子) 値を保持するために、プライマリ・キーまたはその他のユニーク・カラムに使用されます。Ultra Light には、あるコンピュータで生成される UUID 値が他のコンピュータで生成される UUID と一致しないように UUID 値を生成する機能が用意されています。したがって、この方法で生成された UNIQUEIDENTIFIER 値は、同期環境でキーとして使用できます。「UNIQUEIDENTIFIER データ型」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

ユーザ定義データ型とそれに相当する型

SQL Anywhere データベースとは異なり、Ultra Light ではユーザ定義データ型はサポートされていません。SQL Anywhere の組み込みエイリアスに相当する Ultra Light データ型を次の表に示します。

SQL Anywhere データ型	Ultra Light で相当する型
MONEY	NUMERIC(19.4)
SMALLMONEY	NUMERIC(10.4)
TEXT	LONG VARCHAR
XML	LONG VARCHAR

データ型の明示的な変換

Ultra Light では、CAST または CONVERT 関数を使用することで、データ型の変換を明示的に要求できます。

注意
ほとんどの場合、自己キャストは処理にまったく影響しません。ただし、CHAR/VARCHAR、BINARY/VARBINARY、および NUMERIC への自己キャストでは、何らかの処理が行われます。

CAST または CONVERT は、次の表に示すように、ほとんどのデータ型の組み合わせで使用できます。

ただし、変換できるかどうかは変換に使用される値に左右されます。次の表の「値依存」欄は、特定の変換エラーが発生しないよう、値に新しいデータ型との互換性が必要であることを示します。次に例を示します。

- **varchar "1234"** を **long** にキャストした場合、この変換はサポートされます。一方、**varchar "hello"** を **long** にキャストした場合は、**hello** が数値ではないので、**SQLE_CONVERSION_ERROR** エラーが発生します。
- **long 1234** を **short** にキャストした場合、この変換はサポートされます。一方、**long 1000000** を **short** にキャストした場合、**1000000** は short で保持できる値の範囲を超えているので、**SQLE_OVERFLOW_ERROR** エラーが発生します。

変換元	可	不可	値依存
BINARY または VARBINARY	CHAR または VARCHAR BINARY LONG BINARY BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG	LONG VARCHAR REAL TIME TIMESTAMP DOUBLE DATE	NUMERIC UID[1]

1 UUID との互換性を保つには、BINARY 値の長さが 16 バイトである必要があります。

変換元	可	不可	値依存
LONG BINARY	BINARY LONG BINARY	BIT CHAR または VARCHAR LONG VARCHAR TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG REAL DOUBLE NUMERIC DATE TIME TIMESTAMP UID	なし
BIT	CHAR または VARCHAR BINARY BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT REAL SIGNED BIG DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	なし

変換元	可	不可	値依存
CHAR または VARCHAR	BINARY または VARBINARY CHAR または VARCHAR LONG VARCHAR	LONG BINARY	BIT TINY INT SIGNED SHORT SHORT INT LONG INT SIGNED LONG BIG INT SIGNED BIG DOUBLE NUMERIC REAL DATE TIME TIMESTAMP UID

変換元	可	不可	値依存
LONG VARCHAR	CHAR または VARCHAR LONG VARCHAR	BINARY または VARBINARY LONG BINARY BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG REAL NUMERIC DATE TIME TIMESTAMP DOUBLE UID	
TINY INT	BINARY または VARBINARY CHAR または VARCHAR TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	

変換元	可	不可	値依存
SHORT INT	BINARY または VARBINARY CHAR または VARCHAR SHORT INT LONG INT SIGNED LONG BIG INT SIGNED BIG REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	BIT TINY INT SIGNED SHORT
SIGNED SHORT	BINARY または VARBINARY CHAR または VARCHAR SIGNED SHORT SIGNED LONG SIGNED BIG REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	SHORT INT LONG INT BIG INT BIT TINY INT
LONG INT	BINARY または VARBINARY CHAR または VARCHAR LONG INT BIG INT SIGNED BIG REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	BIT TINY INT SHORT INT SIGNED SHORT SIGNED LONG

変換元	可	不可	値依存
SIGNED LONG	BINARY または VARBINARY CHAR または VARCHAR SIGNED LONG SIGNED BIG REAL DOUBLE NUMERIC DATE TIMESTAMP	LONG VARCHAR LONG BINARY TIME UID	BIT TINY INT SHORT INT SIGNED SHORT LONG INT BIG INT
BIG INT	BINARY または VARBINARY CHAR または VARCHAR BIG INT REAL DOUBLE NUMERIC	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG SIGNED BIG
SIGNED BIG	BINARY または VARBINARY CHAR または VARCHAR SIGNED BIG REAL DOUBLE NUMERIC DATE TIMESTAMP	LONG VARCHAR LONG BINARY TIME UID	BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT

変換元	可	不可	値依存
REAL	CHAR または VARCHAR REAL DOUBLE NUMERIC	LONG VARCHAR BINARY または VARBINARY LONG BINARY DATE TIME TIMESTAMP UID	BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG
DOUBLE	CHAR または VARCHAR DOUBLE NUMERIC	LONG VARCHAR BINARY または VARBINARY LONG BINARY DATE TIME TIMESTAMP UID	BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG REAL
NUMERIC	CHAR または VARCHAR REAL NUMERIC DOUBLE	LONG VARCHAR LONG BINARY DATE TIME TIMESTAMP UID	BINARY または VARBINARY[2] BIT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG TINY INT

2 ソースの NUMERIC 値が BIG INT にキャストできる場合のみ機能します。

変換元	可	不可	値依存
DATE	CHAR または VARCHAR SIGNED LONG SIGNED BIG DATE TIMESTAMP	LONG VARCHAR LONG BINARY BIT TINY INT SHORT INT SIGNED SHORT LONG INT BIG INT REAL DOUBLE NUMERIC TIME BINARY または VARBINARY UID	
TIME	CHAR または VARCHAR TIME TIMESTAMP	LONG VARCHAR LONG BINARY BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG REAL DOUBLE NUMERIC DATE BINARY または VARBINARY UID	

変換元	可	不可	値依存
TIMESTAMP	CHAR または VARCHAR SIGNED LONG SIGNED BIG DATE TIME TIMESTAMP	LONG VARCHAR LONG BINARY BIT TINY INT SHORT INT SIGNED SHORT LONG INT BIG INT REAL DOUBLE NUMERIC BINARY または VARBINARY UID	
UID	CHAR または VARCHAR UID	LONG VARCHAR LONG BINARY BIT TINY INT SHORT INT SIGNED SHORT LONG INT SIGNED LONG BIG INT SIGNED BIG REAL DOUBLE NUMERIC DATE TIME TIMESTAMP	BINARY または VARBINARY[1]

Ultra Light の式

式は、多くの場合、カラム参照の形式でデータを関数や演算子と組み合わせることによって形成されます。

構文

expression:
case-expression
| *constant*
| [*correlation-name*.]*column-name*
| - *expression*
| *expression operator expression*
| (*expression*)
| *function-name* (*expression*, ⋯)
| *if-expression*
| *special value*
| *input-parameter*

パラメータ

case-expression:
CASE *expression*
WHEN *expression*
THEN *expression*,⋯
[**ELSE** *expression*]
END

alternative form of case-expression:
CASE
WHEN *search-condition*
THEN *expression*,⋯
[**ELSE** *expression*]
END

constant:
integer | *number* | *string* | *host-variable*

special-value:
CURRENT { **DATE** | **TIME** | **TIMESTAMP** }
| **NULL**
| **SQLCODE**
| **SQLSTATE**

if-expression:
IF *condition*
THEN *expression*
[**ELSE** *expression*]
ENDIF

input-parameter:
{ **?** | :*name* [: *indicator-name*] }

operator:
{ **+** | **-** | ***** | **/** | **||** | **%** }

参照

- 「式内の定数」 275 ページ
- 「Ultra Light の特別値」 258 ページ
- 「式内のカラム名」 275 ページ
- 「Ultra Light SQL 関数のリファレンス」 295 ページ
- 「式のサブクエリ」 278 ページ
- 「Ultra Light の探索条件」 280 ページ
- 「Ultra Light のデータ型」 262 ページ
- 「CASE 式」 276 ページ
- 「入力パラメータ」 279 ページ

式内の定数

Ultra Light における定数とは、数値または文字列リテラルです。

構文

' *constant* '

使用法

文字列定数は、一重引用符 (')で囲まれています。

文字列内にアポストロフィを表すには、一重引用符を 2 つ続けて使用します (')。

参照

- 「エスケープ・シーケンス」 『SQL Anywhere サーバ - SQL リファレンス』

例

所有を表す句を使用するには、次のような文字列リテラルを入力します。

'John''s database'

式内のカラム名

式内の識別子です。

構文

correlation-name.column-name

備考

カラム名は、オプションの相関名 (通常はテーブルの名前) の後に続きます。

カラム名がキーワードであるか、カラム名に英字、数字、アンダースコア以外の文字が使用されている場合は、二重引用符 ("") で囲んでください。次の例は、有効なカラム名です。

```
Employees.Name
address
```



```
"date hired"
"salary"."date paid"
```

参照

- 「FROM 句」 403 ページ

IF 式

データの特定サブセットを返す探索条件を設定します。

構文

IF *search-condition*
THEN *expression1*
[**ELSE** *expression2*]
ENDIF

備考

この式は次のように返します。

- *search-condition* が TRUE の場合、IF 式は *expression1* を返します。
- *search-condition* が FALSE で、ELSE 句が指定されている場合、IF 式は *expression2* を返します。
- *search-condition* が FALSE で *expression2* がない場合、IF 式は NULL を返します。
- *search-condition* が UNKNOWN の場合、IF 式は NULL を返します。

参照

- 「NULL 値」 『SQL Anywhere サーバ - SQL リファレンス』
- 「探索条件」 『SQL Anywhere サーバ - SQL リファレンス』

CASE 式

SQL の条件式です。

構文 1

CASE *expression1*
WHEN *expression2* **THEN** *expression3*, …
[**ELSE** *expression4*]
END

```
SELECT id,
  ( CASE name
    WHEN 'Tee Shirt' THEN 'Shirt'
    WHEN 'Sweatshirt' THEN 'Shirt'
    WHEN 'Baseball Cap' THEN 'Hat'
    ELSE 'Unknown'
  END ) as Type
FROM Product
```

構文 2

CASE
WHEN *search-condition*
THEN *expression1*, …
[**ELSE** *expression2*]
END

備考

CASE 式は、正規表現が使用できればどこでも使用できます。

構文 1 CASE キーワードに続く式が最初の WHEN キーワードに続く式と等しい場合、対応する THEN キーワードの後の式が返されます。それ以外の場合、ELSE キーワードがあればそれに続く式が返されます。

たとえば、下記のコードでは CASE 式が SELECT 文の 2 番目の句として使用されています。この式によって、name カラムの値が Sweatshirt であるローが Product テーブルから選択されます。

構文 2 最初の WHEN キーワードに続く search-condition が TRUE の場合、対応する THEN キーワードに続く式が返されます。それ以外の場合、ELSE 句があればそれに続く式が返されます。

省略形 CASE 式の NULLIF 関数 NULLIF 関数は、CASE 文を省略形で記述する方法の 1 つです。NULLIF の構文は、次のとおりです。

NULLIF (*expression-1*, *expression-2*)

NULLIF は 2 つの式の値を比較します。1 番目の式と 2 番目の式が等しい場合、NULLIF は NULL を返します。1 番目の式と 2 番目の式が異なる場合、NULLIF は 1 番目の式を返します。

例

次の文では、CASE 式が SELECT 文の 3 番目の句として使用され、探索条件と文字列を関連付けています。この式では、name カラムの値が Tee Shirt の場合は、Type カラムに Sale が、name カラムが Tee Shirt ではなくて quantity カラムが 50 以上の場合、Type カラムとして Big Sale が表示されます。

```
SELECT id, name,
  ( CASE
    WHEN name='Tee Shirt' THEN 'Sale'
    WHEN quantity >= 50  THEN 'Big Sale'
    ELSE 'Regular price'
  END ) as Type
FROM Product
```

集合式

Ultra Light ランタイムでは提供されない集計の計算を実行します。

構文

SUM(*expression* **)**

備考

集合式は、一連のローから単一の値を計算します。

集合式は、1 つの集合関数が使用される式か、1 つ以上のオペランドがある式です。

SELECT 文に GROUP BY 句がない場合、*select-list* の式がすべて集合式であるか、いずれの式も集合式でない必要があります。SELECT 文に GROUP BY 句がある場合、*select-list* の非集合式を GROUP BY リストに記載します。

例

たとえば、次のクエリは、employee テーブル内の従業員の給与合計を計算します。このクエリでは、SUM(salary) が集合式です。

```
SELECT SUM( salary )
FROM employee
```

式のサブクエリ

別の SELECT 文の内部でネストされた SELECT 文です。

構文

サブクエリは、通常のクエリのように構成されます。

備考

Ultra Light のサブクエリの参照は、次の場合にのみ使用できます。

- FROM 句内のテーブル式として。この形式のテーブル式 (**抽出テーブル**) には、SELECT リスト内の値をフェッチする抽出テーブル名とカラム名が必要です。
- EXISTS、ANY、ALL、IN の各探索条件に値を指定するため。

サブクエリは、サブクエリの前 (左) に指定されている名前を参照して作成できます。これは、左側への外部参照とも呼ばれます。サブクエリ内の項目は参照できません。これは、内部参照とも呼ばれます。

参照

- 「Ultra Light SELECT 文」 401 ページ
- 「サブクエリの使用」 『SQL Anywhere サーバ - SQL の使用法』
- 「Ultra Light の探索条件」 280 ページ

例

次のサブクエリは、在庫数の少ない項目のすべての製品 ID をリストするのに使用します。

```
FROM SalesOrderItems
 ( SELECT ID
   FROM Products
   WHERE Quantity < 20 )
```


入力パラメータ

エンド・ユーザが準備文に値を指定できるようにするためのプレースホルダとして機能します。文は、ユーザが指定する値を使用して実行されます。

構文

{ **?** | :*name* [: *indicator-name*] }

備考

式でプレースホルダ文字 ? または名前形式を使用します。入力パラメータは、カラム名または定数を使用できる任意の場所で使用できます。

文に値が渡される具体的なメカニズムは、Ultra Light クライアントの作成に使用する API によって異なります。

名前形式の使用　入力パラメータの名前形式には特別な意味があります。一般に、常に *name* を使用して、実際の値を渡す複数のロケーションを指定します。

Embedded SQL アプリケーションの場合にのみ、*indicator-name* が、NULL インジケータが配置される変数を指定します。名前形式と他のコンポーネントを同時に使用した場合、*indicator-name* は無視されます。

データ型の抽出　入力パラメータのデータ型は、文が次のいずれかのパターンから準備されるときに抽出されます。

- **CAST** (**?** **AS** *type*)

  ここで *type* は CHARACTER(32) などのデータベースの型指定です。

- バイナリ演算子の 1 つのオペランドが入力パラメータである。型が抽出され、オペランドの型になります。

Ultra Light で型を抽出できなかった場合は、エラーが発生します。次に例を示します。

- **-?**：オペランドが単項である。
- **? + ?**：両方共入力パラメータである。

参照

- 「ホスト変数の使用」『Ultra Light - C/C++ プログラミング』
- 「文の準備」『SQL Anywhere サーバ - プログラミング』
- Ultra Light C/C++：「データ操作: 挿入、削除、更新」『Ultra Light - C/C++ プログラミング』
- Ultra Light.NET：「データ操作：INSERT、UPDATE、DELETE」『Ultra Light - .NET プログラミング』
- Ultra Light for AppForge：「データ操作：INSERT、UPDATE、DELETE」『Ultra Light - AppForge プログラミング』
- Ultra Light for M-Business Anywhere：「データ操作：INSERT、UPDATE、DELETE」『Ultra Light - M-Business Anywhere プログラミング』

例

次の Embedded SQL 文には 2 つの入力パラメータがあります。

```
INSERT INTO MyTable VALUES ( :v1, :v2, :v1)
```

v1 の最初のインスタンスが、文内の v2 と v1 の両方のロケーションにその値を渡します。

Ultra Light の探索条件

WHERE 句、HAVING 句、ジョインの ON 句、または IF 式の探索条件を指定します。探索条件は、述部とも言います。

構文

search-condition:
expression compare expression
| *expression* **IS** [**NOT**] { **NULL** | **TRUE** | **FALSE** | **UNKNOWN** }
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **IN (** *expression*, ... **)**
| *expression* [**NOT**] **IN (** *subquery* **)**
| *expression* [**NOT**] { **ANY** | **ALL** } **(** *subquery* **)**
| *expression* [**NOT**] **EXISTS (** *subquery* **)**
| **NOT** *search-condition*
| *search-condition* **AND** *search-condition*
| *search-condition* **OR** *search-condition*
| (*search-condition*)

パラメータ

compare:
= | > | < | >= | <= | <> | != | !< | !>

備考

Ultra Light では、次の箇所で探索条件を指定します。

- WHERE 句
- HAVING 句
- ON 句
- SQL クエリ

探索条件は、SELECT 文の FROM 句で使用してテーブル内のローのサブセットを選択するか、IF や CASE などの式で使用して特定の値を選択できます。Ultra Light では、すべての条件が TRUE、FALSE、または UNKNOWN のいずれかに評価されます。これは、**3 値的論理**といいます。比較される値のいずれかが NULL の場合、比較結果は UNKNOWN になります。探索条件は、条件の結果が TRUE の場合にのみ満たされます

Ultra Light では、次のタイプの探索条件がサポートされています。

- ALL 条件
- ANY 条件

- BETWEEN 条件
- EXISTS 条件
- IN 条件

この後の各項で、これらの条件について説明します。

注意
サブクエリは、多数の探索条件で使用される式の重要なクラスを構成します。

参照

- 「比較演算子」 281 ページ
- 「3 値的論理」 『SQL Anywhere サーバ - SQL リファレンス』
- 「式のサブクエリ」 278 ページ

比較演算子

探索条件で、複数の式を比較できる演算子です。

構文

expression operator expression

パラメータ

演算子	意味
=	等価
[**NOT**] **LIKE**	テキスト比較 (場合よっては正規表現を使用)
>	より大きい
<	より小さい
>=	大きいかまたは等価
<=	小さいかまたは等価
!=	等価ではない
<>	等価ではない
!>	以下
!<	以上

備考

日付の比較　日付を比較するときには、< はより古いことを意味し、> はより新しいことを意味します。

大文字と小文字の区別　Ultra Light では、比較処理は、該当のデータベースの文字の区別に合わせて実行されます。デフォルトでは、Ultra Light データベースは大文字と小文字を区別しないで作成されます。

NOT 演算子　NOT 演算子は式を否定します。

参照

- 「論理演算子」 282 ページ
- 「Ultra Light の探索条件」 280 ページ

例

次の 2 つのクエリはどちらも、単価が $10 以下の T シャツ (Tee Shirt) と野球帽 (BaseBall Cap) をすべて検索します。ただし、否定の論理演算子 (NOT) と否定の比較演算子 (!>) では、位置が異なることに注意してください。

```
SELECT ID, Name, Quantity
FROM Products
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND NOT UnitPrice > 10

SELECT ID, Name, Quantity
FROM Products
WHERE (name = 'Tee Shirt' OR name = 'BaseBall Cap')
AND UnitPrice !> 10
```

論理演算子

次のいずれかを行います。

- 条件の比較 (AND、OR、NOT)
- 式 (IS) の真理値または NULL 値のテスト

構文 1

condition1 logical-operator condition2

構文 2

NOT *condition*

構文 3

expression **IS** [**NOT**] { *truth-value* | **NULL** }

備考

探索条件は、SELECT 文の FROM 句で使用してテーブル内のローのサブセットを選択するか、IF や CASE などの式で使用して特定の値を選択できます。Ultra Light では、すべての条件が TRUE、FALSE、または UNKNOWN のいずれかに評価されます。これは、**3 値的論理**といいます。比較される値のいずれかが NULL の場合、比較結果は UNKNOWN になります。探索条件は、条件の結果が TRUE の場合にのみ満たされます

AND　両方の条件が TRUE の場合、結合した条件は TRUE になります。条件のいずれかが FALSE の場合は FALSE、それ以外の場合は UNKNOWN になります。

condition1 **OR** *condition2*

OR　条件のいずれかが TRUE の場合、結合した条件は TRUE になります。両方の条件が FALSE の場合は FALSE、それ以外の場合は UNKNOWN になります。

NOT　*condition* が FALSE の場合、NOT 条件は TRUE です。*condition* が TRUE の場合は FALSE、*condition* が UNKNOWN の場合は UNKNOWN になります。

IS　*expression* が指定の *truth-value* (TRUE、FALSE、UNKNOWN のいずれか) と評価されれば条件は TRUE になります。それ以外の場合、値は FALSE です。

参照

- 「3 値的論理」 『SQL Anywhere サーバ - SQL リファレンス』
- 「比較演算子」 281 ページ
- 「Ultra Light の探索条件」 280 ページ

例

IS NULL 条件は、カラムに NULL 値が含まれる場合に満たされます。IS NOT NULL 演算子を使用すると、この条件は、カラムに NULL でない値が含まれる場合に満たされます。たとえば WHERE paid_date IS NULL は、IS NULL 条件です。

ALL 条件

ALL 条件は、比較演算子と組み合せて使用して、1 つの値をサブクエリが生成するデータ値と比較します。

構文

expression compare [**NOT**] **ALL** (*subquery*)

パラメータ

compare:
= | > | < | >= | <= | <> | != | !< | !>

備考

Ultra Light は指定された比較演算子を使用して、テスト値を結果セットのデータ値と比較します。すべての比較の結果が TRUE になる場合、ALL テストは TRUE を返します。

参照

- 「ALL テスト」 『SQL Anywhere サーバ - SQL の使用法』
- 「比較演算子」 281 ページ

例

注文番号 2001 のすべての製品が出荷された後に受けた注文の注文 ID と顧客 ID を検索します。


```
SELECT ID, CustomerID
FROM SalesOrders
WHERE OrderDate > ALL (
 SELECT ShipDate
 FROM SalesOrderItems
 WHERE ID=2001)
```

ANY 条件

ANY 条件は、比較演算子と組み合せて使用して、1 つの値をサブクエリが生成するデータ値のカラムと比較します。

構文 1

expression compare [**NOT**] **ANY (** *subquery* **)**

構文 2

expression = **ANY (** *subquery* **)**

パラメータ

compare:
= | > | < | >= | <= | <> | != | !< | !>

備考

Ultra Light は指定された比較演算子を使用して、テスト値をカラムのデータ値と比較します。いずれかの比較の結果が TRUE になる場合、ANY テストは TRUE を返します。

構文 1 *expression* がサブクエリ結果のいずれかの値と等しい場合は TRUE、expression が NULL ではなく、サブクエリから返されたいずれの値とも等しくない場合は FALSE です。*expression* が NULL 値の場合、サブクエリ結果にローがあれば ANY 条件は UNKNOWN です。サブクエリ結果にローがなければ、条件は必ず FALSE になります。

参照

- 「ANY テスト」『SQL Anywhere サーバ - SQL の使用法』
- 「比較演算子」 281 ページ

例

注文番号 2005 の最初の製品が出荷された後に受けた注文の注文 ID と顧客 ID を検索します。

```
SELECT ID, CustomerID
FROM SalesOrders
WHERE OrderDate > ANY (
 SELECT ShipDate
 FROM SalesOrderItems
 WHERE ID=2005)
```

BETWEEN 条件

包括的範囲を指定します。包括的範囲には、その範囲の間の値だけではなく、上限値と下限値も含まれます。

構文

expression [**NOT**] **BETWEEN** *start-expression* **AND** *end-expression*

備考

BETWEEN 条件は TRUE、FALSE、または UNKNOWN として評価できます。NOT キーワードがない場合、*expression* が *start-expression* と *end-expression* の間にあれば、条件は TRUE と評価されます。NOT キーワードを使用すると条件の意味が逆になりますが、UNKNOWN は変わりません。

BETWEEN 条件は、次の 2 つの不等式の組み合わせと等価です。

[**NOT**] (*expression* >= *start-expression*
AND *expression* <= *end-expression*)

例

$10 未満か、$15 を超える製品をすべてリストします。

```
SELECT Name, UnitPrice
FROM Products
WHERE UnitPrice NOT BETWEEN 10 AND 15
```

EXISTS 条件

サブクエリがクエリ結果のローを生成するかどうかを調べます。

構文

[**NOT**] **EXISTS (** *subquery* **)**

備考

EXISTS 条件は、サブクエリ結果にローが少なくとも 1 つあれば TRUE で、ローがなければ FALSE です。EXISTS 条件は、UNKNOWN にはなりません。

EXISTS 条件の論理は、NOT EXISTS というフォームで否定できます。この場合、テストはサブクエリがローを返さない場合に TRUE を、ローを返す場合に FALSE を返します。

例

2001 年 7 月 13 日より後に注文した顧客をリストします。

```
SELECT GivenName, Surname
FROM Customers
WHERE EXISTS (
 SELECT *
 FROM SalesOrders
 WHERE (OrderDate > '2001-07-13') AND
      (Customers.ID = SalesOrders.CustomerID))
```

IN 条件

メイン・クエリの値からサブクエリの別の値を探索することでメンバシップをチェックします。

構文

expression [**NOT**] **IN**
{ **(** *subquery* **)** | **(** *value-expr*, ⋯ **)** }

パラメータ

value-expr は、単一値をとる式です。これには、文字列、数字、日付、または他の任意の SQL データ型などがあります。

備考

NOT キーワードがない場合、IN 条件は次の規則に従って評価されます。

- *expression* が NULL でなく、少なくとも 1 つの値と等しい場合、TRUE です。
- *expression* が NULL で、値リストが空でない場合、または少なくとも 1 つの値が NULL で、*expression* が他の値のいずれとも等しくない場合、UNKNOWN です。
- *expression* が NULL で、*subquery* が値を返さない場合、または *expression* が NULL でなく、いずれの値も NULL でなく、*expression* がいずれの値とも等しくない場合、FALSE です。

IN 条件の論理は、NOT IN という形式で否定できます。

探索条件 *expression* **IN** (*values*) は、探索条件 *expression* = **ANY**(*values*) と同じです。探索条件 *expression* **NOT IN** (*values*) は、探索条件 *expression* <> **ALL**(*values*) と同じです。

例

カナダのオンタリオ州、マニトバ州、ケベック州に在住する顧客の会社名と状態を選択します。

```
SELECT CompanyName , Province
FROM Customers
WHERE State IN( 'ON', 'MB', 'PQ')
```

Ultra Light の演算子

演算子は値の計算に使用します。計算された値は、さらに上位の式でオペランドとして使用できます。

Ultra Light SQL では、次のタイプの演算子がサポートされています。

- 比較演算子は、1 つ (単項) または 2 つ (バイナリ) の比較オペランドを使用して評価し、結果を返します。比較の結果は、通常の 3 つの論理値、true、false、または unknown になります。
- 算術演算子は、浮動小数点、小数、整数の数値を評価し、結果セットを返します。
- 文字列演算子は、2 つの文字列を連結します。たとえば、"my" + "string" は文字列 "mystring" を返します。
- ビット処理演算子は、整数の内部表現内の特定のビットをオンまたはオフにします。
- 論理演算子は、探索条件を評価します。論理的評価の結果は、通常の 3 つの論理値、true、false、または unknown になります。

一般的な演算子の優先度が適用されます。

参照

- 「演算子の優先度」 289 ページ
- 「比較演算子」 281 ページ
- 「算術演算子」 287 ページ
- 「文字列演算子」 288 ページ
- 「ビット処理演算子」 288 ページ
- 「論理演算子」 282 ページ

算術演算子

算術演算子を使用すると、計算を実行できます。

expression + expression　加算。いずれかの式が NULL 値の場合、結果は NULL 値になります。

expression - expression　減算。いずれかの式が NULL 値の場合、結果は NULL 値になります。

- expression　反転。式が NULL 値の場合、結果は NULL 値になります。

expression * expression　乗算。いずれかの式が NULL 値の場合、結果は NULL 値になります。

expression / expression　除算。いずれかの式が NULL の場合、または 2 番目の式が 0 の場合、結果は NULL になります。

expression % expression　剰余による、2 つの整数での除算の余り (整数) の算出。たとえば、21 を 11 で割ると商は 1、余りは 10 なので、21 % 11 = 10 になります。いずれかの式が NULL 値の場合、結果は NULL 値になります。

参照

- 「算術演算」 『SQL Anywhere サーバ - SQL の使用法』

文字列演算子

文字列演算子を使用すると、文字列を結合できます。ただし、LONGVARCHAR データ型と LONGBINARY データ型ではできません。

expression || expression　文字列連結 (2 本の縦線)。いずれかの文字列が NULL 値の場合、連結には空文字列として扱われます。

expression + expression　代替の文字列連結。+ 連結演算子を使用する場合は、暗黙的データ変換を行わないで、必ずオペランドを文字データ型に明示設定してください。

たとえば、次のクエリは整数値 **579** を返します。

```
SELECT 123 + 456
```

これに対し、次のクエリは文字列 **123456** を返します。

```
SELECT '123' + '456'
```

CAST または CONVERT 関数を使用すると、データ型を明示的に変換できます。

ビット処理演算子

ビット処理演算子は、2 つの式間でビット操作を実行します。Ultra Light では、次の演算子を整数データ型に対して使用できます。

演算子	説明
&	ビット処理 AND
|	ビット処理 OR
^	ビット処理排他的 OR
~	ビット処理 NOT

ビット処理演算子 &、|、~ は、論理演算子 AND、OR、NOT で代用することはできません。ビット処理演算子は、値のビット表現を使用して整数値に対して作用します。

例

次の文は指定するビットが設定されているローを選択します。

```
SELECT *
FROM tableA
WHERE (options & 0x0101) <> 0
```

演算子の優先度

式における演算子の優先度は次のとおりです。カッコ内の式が最初に評価され、続いて乗算と除算、最後に加算と減算が評価されます。その後、文字列の連結が行われます。リストの最上部にある演算子から順に評価されます。

ヒント
演算子の優先度に依存しないで、演算の順序を明示的に指定してください。式で複数の演算子を使用する場合は、カッコを使用して演算の順序を明確にしてください。

1. 名前、関数、定数、IF 式、CASE 式
2. ()
3. 単項演算子 (1 つのオペランドを必要とする演算子) : +、-
4. **~**
5. **&**, **|** , **^**
6. *, /, **%**
7. **+**, **-**
8. **||**
9. 比較 : >、<、<>、!=、<=、>=、[NOT] BETWEEN、[NOT] IN、[NOT] LIKE
10. 比較 : IS [NOT] TRUE、FALSE、UNKNOWN
11. NOT
12. AND
13. OR

Ultra Light の変数

Ultra Light アプリケーションでは、@@identity グローバル変数を除く SQL 変数 (グローバル変数を含む) は使用できません。

@@identity

@@identity グローバル変数は、Ultra Light で使用できる唯一の SQL 変数です。Ultra Light では、SQL クエリではなく、API 呼び出しでこの変数を使用します。「@@identity グローバル変数」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

参照

- 「Ultra Light でのグローバル・データベース ID の考慮事項」 74 ページ
- 「Ultra Light global_database_id オプション」 162 ページ

Ultra Light のクエリ・アクセス・プラン

Ultra Light のクエリ・アクセス・プランは、クエリの実行時にテーブルとインデックスがアクセスされる方法を示します。Ultra Light には**クエリ・オプティマイザ**があります。これは、Ultra Light ランタイムの内部コンポーネントであり、クエリの効率的なプランを生成しようとします。オプティマイザは、テンポラリ・テーブルを使用して中間結果が格納されないようにしたり、クエリで 2 つのテーブルがジョインされるときに、テーブルの関連するサブセットだけがアクセスされるようにしたりします。

オプティマイザの上書き

オプティマイザでは、常に最も効率的なアクセス・プランが目標とされますが、特に多数の可能性がある複雑なクエリでは、この目標の達成は保証されません。極端な場合、OPTION (FORCE ORDER) 句をクエリに追加することによって、Ultra Light が選択するテーブル順序を上書して、クエリに出現する順序でテーブルにアクセスすることもできます。*このオプションは一般的には使用しないことをおすすめします。*パフォーマンスが低い場合、通常は、適切なインデックスを作成して実行速度を上げるようにしてください。

パフォーマンスに関するヒント
クエリを使用してデータを更新しない場合は、クエリで FOR READ ONLY 句を指定してみてください。こうすると、パフォーマンスが向上することがあります。

クエリ・アクセス・プランを確認する場合

次の情報が必要な場合に、Interactive SQL でクエリ・アクセス・プランを確認します。

- 結果を返すために使用されるインデックス。インデックス・スキャン・オブジェクトにテーブルの名前と、そのテーブルで使用されているインデックスが含まれます。
- 結果を返すためにテンポラリ・テーブルが使用されるかどうか。テンポラリ・テーブルは、Ultra Light のテンポラリ・ファイルに書き込まれます。「Ultra Light のテンポラリ・ファイル」 12 ページを参照してください。
- テーブルがジョインされる順序。パフォーマンスへの影響を確認できます。
- クエリの実行が遅い理由、またはクエリの実行が遅くならないかどうか。

Ultra Light のクエリ・アクセス・プランの表示

開発時のサポートとして、Interactive SQL を使用して、準備文の実行方法をまとめた Ultra Light のプランを表示できます。プランは、ユーティリティの [結果] ウィンドウ枠にあるタブに表示されます。

Ultra Light では、クエリ・プランはプランをテキスト形式で短くまとめたものです。他のプランのタイプはサポートされていません。ただし、プランは短く、情報が 1 行にまとめてあるので、簡単に比較できます。

次の文を考えてみます。

```
SELECT I.inv_no, I.name, T.quantity, T.prod_no
FROM Invoice I, Transactions T
WHERE I.inv_no = T.inv_no
```

この文により、下記のようなプランが生成されます。

```
join[scan(Invoice,primary),index-scan(Transactions,secondary)]
```

このプランは、Invoice テーブルのすべてのローを読み込んでから (primary という index を使用)、Transaction テーブルの secondary という index を使用して、inv_no カラムが一致するローのみを読み込むことによって、ジョイン操作を実行することを示します。

参照

- 「Interactive SQL ユーティリティ (dbisql)」 『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light のアクセス・プランの解釈」 292 ページ

Ultra Light のアクセス・プランの解釈

Ultra Light のプランは、クエリのアクセス方法をテキスト形式で短くまとめたものなので、テーブルのジョイン操作またはスキャン操作が実装される方法を理解している必要があります。

- **スキャン操作の場合**　1 つのオペランドで表されます。1 つのテーブルのみに適用され、インデックスが使用されます。テーブル名とインデックス名は操作名の後に丸カッコ ((と))で表されます。
- **その他の操作**　1 つまたは複数のオペランドで表されます。このオペランド自体もプランの場合があります。Ultra Light では、これらのオペランドは、角カッコ ([と]) で囲まれた、カンマ区切りのリストです。

操作リスト

次の表は、Ultra Light でサポートされている操作を示します。

操作	説明
count(*)	テーブルのローの数を数えます。
distinct[*plan*]	クエリの DISTINCT 部分を実装し、重複するローを比較、排除します。基本となるプランでローがソートされるときに使用され、重複、連続するローが排除されます。2 つの連続するローが一致する場合、最初のローだけが結果セットに追加されます。

操作	説明
dummy	処理は行われません。次の 2 つの場合にのみ使用されます。 ♦ FROM 句で DUMMY を指定した場合。 ♦ クエリに FROM 句がない場合。
filter[*plan*]	基本となるプランによって指定される各ローに探索条件を実行します。true と評価されたローだけが、結果セットの一部として転送されます。
group-by[*plan*]	GROUP BY の結果を集約したものを作成し、グループ化されたデータの複数の行をソートします。ローは、発生する順序で表示され、連続するローを比較してグループ化されます。
group-single[*plan*]	1 行だけが返されることがわかっている場合にのみ、GROUP BY の結果を集約したものを作成します。
keyset[*plan*]	テンポラリ・テーブル内のローの作成に使用されたローを記録し、Ultra Light で元の行を更新できるようにします。これらのローを更新しない場合は、クエリで FOR READ ONLY 句を使用してこの操作をなくします。
index-scan(*table-name*, *index-name*)	テーブルの一部だけを読み出します。開始ローはインデックスを使用して検索します。
join[*plan*, *plan*]	2 つのプラン間で内部ジョインを行います。
lojoin[*plan*, *plan*]	2 つのプラン間で左外部ジョインを行います。
like-scan(*table-name*, *index-name*)	テーブルの一部だけを読み出します。開始ローはインデックスを使用してパターン一致で検索します。
rowlimit[*plan*]	伝達されたローに対してローの制限処理を行います。ローの制限は、SELECT 文の TOP n または FIRST 句によって設定されます。
scan(*table-name*, *index-name*)	インデックスが示す順序でテーブル全体を読み出します。
sub-query[*plan*]	サブクエリの開始を示します。
temp[*plan*]	基本となるプラン内のローからテンポラリ・テーブルを作成します。Ultra Light では、基本となるローを順番に並べる必要があり、このとき使用できるインデックスが見つからなかった場合にテンポラリ・テーブルが使用されます。 インデックスを追加して、テンポラリ・テーブルを不要にできます。ただし、インデックスを追加すると、インデックス対象のテーブル内のローの挿入または同期に必要な時間が長くなります。

操作	説明
union-all[*plan*, ..., *plan*]	基本となるプランで生成されるローに対して UNION ALL 操作を行います。

第 13 章

Ultra Light SQL 関数のリファレンス

目次

SQL 関数の概要

関数は、データベースの情報を返すために使用します。式を使用できるところでは必ず関数を使用できます。

関数には、SQL 文と同じ構文の表記規則を使用します。構文の表記規則の全リストについては、「SQL 構文の表記規則」『SQL Anywhere サーバ - SQL リファレンス』を参照してください。

関数のタイプ

この項では、使用可能な関数をタイプ別に分類します。

Ultra Light では、SQL Anywhere 用に記載されている関数のサブセットをサポートしていますが、一部に違いがあります。

注意
特に明記しないかぎり、NULL をパラメータとして受け取る関数は、NULL を返します。

Ultra Light 集合関数

集合関数は、データベースから選択されたロー・グループのデータを要約します。グループは、SELECT 文の GROUP BY 句を使用して形成されます。集合関数は、select リストと、SELECT 文の HAVING 句と ORDER BY 句でのみ使用できます。

関数のリスト

次の集合関数を使用できます。

- 「AVG 関数 [集合]」 306 ページ
- 「COUNT 関数 [集合]」 316 ページ
- 「LIST 関数 [集合]」 337 ページ
- 「MAX 関数 [集合]」 341 ページ
- 「MIN 関数 [集合]」 342 ページ
- 「SUM 関数 [集合]」 368 ページ

Ultra Light データ型変換関数

データ型変換関数を使用して、引数のデータ型を他のデータ型に変換したり、変換が可能かどうかをテストしたりします。

関数のリスト

次のデータ型変換関数を使用できます。

- 「CAST 関数 [データ型変換]」 308 ページ
- 「CONVERT 関数 [データ型変換]」 312 ページ
- 「HEXTOINT 関数 [データ型変換]」 329 ページ
- 「INTTOHEX 関数 [データ型変換]」 333 ページ
- 「ISDATE 関数 [データ型変換]」 334 ページ

Ultra Light 日付と時刻関数

日付と時刻関数は、date データ型と time データ型の操作を行ったり、日付または時刻の情報を返したりします。

この章では、「**日時**」を日付、時刻、またはタイムスタンプを意味する用語として使用しています。特定のデータ型 DATETIME を示す場合は、DATETIME を使用します。

DATETIME データ型の詳細については、「Ultra Light のデータ型」 262 ページを参照してください。

関数のリスト

次の日付と時刻関数を使用できます。

日付の単位

日付関数の多くは、「**日付の単位**」で構成される日付を使用します。次の表は、使用できる日付の単位の値を示します。

日付の単位	省略形	値の範囲
Year	yy	1–9999
Quarter	qq	1–4
Month	mm	1–12
Week	wk	1 ー 54。日曜日を週の最初の日とします。
Day	dd	1–31
Dayofyear	dy	1–366
Weekday	dw	1ー 7 (日曜日 = 1、…、土曜日 = 7)
Hour	hh	0–23
Minute	mi	0–59
Second	ss	0–59
Millisecond	ms	0–999
Calyearofweek	cyr	整数。その週が何年に開始したかを示します。週に年の最初の数日が含まれている場合は、その年の最初の曜日に応じて、週の最初の日が前年になる場合があります。年の最初の曜日が月曜日ー木曜日の場合は、前年に属する日がその年に含まれることはありませんが、年の最初の曜日が金曜日ー日曜日の場合、年の最初の週はその年の最初の月曜日から開始します。
Calweekofyear	cwk	1 ー 54。指定した日付がその年の第何週であるかを示します。
Caldayofweek	cdw	1 ー 7 (月曜日 = 1、…、日曜日 = 7)

Ultra Light その他の関数

その他の関数は、算術式、文字列式、日付／時刻式、他の関数の戻り値に対して操作を実行します。

関数のリスト

次の各種関数を使用できます。

- 「ARGN 関数 [その他]」 303 ページ
- 「COALESCE 関数 [その他]」 312 ページ
- 「EXPLANATION 関数 [その他]」 327 ページ
- 「GREATER 関数 [その他]」 329 ページ
- 「IFNULL 関数 [その他]」 332 ページ

- 「ISNULL 関数 [その他]」 334 ページ
- 「LESSER 関数 [その他]」 337 ページ
- 「NEWID 関数 [その他]」 347 ページ
- 「NULLIF 関数 [その他]」 348 ページ

Ultra Light 数値関数

数値関数は、数値データ型の算術演算を実行したり、数値情報を返したりします。

関数のリスト

次の数値関数を使用できます。

- 「ABS 関数 [数値]」 302 ページ
- 「ACOS 関数 [数値]」 302 ページ
- 「ASIN 関数 [数値]」 304 ページ
- 「ATAN 関数 [数値]」 305 ページ
- 「ATAN2 関数 [数値]」 305 ページ
- 「CEILING 関数 [数値]」 309 ページ
- 「COS 関数 [数値]」 314 ページ
- 「COT 関数 [数値]」 315 ページ
- 「DEGREES 関数 [数値]」 325 ページ
- 「EXP 関数 [数値]」 326 ページ
- 「FLOOR 関数 [数値]」 328 ページ
- 「LOG 関数 [数値]」 339 ページ
- 「LOG10 関数 [数値]」 339 ページ
- 「MOD 関数 [数値]」 344 ページ
- 「PI 関数 [数値]」 350 ページ
- 「POWER 関数 [数値]」 351 ページ
- 「RADIANS 関数 [数値]」 352 ページ
- 「REMAINDER 関数 [数値]」 352 ページ
- 「ROUND 関数 [数値]」 356 ページ
- 「SIGN 関数 [数値]」 360 ページ
- 「SIN 関数 [数値]」 361 ページ
- 「SQRT 関数 [数値]」 363 ページ
- 「TAN 関数 [数値]」 368 ページ
- 「TRUNCNUM 関数 [数値]」 370 ページ

Ultra Light 文字列関数

文字列関数は、文字列に対して変換、抽出、操作の演算を実行したり、文字列に関する情報を返したりします。

マルチバイト文字セットを操作する場合は、使用する関数が文字とバイトのどちらの情報を返すかを十分に確認してください。

関数のリスト

次の文字列関数を使用できます。

- 「ASCII 関数 [文字列]」 303 ページ
- 「BYTE_LENGTH 関数 [文字列]」 307 ページ
- 「BYTE_SUBSTR 関数 [文字列]」 307 ページ
- 「CHAR 関数 [文字列]」 309 ページ
- 「CHARINDEX 関数 [文字列]」 310 ページ
- 「CHAR_LENGTH 関数 [文字列]」 311 ページ
- 「DIFFERENCE 関数 [文字列]」 325 ページ
- 「INSERTSTR 関数 [文字列]」 333 ページ
- 「LCASE 関数 [文字列]」 335 ページ
- 「LEFT 関数 [文字列]」 335 ページ
- 「LENGTH 関数 [文字列]」 336 ページ
- 「LOCATE 関数 [文字列]」 338 ページ
- 「LOWER 関数 [文字列]」 340 ページ
- 「LTRIM 関数 [文字列]」 341 ページ
- 「PATINDEX 関数 [文字列]」 349 ページ
- 「REPEAT 関数 [文字列]」 353 ページ
- 「REPLACE 関数 [文字列]」 353 ページ
- 「REPLICATE 関数 [文字列]」 354 ページ
- 「RIGHT 関数 [文字列]」 355 ページ
- 「RTRIM 関数 [文字列]」 356 ページ
- 「SIMILAR 関数 [文字列]」 360 ページ
- 「SOUNDEX 関数 [文字列]」 361 ページ
- 「SPACE 関数 [文字列]」 362 ページ
- 「STR 関数 [文字列]」 363 ページ
- 「STRING 関数 [文字列]」 364 ページ
- 「STRTOUUID 関数 [文字列]」 365 ページ
- 「STUFF 関数 [文字列]」 366 ページ
- 「SUBSTRING 関数 [文字列]」 366 ページ
- 「TRIM 関数 [文字列]」 369 ページ
- 「UCASE 関数 [文字列]」 371 ページ
- 「UPPER 関数 [文字列]」 371 ページ
- 「UUIDTOSTR 関数 [文字列]」 372 ページ

Ultra Light システム関数

システム関数は、システム情報を返します。

関数のリスト

Ultra Light では、次のシステム関数を使用できます。

- 「DB_PROPERTY 関数 [システム]」 324 ページ

アルファベット順の関数リスト

関数を 1 つずつリストし、その右側に関数のタイプ (数値、文字など) を示します。

特定のタイプのすべての関数へのリンクについては、「関数のタイプ」 297 ページを参照してください。

ABS 関数 [数値]

数値式の絶対値を返します。

構文

ABS(*numeric-expression* **)**

パラメータ

numeric-expression　絶対値が返される数値。

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能。

例

次の文は、値 66 を返します。

```
SELECT ABS( -66 );
```

ACOS 関数 [数値]

数値式のアークコサインをラジアンで返します。

構文

ACOS(*numeric-expression* **)**

パラメータ

numeric-expression　角度のコサイン。

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「ASIN 関数 [数値]」 304 ページ
- 「ATAN 関数 [数値]」 305 ページ
- 「ATAN2 関数 [数値]」 305 ページ
- 「COS 関数 [数値]」 314 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、0.52 のアークコサイン値を返します。

```
SELECT ACOS( 0.52 );
```

ARGN 関数 [その他]

引数リストから選択された引数を返します。

構文

ARGN(*integer-expression*, *expression* [, ⋯] **)**

パラメータ

integer-expression　式リスト内での引数の位置。

expression　関数に渡される任意のデータ型の式。すべて同じデータ型の式を指定してください。

備考

integer-expression の値に n を使用すると、残りの引数リストから n 番目の引数 (1 から開始) が返されます。式のデータ型は任意ですが、すべて同じデータ型にしてください。integer-expression は、1 からリスト内の式の数までの範囲内で指定してください。範囲外の値を指定すると、NULL が返されます。複数の式は、カンマで区切って指定します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 6 を返します。

```
SELECT ARGN( 6, 1,2,3,4,5,6 );
```

ASCII 関数 [文字列]

文字列式の最初のバイトの ASCII 値を整数で返します。

構文

ASCII(*string-expression* **)**

パラメータ

string-expression　文字列。

備考

文字列が空の場合は、0 を返します。リテラル文字列は、引用符で囲んで指定します。データベース文字セットがマルチバイトであり、パラメータ文字列の最初の文字が複数バイトから構成される場合、結果は NULL です。

参照

- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 90 を返します。

```
SELECT ASCII( 'Z' );
```

ASIN 関数 [数値]

数値式のアークサインをラジアンで返します。

構文

ASIN(*numeric-expression* **)**

パラメータ

numeric-expression 角度のサイン。

備考

SIN 関数と ASIN 関数は逆変換の演算です。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「ACOS 関数 [数値]」 302 ページ
- 「ATAN 関数 [数値]」 305 ページ
- 「ATAN2 関数 [数値]」 305 ページ
- 「SIN 関数 [数値]」 361 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、0.52 のアークサイン値を返します。

```
SELECT ASIN( 0.52 );
```


ATAN 関数 [数値]

数値式のアークタンジェントをラジアンで返します。

構文

ATAN(*numeric-expression* **)**

備考

ATAN 関数と TAN 関数は逆変換の演算です。

パラメータ

numeric-expression　角度のタンジェント。

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「ACOS 関数 [数値]」 302 ページ
- 「ASIN 関数 [数値]」 304 ページ
- 「ATAN2 関数 [数値]」 305 ページ
- 「TAN 関数 [数値]」 368 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、0.52 のアークタンジェント値を返します。

```
SELECT ATAN( 0.52 );
```

ATAN2 関数 [数値]

2 つの数の比率のアークタンジェントをラジアンで返します。

構文

ATAN2 (*numeric-expression-1*, *numeric-expression-2* **)**

パラメータ

numeric-expression-1　アークタンジェントが計算される比率の分子。

numeric-expression-2　アークタンジェントが計算される比率の分母。

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「ACOS 関数 [数値]」 302 ページ
- 「ASIN 関数 [数値]」 304 ページ
- 「ATAN 関数 [数値]」 305 ページ
- 「TAN 関数 [数値]」 368 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、比率 0.52 ー 0.60 のアークタンジェント値を返します。

```
SELECT ATAN2( 0.52, 0.60 );
```

AVG 関数 [集合]

対象となるロー・セットの、数値式または設定されたユニークな値の平均値を計算します。

構文 1

AVG(*numeric-expression* | **DISTINCT** *numeric-expression* **)**

パラメータ

numeric-expression　ロー・セットで平均値を計算する対象の式。

DISTINCT 数値式　入力中で一意の数値の平均を計算します。

備考

この平均値には、*numeric-expression* が NULL 値のローは含まれません。グループにローが含まれていない場合は、NULL 値を返します。

参照

- 「SUM 関数 [集合]」 368 ページ
- 「COUNT 関数 [集合]」 316 ページ

標準と互換性

- **SQL/2003**　コア機能。構文 2 は機能 T611 です。

例

次の文は、値 49988.623200 を返します。

```
SELECT AVG( Salary ) FROM Employees ;
```

次の文は、製品リストの一意の価格に基づいて平均値を計算するときに使用できます。

```
SELECT AVG( DISTINCT ListPrice ) FROM Production ;
```

BYTE_LENGTH 関数 [文字列]

文字列のバイト数を返します。

構文

BYTE_LENGTH(*string-expression* **)**

パラメータ

string-expression　長さが計算される文字列。

備考

string-expression の後続空白スペースは、返される長さに含まれます。

NULL 文字列の戻り値は NULL です。

マルチバイト文字セットの文字列の場合、BYTE_LENGTH の値は、CHAR_LENGTH で返される文字数と異なることがあります。

参照

- 「CHAR_LENGTH 関数 [文字列]」 311 ページ
- 「DATALENGTH 関数 [システム]」 316 ページ
- 「LENGTH 関数 [文字列]」 336 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 12 を返します。

```
SELECT BYTE_LENGTH( 'Test Message' );
```

BYTE_SUBSTR 関数 [文字列]

文字列の部分文字列を返します。部分文字列は、文字ではなくバイトを使用して計算されます。

構文

BYTE_SUBSTR(*string-expression*, *start* [, *length*] **)**

パラメータ

string-expression　部分文字列が取得される文字列。

start　部分文字列の開始位置を示す整数式。正の整数の場合、文字列の先頭から開始します。先頭文字の位置は 1 です。負の整数の場合、文字列の末尾から開始します。最後の文字の位置は -1 です。

length　部分文字列の長さを示す整数式。正の *length* は、取得するバイト数が開始位置から*開始*することを指定します。負の *length* は、開始位置から最大 *length* バイト左側のバイトを返します。

備考

length を指定すると、部分文字列は指定したバイト数に制限されます。*start* と *length* には、正または負の値を指定できます。負の数と正の数を適切に組み合わせて使用すると、文字列の先頭または末尾のどちらからでも部分文字列を取得できます。

start が 0 で length が負でない場合は、1 の *start* 値が使用されます。*start* が 0 で *length* が負の場合は、-1 の start 値が使用されます。

参照

- 「SUBSTRING 関数 [文字列]」 366 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 Test を返します。

```
SELECT BYTE_SUBSTR( 'Test Message', 1, 4 );
```

CAST 関数 [データ型変換]

指定されたデータ型に変換した式の値を返します。

構文

CAST(*expression* **AS** *datatype* **)**

パラメータ

expression　変換される式。

data type　ターゲットのデータ型。

備考

文字列型の長さを指定しない場合は、データベース・サーバによって適切な長さが選択されます。DECIMAL 変換で精度も位取りも指定しない場合は、データベース・サーバによって適切な値が選択されます。

参照

- 「CONVERT 関数 [データ型変換]」 312 ページ
- 「LEFT 関数 [文字列]」 335 ページ

標準と互換性

- **SQL/2003**　コア機能。

例

次の関数は、文字列を日付として使用することを保証します。

```
SELECT CAST( '2000-10-31' AS DATE );
```

式 1 + 2 の値を計算し、その結果を 1 文字の文字列にキャストします。

```
SELECT CAST( 1 + 2 AS CHAR );
```

次のように CAST 関数を使用して、文字列を短縮できます。

```
SELECT CAST ( 'Surname' AS CHAR(5) );
```

CEILING 関数 [数値]

数値の切り上げ値を返します。

構文

CEILING(*numeric-expression* **)**

パラメータ

numeric-expression　切り上げ値を計算する対象の数値。

備考

Ceiling 関数は指定した値以上の最初の整数を返します。正の数値の場合、「丸め」とも呼ばれます。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「FLOOR 関数 [数値]」 328 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 60 を返します。

```
SELECT CEILING( 59.84567 );
```

CHAR 関数 [文字列]

数値の ASCII 値を持つ文字を返します。

構文

CHAR(*integer-expression* **)**

パラメータ

integer-expression　ASCII 文字に変換される数値。0 ー 255 の範囲内の数を指定します。

備考

返される文字は、バイナリ・ソート順に従って、現在のデータベース側文字セットの指定した数値式に対応しています。

整数式の値が 255 より大きいか、0 より小さい場合、CHAR は NULL を返します。

参照

- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 Y を返します。

```
SELECT CHAR( 89 );
```

CHARINDEX 関数 [文字列]

ある文字列内でのもう 1 つの文字列の位置を返します。

構文

CHARINDEX(*string-expression-1*, *string-expression-2* **)**

パラメータ

string-expression-1　検索する文字列。

string-expression-2　検索される文字列。

備考

string-expression-1 の先頭文字の位置を 1 とします。検索される文字列内に、検索する文字列のインスタンスが複数存在する場合、CHARINDEX 関数は最初のインスタンスの位置を返します。

検索される文字列内に、検索する文字列が存在しない場合、CHARINDEX 関数は 0 を返します。

参照

- 「SUBSTRING 関数 [文字列]」 366 ページ
- 「REPLACE 関数 [文字列]」 353 ページ
- 「LOCATE 関数 [文字列]」 338 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、名に K という文字が含まれる場合にのみ、Surname テーブルと GivenName テーブルから姓名を返します。

```
SELECT Surname, GivenName
FROM Employees
WHERE CHARINDEX( 'K', Surname ) = 1 ;
```

返された結果：

Surname	GivenName
Klobucher	James
Kuo	Felicia
Kelly	Moira

CHAR_LENGTH 関数 [文字列]

文字列の文字数を返します。

構文

CHAR_LENGTH (*string-expression* **)**

パラメータ

string-expression　長さが計算される文字列。

備考

後続空白スペースは、返される長さに含まれます。

NULL 文字列の戻り値は NULL です。

マルチバイト文字セットの文字列の場合、CHAR_LENGTH 関数で返される値は、BYTE_LENGTH 関数で返されるバイト数と異なることがあります。

注意
データ型が CHAR、VARCHAR、LONG VARCHAR の場合、CHAR_LENGTH 関数と LENGTH 関数を使用すると同じ結果が得られます。ただし、BINARY データ型とビット配列データ型には LENGTH 関数を使用します。

参照

- 「BYTE_LENGTH 関数 [文字列]」 307 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　コア機能。

例

次の文は、値 8 を返します。

```
SELECT CHAR_LENGTH( 'Chemical' );
```

COALESCE 関数 [その他]

リストの中から NULL でない最初の式を返します。この関数は ISNULL 関数と同じです。

構文

COALESCE(*expression*, *expression* [, ⋯] **)**

パラメータ

expression　任意の式。

2 つ以上の式を関数に渡します。すべての式は比較可能となっている必要があります。

備考

結果が NULL になるのはすべての引数が NULL の場合のみです。

このパラメータにはスカラ型を指定できますが、同じ型を指定する必要はありません。

この関数がデータベース・サーバで処理される方法の詳細については、「ISNULL 関数 [その他]」 334 ページを参照してください。

参照

- 「ISNULL 関数 [その他]」 334 ページ

標準と互換性

- **SQL/2003**　コア機能。

例

次の文は、値 34 を返します。

```
SELECT COALESCE( NULL, 34, 13, 0 );
```

CONVERT 関数 [データ型変換]

指定されたデータ型に変換した式を返します。

構文

CONVERT(*datatype*, *expression* [, *format-style*] **)**

パラメータ

datatype　変換後の式のデータ型。

expression　変換される式。

format-style　出力値に適用されるスタイル・コード。このパラメータを使用するのは、文字列を日付または時刻のデータ型に変換するとき、またはその反対方向に変換するときです。次の表は、サポートされるスタイル・コードと、そのスタイル・コードで作成される出力形式の表現です。スタイル・コードは世紀を出力形式に含めるかどうかによって 2 つのカラムに分けられます (たとえば、06 と 2006 など)。

100 以上の位なし (yy) のスタイル・コード	100 以上の位あり (yyyy) のスタイル・コード	出力形式
-	0 または 100	Mmm dd yyyy hh:nnAA
1	101	mm/dd/yy[yy]
2	102	[yy]yy.mm.dd
3	103	dd/mm/yy[yy]
4	104	dd.mm.yy[yy]
5	105	dd-mm-yy[yy]
6	106	dd Mmm yy[yy]
7	107	Mmm dd, yy[yy]
8	108	hh:nn:ss
-	9 または 109	Mmm dd yyyy hh:nn:ss:sssAA
10	110	mm-dd-yy[yy]
11	111	[yy]yy/mm/dd
12	112	[yy]yymmdd
-	13 または 113	dd Mmm yyyy hh:nn:ss:sss (24 時間表記、ヨーロッパのデフォルト、ミリ秒、4 桁の年)
-	14 または 114	hh:nn:ss:sss (24 時間表記)
-	20 または 120	yyyy-mm-dd hh:nn:ss (24 時間表記、ODBC 標準、4 桁の年)
-	21 または 121	yyyy-mm-dd hh:nn:ss.sss (24 時間表記、ODBC 標準、ミリ秒、4 桁の年)

備考

format-style 引数を指定しない場合は、スタイル・コード 0 が使用されます。

各出力記号 (Mmm など) が出力するスタイルの詳細については、「Ultra Light date_format プロパティ」 139 ページを参照してください。

参照

- 「CAST 関数 [データ型変換]」 308 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、フォーマット・スタイルの使用方法を示します。

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 104 ) FROM SalesOrders ;
```

OrderDate
16.03.2000
20.03.2000
23.03.2000
25.03.2000
…

```
SELECT CONVERT( CHAR( 20 ), OrderDate, 7 ) FROM SalesOrders;
```

OrderDate
Mar 16, 00
Mar 20, 00
Mar 23, 00
Mar 25, 00
…

次の文は、整数への変換を示し、値 5 を返します。

```
SELECT CONVERT( integer, 5.2 );
```

COS 関数 [数値]

数値をラジアンからコサインに変換します。

構文

COS(*numeric-expression* **)**

パラメータ

numeric-expression　角度 (ラジアン)。

備考

COS 関数は *numeric-expression* で指定される角度のコサインを返します。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメータが NULL 値の場合、結果は NULL 値になります。

参照

- 「ACOS 関数 [数値]」 302 ページ
- 「COT 関数 [数値]」 315 ページ
- 「SIN 関数 [数値]」 361 ページ
- 「TAN 関数 [数値]」 368 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、角度 0.52 ラジアンのコサイン値を返します。

```
SELECT COS( 0.52 );
```

COT 関数 [数値]

数値をラジアンからコタンジェントに変換します。

構文

COT(*numeric-expression* **)**

パラメータ

numeric-expression 角度 (ラジアン)。

備考

COT 関数は *numeric-expression* で指定される角度のコタンジェントを返します。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメータが NULL 値の場合、結果は NULL 値になります。

参照

- 「COS 関数 [数値]」 314 ページ
- 「SIN 関数 [数値]」 361 ページ
- 「TAN 関数 [数値]」 368 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、0.52 のコタンジェント値を返します。

```
SELECT COT( 0.52 );
```

COUNT 関数 [集合]

指定されたパラメータに従って、グループのロー数をカウントします。

構文 1

COUNT(
*
| *expression*
| **DISTINCT** *expression*
)

パラメータ

***** 各グループの中のロー数を返します。

expression ローの数を返す式。

DISTINCT 式 排他ローの数を返す式。

備考

値が NULL 値のローは、カウントに含まれません。

参照

- 「AVG 関数 [集合]」 306 ページ
- 「SUM 関数 [集合]」 368 ページ

標準と互換性

- **SQL/2003** コア機能。構文 2 は機能 T611 です。

例

次の文は、ユニークな各都市と、その都市の値を持つロー数を返します。

```
SELECT City , COUNT(*) FROM Employees GROUP BY City;
```

DATALENGTH 関数 [システム]

式の結果に必要な基本となる記憶領域の長さ (バイト) を返します。

構文

DATALENGTH(*expression* **)**

パラメータ

expression 通常、*expression* はカラム名です。*expression* が文字列定数の場合は、引用符で囲みます。

備考

DATALENGTH 関数の戻り値は次のとおりです。

データ型	DATALENGTH
SMALLINT	2
INTEGER	4
DOUBLE	8
CHAR	データの長さ
BINARY	データの長さ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、CompanyName カラムの最も長い文字列である値 27 を返します。

```
SELECT MAX( DATALENGTH( CompanyName ) )
FROM Customers;
```

次の文は、文字列 '8sdofinsv8s7a7s7gehe4h' の長さとして、値 22 を返します。

```
SELECT DATALENGTH( '8sdofinsv8s7a7s7gehe4h' );
```

DATE 関数 [日付と時刻]

式を日付に変換し、時間、分、秒を削除します。

日付フォーマットの解釈の制御については、「Ultra Light date_order プロパティ」 142 ページを参照してください。

構文

DATE(*expression* **)**

パラメータ

expression 日付フォーマットに変換される値。通常、文字列。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 1999-01-02 を日付として返します。

```
SELECT DATE( '1999-01-02 21:20:53' );
```


次の文は、SYSOBJECT システム・ビューにリストされるすべてのオブジェクトについて、作成日を返します。

```
SELECT DATE( creation_time ) FROM SYSOBJECT;
```

DATEADD 関数 [日付と時刻]

日付にいくつかの日付の単位を加算した日付を返します。

構文

DATEADD(*date-part*, *numeric-expression*, *date-expression* **)**

date-part :
year | **quarter** | **month** | **week** | **day** | **dayofyear** | **hour** | **minute** | **second** | **millisecond**

パラメータ

date-part 日付に加算される日付の単位。日付の単位の詳細については、「日付の単位」 298 ページを参照してください。

numeric-expression 日付に加算される日付の単位の数。*numeric_expression* には任意の数値型を指定できますが、値は整数にトランケートされます。

date-expression 変更される日付。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 1995-11-02 00:00:000.000 を返します。

```
SELECT DATEADD( month, 102, '1987/05/02' );
```

DATEDIFF 関数 [日付と時刻]

2 つの日付間の期間を返します。

構文

DATEDIFF(*date-part*, *date-expression-1*, *date-expression-2* **)**

date-part :
year | **quarter** | **month** | **week** | **day** | **dayofyear** | **hour** | **minute** | **second** | **millisecond**

パラメータ

date-part 期間を測定する日付の単位を指定します。上記の日付オブジェクトから 1 つを選択します。日付の単位の完全なリストについては、「日付の単位」 298 ページを参照してください。

date-expression-1 期間の開始日。この値を *date-expression-2* から引いて、2 つの引数の間に存在する *date-part* の数を返します。

date-expression-2　期間の終了日。この値から *date-expression-1* を引いて、2 つの引数の間に存在する *date-part* の数を返します。

備考

この関数は、指定した 2 つの日付間に存在する日付の単位の数を計算します。結果は、日付の単位の数を表す、(date2 - date1) と同値の符号付き整数です。

DATEDIFF 関数の結果が日付の単位の整数倍でない場合、結果は丸められずに切り捨てになります。

day を日付の単位として使用する場合、DATEDIFF 関数は指定された 2 つの時刻の間の午前 0 時の回数を返します。このとき、2 番目の日付は計算に含まれますが、最初の日付は含まれません。

month を日付の単位として使用する場合、DATEDIFF 関数は 2 つの日付の間に存在する月の初日の数を返します。このとき、2 番目の日付は計算に含まれますが、最初の日付は含まれません。

week を日付の単位として使用する場合、DATEDIFF 関数は 2 つの日付の間に存在する日曜日の数を返します。このとき、2 番目の日付は計算に含まれますが、最初の日付は含まれません。

より短い時間単位のために、オーバフロー値が用意されています。

- **ミリ秒**　24 日
- **秒**　68 年
- **分**　4083 年
- **その他**　オーバフロー制限なし

これらの制限値を超えると、この関数はオーバフロー・エラーを返します。

標準と互換性

- **SQL/2003**　Transact-SQL 拡張。

例

次の文は、1 を返します。

```
SELECT DATEDIFF( hour, '4:00AM', '5:50AM' );
```

次の文は、102 を返します。

```
SELECT DATEDIFF( month, '1987/05/02', '1995/11/15' );
```

次の文は、0 を返します。

```
SELECT DATEDIFF( day, '00:00', '23:59' );
```

次の文は、4 を返します。

```
SELECT DATEDIFF( day,
  '1999/07/19 00:00',
  '1999/07/23 23:59' );
```

次の文は、0 を返します。

```
SELECT DATEDIFF( month, '1999/07/19', '1999/07/23' );
```

次の文は、1 を返します。

```
SELECT DATEDIFF( month, '1999/07/19', '1999/08/23' );
```

DATEFORMAT 関数 [日付と時刻]

日付式を表す文字列を、指定したフォーマットで返します。

構文

DATEFORMAT(*datetime-expression*, *string-expression* **)**

パラメータ

datetime-expression　変換される日時。

string-expression　変換後の日付のフォーマット。

日付フォーマットの説明については、「Ultra Light date_format プロパティ」 139 ページを参照してください。

備考

string-expression には、許容される任意の日付フォーマットを使用できます。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 Jan 01, 1989 を返します。

```
SELECT DATEFORMAT( '1989-01-01', 'Mmm dd, yyyy' );
```

DATENAME 関数 [日付と時刻]

日時の値の特定部分の名前 (月の名前 "June" など) を文字列で返します。

構文

DATENAME(*date-part*, *date-expression* **)**

パラメータ

date-part　名前が返される日付の単位。使用可能な日付の単位の完全なリストについては、「日付の単位」 298 ページを参照してください。

date-expression　日付の単位名が返される日付。要求する *date-part* が含まれた日付を指定してください。

備考

結果が数値 (23 日など) の場合でも、DATENAME 関数は文字列を返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 May を返します。

```
SELECT DATENAME( month, '1987/05/02' );
```

DATEPART 関数 [日付と時刻]

日時の値の一部の値を返します。

構文

DATEPART(*date-part*, *date-expression* **)**

パラメータ

date-part　名前が返される日付の単位。使用可能な日付の単位の完全なリストについては、「日付の単位」 298 ページを参照してください。

date-expression　値が返される日付。

備考

date-part フィールドが含まれた日付を指定してください。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 5 を返します。

```
SELECT DATEPART( month , '1987/05/02' );
```

DATETIME 関数 [日付と時刻]

式をタイムスタンプに変換します。

構文

DATETIME(*expression* **)**

パラメータ

expression　変換される式。通常は文字列です。

備考

数値を変換しようとするとエラーが返ります。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 1998-09-09 12:12:12.000 のタイムスタンプを返します。

```
SELECT DATETIME( '1998-09-09 12:12:12.000' );
```

DAY 関数 [日付と時刻]

1 ー 31 の整数を返します。

構文

DAY(*date-expression* **)**

パラメータ

date-expression 日付。

備考

日付の月日に対応する整数 1 ー 31。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 12 を返します。

```
SELECT DAY( '2001-09-12' );
```

DAYNAME 関数 [日付と時刻]

日付から曜日の名前を返します。

構文

DAYNAME(*date-expression* **)**

パラメータ

date-expression 日付。

備考

返される曜日名は英語で、Sunday、Monday、Tuesday、Wednesday、Thursday、Friday、Saturday です。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 Saturday を返します。

```
SELECT DAYNAME ( '1987/05/02' );
```

DAYS 関数 [日付と時刻]

日付を評価する関数。詳細については、この関数の使用法を参照してください。

構文 1：整数

DAYS([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

DAYS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する日数。*integer-expression* が負の場合、タイムスタンプから適切な日数が引かれます。整数式を指定する場合は、*datetime-expression* を日付またはタイムスタンプとして明示的にキャストしてください。

データ型のキャストの詳細については、「CAST 関数 [データ型変換]」 308 ページを参照してください。

備考

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02-29 からの日数を返します。

  注意
  0000-02-29 は実際の日付を指すための値ではありません。日付アルゴリズムで使用される日付です。

- 2 つの日付を指定すると、2 つの日付の間の日数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- この関数で 1 つの日付と整数を指定すると、指定した日付に、指定した整数の日数が加算されます。代わりに、DATEADD 関数を使用します。

この関数では、時間、分、秒が無視されます。

参照

- 「DATEDIFF 関数 [日付と時刻]」 318 ページ
- 「DATEADD 関数 [日付と時刻]」 318 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、整数 729889 を返します。

```
SELECT DAYS( '1998-07-13 06:07:12' );
```

次の文は、整数値 −366 を返します。これは、2 番目の日付が最初の日付より 366 日前の日付であることを示します。2 つ目の例 (DATEDIFF) の使用をおすすめします。

```
SELECT DAYS( '1998-07-13 06:07:12',
        '1997-07-12 10:07:12' );

SELECT DATEDIFF( day,
  '1998-07-13 06:07:12',
  '1997-07-12 10:07:12' );
```

次の文は、タイムスタンプ 1999-07-14 00:00:00.000 を返します。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT DAYS( CAST('1998-07-13' AS DATE ), 366 );

SELECT DATEADD( day, 366, '1998-07-13' );
```

DB_PROPERTY 関数 [システム]

指定したプロパティの値を返します。

構文

DB_PROPERTY(*property-name***)**

パラメータ

property-name　データベース・プロパティ名。

備考

文字列を返します。

Ultra Light でオプションを設定するには、SET OPTION 文、またはコンポーネントの API に固有のデータベース・オプション設定メソッドを使用します。

参照

- 「Ultra Light SET OPTION 文」 406 ページ
- Ultra Light C++：「SetDatabaseOption 関数」 『Ultra Light - C/C++ プログラミング』
- Ultra Light.NET：「SetDatabaseOption メソッド」 『Ultra Light - .NET プログラミング』
- Ultra Light for AppForge：「SetDatabaseOption メソッド」 『Ultra Light - AppForge プログラミング』

例

次の文は、現在のデータベースのページ・サイズをバイト単位で返します。

```
SELECT DB_PROPERTY( 'page_size');
```

DEGREES 関数 [数値]

数値をラジアンから度数に変換します。

構文

DEGREES(*numeric-expression* **)**

パラメータ

numeric-expression　角度 (ラジアン)。

備考

DEGREES 関数は *numeric-expression* で指定される角度を返します。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメータが NULL 値の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 29.79380534680281 を返します。

```
SELECT DEGREES( 0.52 );
```

DIFFERENCE 関数 [文字列]

2 つの文字列式の SOUNDEX 値の差を返します。

構文

DIFFERENCE (*string-expression-1*, *string-expression-2* **)**

パラメータ

string-expression-1　最初の SOUNDEX 引数。

string-expression-2　2 番目の SOUNDEX 引数。

備考

DIFFERENCE 関数は 2 つの文字列の SOUNDEX 値を比較し、値の類似性を評価し、0 ー 4 の値を返します。最も一致する場合は 4 です。

この関数は常に何らかの値を返します。結果が NULL になるのは引数の 1 つが NULL の場合のみです。

参照

- 「SOUNDEX 関数 [文字列]」 361 ページ

- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 3 を返します。

```
SELECT DIFFERENCE( 'test', 'chest' );
```

DOW 関数 [日付と時刻]

指定した日付の曜日を表す 1 ー 7 の数を返します (日曜日 = 1、月曜日 = 2、以下同様)。

構文

DOW(*date-expression* **)**

パラメータ

date-expression 評価する日付。

備考

DOW 関数は、first_day_of_week データベース・オプションで指定された値の影響を受けません。たとえば、first_day_of_week が月曜日に設定されている場合でも、DOW 関数は月曜日に 2 を返します。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 5 を返します。

```
SELECT DOW( '1998-07-09' );
```

次の文は、Employees テーブルに問い合わせ、週の曜日を表す数として StartDate を返します。

```
SELECT DOW( StartDate ) FROM Employees;
```

EXP 関数 [数値]

指数関数 (e を底とする数のべき乗) を返します。

構文

EXP(*numeric-expression* **)**

パラメータ

numeric-expression 指数。

備考

EXP 関数は、*numeric-expression* で指定された値の指数値を返します。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメータが NULL 値の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 3269017.3724721107 を返します。

```
SELECT EXP( 15 );
```

EXPLANATION 関数 [その他]

SQL 文のプランの最適化方法を返します。

構文

EXPLANATION(string-expression **)**

パラメータ

string-expression　SQL 文。通常は SELECT 文ですが、UPDATE 文または DELETE 文も指定できます。

備考

最適化結果は文字列として返されます。

この情報は、追加するインデックスの決定や、パフォーマンスを向上するためのデータベース構造の決定に役立ちます。

Interactive SQL では、SQL 文のプランを [結果] ウィンドウ枠の [プラン] タブに表示できます。

参照

- 「Ultra Light のクエリ・アクセス・プラン」 291 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、SELECT 文を文字列パラメータとして渡し、クエリを実行するためのプランを返します。

```
SELECT EXPLANATION( 'SELECT * FROM Departments WHERE DepartmentID > 100' );
```


FLOOR 関数 [数値]

数値の切り捨て値 (値以下の最大整数値) を返します。

構文

FLOOR(*numeric-expression* **)**

パラメータ

numeric-expression　値を指定します。通常は FLOAT です。

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「CEILING 関数 [数値]」 309 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、123 の Floor 値を返します。

```
SELECT FLOOR (123);
```

次の文は、123 の Floor 値を返します。

```
SELECT FLOOR (123.45);
```

次の文は、-124 の Floor 値を返します。

```
SELECT FLOOR (-123.45);
```

GETDATE 関数 [日付と時刻]

現在の年、月、日、時、分、秒、秒以下を返します。

構文

GETDATE( )

備考

精度はシステム・クロックの精度によって制限されます。

GETDATE 関数が返す情報は、NOW 関数と CURRENT TIMESTAMP 特別値が返す情報と同じです。

参照

- 「NOW 関数 [日付と時刻]」 348 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、システムの日付と時刻を返します。

```
SELECT GETDATE( );
```

GREATER 関数 [その他]

2 つのパラメータ値のうち、より大きい値を返します。

構文

GREATER(*expression-1*, *expression-2* **)**

パラメータ

expression-1　比較される最初のパラメータ値。

expression-2　比較される 2 番目のパラメータ値。

備考

パラメータが等しい場合は、最初のパラメータを返します。

参照

- 「LESSER 関数 [その他]」 337 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 10 を返します。

```
SELECT GREATER( 10, 5 ) FROM dummy;
```

HEXTOINT 関数 [データ型変換]

16 進文字列と同等の 10 進整数を返します。

構文

HEXTOINT(*hexadecimal-string* **)**

パラメータ

hexadecimal-string　整数に変換される文字列。

備考

HEXTOINT 関数は、数値、大文字または小文字の A ー F のみで構成される文字列リテラルまたは変数を受け入れます (0x プレフィクスが付いている場合、付いていない場合の両方)。次に HEXTOINT の有効な使用例をすべて示します。

```
SELECT HEXTOINT( '0xFFFFFFFF' );
SELECT HEXTOINT( '0x00000100' );
SELECT HEXTOINT( '100' );
SELECT HEXTOINT( '0xffffffff80000001' );
```

HEXTOINT 関数は 0x プレフィクスがあれば削除します。データが 8 桁を超える場合、符号付き 32 ビット整数値として表現できる値を示す必要があります。

HEXTOINT 関数は、プラットフォームに依存しない SQL INTEGER 相当の 16 進文字列を返します。右から 8 桁目が数値 8 ー 9 か大文字または小文字の A ー F で、その前の桁がすべて大文字または小文字の F の場合、16 進値は負の整数値になります。次の HEXTOINT は無効な使用例です。引数が符号付き 32 ビット整数で表現できない正の整数値を示しているためです。

```
SELECT HEXTOINT( '0x0080000001' );
```

参照

- 「INTTOHEX 関数 [データ型変換]」 333 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 420 を返します。

```
SELECT HEXTOINT( '1A4' );
```

HOUR 関数 [日付と時刻]

日時の時間コンポーネントを返します。

構文

HOUR(*datetime-expression* **)**

パラメータ

datetime-expression 日時。

備考

返される値は日時の時間に対応する 0 ー 23 の数値です。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 21 を返します。


```
SELECT HOUR( '1998-07-09 21:12:13' );
```

HOURS 関数 [日付と時刻]

時間を評価する関数。詳細については、この関数の使用法を参照してください。

構文 1：整数

HOURS ([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

HOURS (*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する時間数。*integer-expression* が負の場合、日時から適切な時間数が引かれます。整数式を指定する場合は、*datetime-expression* を DATETIME データ型として明示的にキャストしてください。

データ型のキャストの詳細については、「CAST 関数 [データ型変換]」 308 ページを参照してください。

備考

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02-29 からの時間数を返します。

  注意
  0000-02-29 は実際の日付を指すための値ではありません。日付アルゴリズムで使用される日付です。

- 2 つのタイムスタンプを指定すると、2 つの時刻の間の時間数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- 1 つの日付と整数を指定すると、指定した日付に、指定した整数の時間数が加算されます。代わりに、DATEADD 関数を使用します。

参照

- 「DATEDIFF 関数 [日付と時刻]」 318 ページ
- 「DATEADD 関数 [日付と時刻]」 318 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 4 を返します。これは、2 番目のタイムスタンプが、最初のタイムスタンプの 4 時間後であることを示します。2 つ目の例 (DATEDIFF) の使用をおすすめします。


```
SELECT HOURS( '1999-07-13 06:07:12',
  '1999-07-13 10:07:12' );

SELECT DATEDIFF( hour,
  '1999-07-13 06:07:12',
  '1999-07-13 10:07:12' );
```

次の文は、値 17517342 を返します。

```
SELECT HOURS( '1998-07-13 06:07:12' );
```

次の文は、日時 1999-05-13 02:05:07.000 を返します。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT HOURS(
  CAST( '1999-05-12 21:05:07' AS DATETIME ), 5 );

SELECT DATEADD( hour, 5, '1999-05-12 21:05:07' );
```

IFNULL 関数 [その他]

最初の NULL 以外の式を返します。

構文

IFNULL(*expression-1*, *expression-2* [, *expression-3*] **)**

パラメータ

expression-1　評価される式。この値によって、*expression-2* または *expression-3* のどちらが返るかが決まります。

expression-2　*expression-1* が NULL の場合の戻り値。

expression-3　*expression-1* が NULL でない場合の戻り値。

備考

最初の式が NULL 値の場合は、2 番目の式の値を返します。最初の式が NULL でない場合は、3 番目の式の値を返します。最初の式が NULL ではなく、3 番目の式が存在しない場合は、NULL を返します。

標準と互換性

- **SQL/2003**　Transact-SQL 拡張。

例

次の文は、値−66 を返します。

```
SELECT IFNULL( NULL, -66 );
```

次の文は、最初の式が NULL ではなく、3 番目の式が存在しないため、NULL を返します。

```
SELECT IFNULL( -66, -66 );
```

INSERTSTR 関数 [文字列]

別の文字列の指定された位置に文字列を挿入します。

構文

INSERTSTR(*integer-expression*, *string-expression-1*, *string-expression-2* **)**

パラメータ

integer-expression　文字列の挿入位置。文字列の先頭に挿入する場合は、0 を使用します。

string-expression-1　別の文字列が挿入される文字列。

string-expression-2　挿入する文字列。

備考

参照

- 「STUFF 関数 [文字列]」 366 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 backoffice を返します。

```
SELECT INSERTSTR( 0, 'office ', 'back' );
```

INTTOHEX 関数 [データ型変換]

整数に対応する 16 進値の文字列を返します。

構文

INTTOHEX(*integer-expression* **)**

パラメータ

integer-expression　16 進に変換される整数。

参照

- 「HEXTOINT 関数 [データ型変換]」 329 ページ

標準と互換性

- **SQL/2003**　Transact-SQL 拡張。

例

次の文は、値 0000009c を返します。

```
SELECT INTTOHEX( 156 );
```

ISDATE 関数 [データ型変換]

文字列引数を日付に変換できるかどうかをテストします。

構文

ISDATE(*string* **)**

パラメータ

string　文字列が有効な日付を表すかどうかを判断するために分析される文字列。

備考

変換できる場合は 1 を返し、できない場合は 0 を返します。引数が NULL の場合は 0 を返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

ISNULL 関数 [その他]

リストの中から NULL でない最初の式を返します。この関数は COALESCE 関数と同じです。

構文

ISNULL(*expression*, *expression* [, …] **)**

パラメータ

expression　NULL かどうかがテストされる式。

2 つ以上の式を関数に渡します。すべての式は比較可能となっている必要があります。

備考

この関数の戻り値は、指定した式によって異なります。データベース・サーバが関数を評価するとき、まず、式の比較が可能なデータ型を検索します。該当するデータ型が見つかると、データベース・サーバは式を比較し、比較に使用したデータ型で結果を返します (NULL でない最初の式)。データベース・サーバは、一般に比較が可能なデータ型を見つけることができないと、エラーを返します。

参照

- 「COALESCE 関数 [その他]」 312 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値−66 を返します。

```
SELECT ISNULL( NULL ,-66, 55, 45, NULL, 16 );
```

LCASE 関数 [文字列]

文字列中のすべての文字を小文字に変換します。この関数は LOWER 関数と同じです。

構文

LCASE(*string-expression* **)**

パラメータ

string-expression　小文字に変換される文字列。

参照

- 「LOWER 関数 [文字列]」 340 ページ
- 「UCASE 関数 [文字列]」 371 ページ
- 「UPPER 関数 [文字列]」 371 ページ
- 「Ultra Light 文字列関数」 300 ページ

備考

LCASE 関数は LOWER 関数と同じです。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 chocolate を返します。

```
SELECT LCASE( 'ChoCOlatE' );
```

LEFT 関数 [文字列]

文字列の先頭からいくつかの文字を返します。

構文

LEFT(*string-expression*, *integer-expression* **)**

パラメータ

string-expression　文字列。

integer-expression　返す文字数。

備考

文字列にマルチバイト文字があり、適切な照合が使用されている場合、返されるバイト数が指定した文字数よりも大きい場合があります。

カラムの値より大きな *integer-expression* を指定できます。この場合、全体の値が返されます。

入力文字が文字長のセマンティックを使用している場合、可能であれば、戻り値は文字長のセマンティックの観点から記述されます。

参照

- 「RIGHT 関数 [文字列]」 355 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、Customers テーブルに含まれる各 Surname 値の最初の 5 文字を返します。

```
SELECT LEFT( Surname, 5) FROM Customers;
```

LENGTH 関数 [文字列]

指定した文字列の文字数を返します。

構文

LENGTH(*string-expression* **)**

パラメータ

string-expression 文字列。

備考

この関数を使用すると、文字列の長さをがわかります。たとえば、*string-expression* にカラム名を指定すると、カラムの値の長さがわかります。

文字列にマルチバイト文字があり、適切な照合が使用されている場合、LENGTH はバイト数ではなく、文字数を返します。文字列が BINARY データ型の場合、LENGTH 関数は BYTE_LENGTH 関数のように動作します。

> **注意**
> データ型が CHAR、VARCHAR、LONG VARCHAR の場合、LENGTH 関数と CHAR_LENGTH 関数を使用すると同じ結果が得られます。ただし、BINARY データ型とビット配列データ型には LENGTH 関数を使用します。

参照

- 「BYTE_LENGTH 関数 [文字列]」 307 ページ
- 「国際言語と文字セット」 『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 9 を返します。

```
SELECT LENGTH( 'chocolate' );
```

LESSER 関数 [その他]

2 つのパラメータ値のうち、より小さい値を返します。

構文

LESSER(*expression-1*, *expression-2* **)**

パラメータ

expression-1　比較される最初のパラメータ値。

expression-2　比較される 2 番目のパラメータ値。

備考

パラメータが等しい場合は、最初の値を返します。

参照

- 「GREATER 関数 [その他]」 329 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 5 を返します。

```
SELECT LESSER( 10, 5 ) FROM dummy;
```

LIST 関数 [集合]

カンマで区切られた値のリストを返します。

構文

LIST(
{ *string-expression* | **DISTINCT** *string-expression* }
[, *delimiter-string*] **)**

パラメータ

string-expression　文字列。通常はカラム名です。各ローに対して、カンマで区切られたリストに式の値が追加されます。

DISTINCT 文字列式　クエリで使用するカラム名などの式。そのカラムのユニークな値が、カンマで区切られたリストに 1 つずつ追加されます。

delimiter-string　リスト項目のデリミタ文字列。デフォルト設定はカンマです。NULL 値または空の文字列を指定した場合は、デリミタはありません。*delimiter-string* は定数です。

備考

NULL 値は、リストに追加されません。LIST (X) は、グループの各ローの、NULL 以外のすべての X の値を (デリミタ付きで) 連結させて返します。グループ内に明確な X 値を持ったローが 1 つ以上存在しない場合、LIST(X) は空の文字列を返します。

LIST 関数は Window 関数として使用できませんが、Window 関数の入力には使用できます。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、Employees テーブルのすべての住所を返します。

```
SELECT LIST( Street ) FROM Employees;
```

ソートされた ID '1751,591,1062,1191,992,888,318,184,1576,207,1684,1643,1607,1740,409,1507'

LOCATE 関数 [文字列]

異なる文字列内のある文字列の位置を返します。

構文

LOCATE(*string-expression-1*, *string-expression-2* [, *integer-expression*] **)**

パラメータ

string-expression-1　検索される文字列。

string-expression-2　検索する文字列。

integer-expression　文字列内の、検索を開始する位置。最初の文字の位置は 1 です。開始オフセットが負の場合は、最初に一致した文字列ではなく、最後に一致した文字列のオフセットを返します。負のオフセットは、文字列の末尾のどれだけの部分が検索から除外されるかを示します。除外されるバイト数は、(−1 * offset) −1 で計算されます。

備考

integer-expression を指定すると、文字列のオフセットから検索が開始します。

最初の文字列には長い文字列 (255 バイト以上) を指定できますが、2 番目の文字列は 255 バイトに制限されます。長い文字列を 2 番目の引数として指定すると、LOCATE 関数は NULL 値を返します。文字列が見つからない場合は、0 を返します。長さ 0 の文字列を検索すると、1 を返します。いずれかの引数が NULL の場合は、結果は NULL です。

マルチバイト文字が使用され、適切な照合が使用されている場合は、開始位置と返される値はバイトで計算した位置と異なる場合があります。

参照

- 「Ultra Light 文字列関数」 300 ページ
- 「CHARINDEX 関数 [文字列]」 310 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 8 を返します。

```
SELECT LOCATE(
  'office party this week - rsvp as soon as possible',
  'party',
  2 );
```

LOG 関数 [数値]

数の自然対数を返します。

構文

LOG(*numeric-expression* **)**

パラメータ

numeric-expression　数値。

参照

- 「LOG10 関数 [数値]」 339 ページ

備考

引数は、組み込みの数値データ型の値を返す式です。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメータが NULL 値の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、50 の自然対数を返します。

```
SELECT LOG( 50 );
```

LOG10 関数 [数値]

数値の対数 (基数 10) を返します。

構文

LOG10(*numeric-expression* **)**

パラメータ

numeric-expression　数値。

備考

引数は、組み込みの数値データ型の値を返す式です。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。パラメータが NULL 値の場合、結果は NULL 値になります。

参照

- 「LOG 関数 [数値]」 339 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、10 を底とする 50 の対数を返します。

```
SELECT LOG10( 50 );
```

LOWER 関数 [文字列]

文字列中のすべての文字を小文字に変換します。この関数は LCASE 関数と同じです。

構文

LOWER(*string-expression* **)**

パラメータ

string-expression 変換される文字列。

備考

LCASE 関数は LOWER 関数と同じです。

参照

- 「LCASE 関数 [文字列]」 335 ページ
- 「UCASE 関数 [文字列]」 371 ページ
- 「UPPER 関数 [文字列]」 371 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** コア機能。

例

次の文は、値 chocolate を返します。

```
SELECT LOWER( 'chOCOLate' );
```


LTRIM 関数 [文字列]

文字列の先行ブランクを削除します。

構文

LTRIM(*string-expression* **)**

パラメータ

string-expression　削除される文字列。

備考

結果の実際の長さは、式の長さから、削除される文字数を引いた値です。すべての文字を削除すると、結果は空の文字列です。

パラメータに NULL を指定できる場合、結果は NULL になります。

パラメータが NULL の場合、結果は NULL 値になります。

参照

- 「RTRIM 関数 [文字列]」 356 ページ
- 「TRIM 関数 [文字列]」 369 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

  SQL/2003 規格で定義される TRIM 仕様 (LEADING と TRAILING) は、それぞれ SQL Anywhere の LTRIM 関数と RTRIM 関数から指定されます。

例

次の文は、すべての先行ブランクを削除して、値 Test Message を返します。

```
SELECT LTRIM( '    Test Message' );
```

MAX 関数 [集合]

各ロー・グループで見つかった *expression* の最大値を返します。

構文 1

MAX(*expression* | **DISTINCT** *expression* **)**

パラメータ

expression　最大値が計算される式。通常はカラム名です。

DISTINCT 式　MAX(*expression*) の場合と同じ値を返します。万全を期すために含まれています。

備考

expression が NULL のローは、無視されます。グループにローが含まれていない場合は、NULL を返します。

参照

- 「MIN 関数 [集合]」 342 ページ

標準と互換性

- **SQL/2003** コア機能。構文 2 は機能 T611 です。

例

次の文は、Employees テーブル内の給与の最大値 138948.000 を返します。

```
SELECT MAX( Salary )
FROM Employees;
```

MIN 関数 [集合]

各ロー・グループで見つかった expression の最小値を返します。

構文 1

MIN(*expression* | **DISTINCT** *expression* **)**

パラメータ

expression 最小値が計算される式。通常はカラム名です。

DISTINCT 式 MIN(*expression*) の場合と同じ値を返します。万全を期すために含まれています。

備考

expression が NULL のローは、無視されます。グループにローが含まれていない場合は、NULL を返します。

参照

- 「MAX 関数 [集合]」 341 ページ

標準と互換性

- **SQL/2003** コア機能。構文 2 は機能 T611 です。

例

次の文は、Employees テーブル内の給与の最小値 24903.000 を返します。

```
SELECT MIN( Salary )
FROM Employees;
```

MINUTE 関数 [日付と時刻]

日時値の分コンポーネントを返します。

構文

MINUTE(*datetime-expression* **)**

パラメータ

datetime-expression　日時の値。

備考

返される値は日時の分に対応する 0 ー 59 の数値です。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 22 を返します。

```
SELECT MINUTE( '1998-07-13 12:22:34' );
```

MINUTES 関数 [日付と時刻]

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02-29 からの分数を返します。

  注意
  0000-02-29 は実際の日付を指すための値ではありません。日付アルゴリズムで使用される日付です。

- 2 つのタイムスタンプを指定すると、2 つの時刻の間の分数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- 1 つの日付と整数を指定すると、指定した日付に、指定した整数の分数が加算されます。代わりに、DATEADD 関数を使用します。

構文 1：整数

MINUTES([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

MINUTES(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する分数。*integer-expression* が負の場合、日時の値から適切な分数が引かれます。整数式を指定する場合は、*datetime-expression* を DATETIME データ型として明示的にキャストしてください。

備考

この関数は整数を返すので、構文 1 で 4083-03-23 02:08:00 以上のタイムスタンプを使用すると、オーバフローが起こる場合があります。

参照

- 「CAST 関数 [データ型変換]」 308 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 240 を返します。これは、2 番目のタイムスタンプが、最初のタイムスタンプの 240 秒後であることを示します。2 つ目の例 (DATEDIFF) の使用をおすすめします。

```
SELECT MINUTES( '1999-07-13 06:07:12',
   '1999-07-13 10:07:12' );

SELECT DATEDIFF( minute,
    '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' );
```

次の文は、値 1051040527 を返します。

```
SELECT MINUTES( '1998-07-13 06:07:12' );
```

次の文は、タイムスタンプ 1999-05-12 21:10:07.000 を返します。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT MINUTES( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5);

SELECT DATEADD( minute, 5, '1999-05-12 21:05:07' );
```

MOD 関数 [数値]

整数を整数で割ったときの余りを返します。

構文

MOD(*dividend*, *divisor* **)**

パラメータ

dividend　被除数 (分数の分子)。

divisor　除数 (分数の分母)。

備考

被除数が負の場合、除算の結果は負または 0 になります。除数の符号は計算結果に影響を与えません。

参照

- 「REMAINDER 関数 [数値]」 352 ページ

標準と互換性

- **SQL/2003**　コア SQL に含まれない SQL 基本機能。

例

次の文は、値 2 を返します。

```
SELECT MOD( 5, 3 );
```

MONTH 関数 [日付と時刻]

指定した日付の月を返します。

構文

MONTH(*date-expression* **)**

パラメータ

date-expression　日時の値。

備考

返される値は日時の月に対応する 1 ー 12 の数値です。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 7 を返します。

```
SELECT MONTH( '1998-07-13' );
```

MONTHNAME 関数 [日付と時刻]

日付から月の名前を返します。

構文

MONTHNAME(*date-expression* **)**

パラメータ

date-expression　日時の値。

備考

結果が数値 (2 月を示す 2 など) の場合でも、MONTHNAME 関数は文字列を返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 September を返します。

```
SELECT MONTHNAME( '1998-09-05' );
```

MONTHS 関数 [日付と時刻]

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02 からの月数を返します。

  注意
  0000-02 は実際の日付を指すための値ではありません。日付アルゴリズムで使用される日付です。

- 2 つのタイムスタンプを指定すると、2 つの日付の間の月数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- 1 つの日付と整数を指定すると、指定した日付に、指定した整数の分数が加算されます。代わりに、DATEADD 関数を使用します。

構文 1：整数

MONTHS([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

MONTHS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する月数。*integer-expression* が負の場合、日時の値から適切な月数が引かれます。*integer-expression* を指定する場合は、*datetime-expression* を datetime データ型として明示的にキャストしてください。

データ型のキャストの詳細については、「CAST 関数 [データ型変換]」 308 ページを参照してください。

備考

MONTHS の値を求めるには、2 つの日付の間に月の最初の日がいくつあるかを計算します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 2 を返します。これは、2 番目の日付が、最初の日付の 2 か月後であることを示します。2 つ目の例 (DATEDIFF) の使用をおすすめします。

```
SELECT MONTHS( '1999-07-13 06:07:12',
  '1999-09-13 10:07:12' );

SELECT DATEDIFF( month,
  '1999-07-13 06:07:12',
  '1999-09-13 10:07:12' );
```

次の文は、値 23981 を返します。

```
SELECT MONTHS( '1998-07-13 06:07:12' );
```

次の文は、タイムスタンプ 1999-10-12 21:05:07.000 を返します。2 つ目の例 (DATEADD) の使用をおすすめします。

```
SELECT MONTHS( CAST( '1999-05-12 21:05:07'
AS DATETIME ), 5);

SELECT DATEADD( month, 5, '1999-05-12 21:05:07' );
```

NEWID 関数 [その他]

UUID (ユニバーサル・ユニーク識別子) 値を生成します。UUID は、GUID (グローバル・ユニーク識別子) と同じです。

構文

NEWID( )

パラメータ

NEWID 関数に関連付けられているパラメータはありません。

備考

NEWID 関数は、ユニークな識別子の値を生成します。この関数は、カラムの DEFAULT 句で使用できます。

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。あるコンピュータで生成される値は、他のコンピュータで生成される値と一致しないようになっています。したがって、同期環境やレプリケーション環境で、これらをキーとして使用することもできます。

参照

- 「NEWID デフォルト」『SQL Anywhere サーバ - SQL の使用法』
- 「STRTOUUID 関数 [文字列]」 365 ページ
- 「UUIDTOSTR 関数 [文字列]」 372 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、2 つのカラムを持つテーブル mytab を作成します。カラム pk は uniqueidentifier データ型とし、NEWID 関数をデフォルト値として割り当てます。カラム c1 は integer データ型です。

```
CREATE TABLE mytab(
  pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
  c1 INT );
```

次の文は、ユニーク識別子を文字列として返します。

```
SELECT NEWID();
```

たとえば、戻り値が 96603324-6FF6-49DE-BF7D-F44C1C7E6856 の場合もあります。

NOW 関数 [日付と時刻]

現在の年、月、日、時、分、秒、秒以下を返します。精度はシステム・クロックの精度によって制限されます。

構文

NOW(*)

備考

NOW 関数が返す情報は、GETDATE 関数と CURRENT TIMESTAMP 特別値が返す情報と同じです。

参照

- 「GETDATE 関数 [日付と時刻]」 328 ページ
- 「CURRENT TIMESTAMP 特別値」 259 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、現在の日付と時刻を返します。

```
SELECT NOW( * );
```

NULLIF 関数 [その他]

式を比較して、CASE 式の省略形を提供します。

構文

NULLIF(*expression-1*, *expression-2* **)**

パラメータ

expression-1　比較される式。

expression-2　比較される式。

備考

NULLIF は 2 つの式の値を比較します。

1 番目の式と 2 番目の式が等しい場合、NULLIF は NULL を返します。

1 番目の式と 2 番目の式が異なる場合、または 2 番目の式が NULL の場合、NULLIF は 1 番目の式を返します。

NULLIF 関数は、いくつかの CASE 式の簡単な作成方法を提供します。

参照

- 「CASE 式」 276 ページ

標準と互換性

- **SQL/2003**　コア機能。

例

次の文は、値 a を返します。

```
SELECT NULLIF( 'a', 'b' );
```

次の文は、NULL を返します。

```
SELECT NULLIF( 'a', 'a' );
```

PATINDEX 関数 [文字列]

文字列のパターンが最初に出現した開始位置を表す整数を返します。

構文

PATINDEX('%*pattern***%'**, *string_expression* **)**

パラメータ

pattern　検索するパターン。先頭の % ワイルドカードを省略すると、PATINDEX 関数は、パターンが文字列の先頭に出現する場合は 1 を返し、それ以外の場合は 0 を返します。

Ultra Light のパターンは、次のワイルドカードを使用します。

ワイルドカード	一致するもの
_ (アンダースコア)	任意の 1 文字
% (パーセント記号)	0 個以上の文字からなる任意の文字列

ワイルドカード	一致するもの
[]	指定範囲内、または一連の指定文字の任意の 1 文字
[^]	指定範囲外、または一連の指定文字以外の任意の 1 文字

string-expression　パターンを検索する文字列。

備考

PATINDEX 関数は、パターンが最初に出現した開始位置を返します。パターンが見つからない場合は、0 を返します。

参照

- 「LOCATE 関数 [文字列]」 338 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 2 を返します。

```
SELECT PATINDEX( '%hoco%', 'chocolate' );
```

次の文は、値 11 を返します。

```
SELECT PATINDEX( '%4_5_', '0a1A 2a3A 4a5A' );
```

PI 関数 [数値]

PI の数値を返します。

構文

PI(*)

標準と互換性

- **SQL/2003**　ベンダ拡張。

備考

この関数は DOUBLE 値を返します。

例

次の文は、値 3.141592653… を返します。

```
SELECT PI( * );
```

POWER 関数 [数値]

数のべき乗を表す数を計算します。

構文

POWER(*numeric-expression-1*, *numeric-expression-2* **)**

パラメータ

numeric-expression-1　べき乗の底。

numeric-expression-2　指数。

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。引数のいずれかが NULL の場合、結果は NULL 値になります。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 64 を返します。

```
SELECT POWER( 2, 6 );
```

QUARTER 関数 [日付と時刻]

指定した日付式から、四半期を示す数を返します。

構文

QUARTER(*date-expression* **)**

パラメータ

date-expression　日付。

備考

四半期は次のとおりです。

Quarter	期間 (開始日と終了日を含む)
1	1 月 1 日ー 3 月 31 日
2	4 月 1 日ー 6 月 30 日
3	7 月 1 日ー 9 月 30 日
4	10 月 1 日ー 12 月 31 日

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 2 を返します。

```
SELECT QUARTER( '1987/05/02' );
```

RADIANS 関数 [数値]

度数をラジアンに変換します。

構文

RADIANS(*numeric-expression* **)**

パラメータ

numeric-expression　度数。この角度をラジアンに変換します。

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、約 0.5236 の値を返します。

```
SELECT RADIANS( 30 );
```

REMAINDER 関数 [数値]

整数を整数で割ったときの余りを返します。

構文

REMAINDER(*dividend*, *divisor* **)**

パラメータ

dividend　被除数 (分数の分子)。

divisor　除数 (分数の分母)。

備考

または、MOD 関数も使用できます。

参照

- 「MOD 関数 [数値]」 344 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 2 を返します。

```
SELECT REMAINDER( 5, 3 );
```

REPEAT 関数 [文字列]

文字列を指定された回数だけ連結します。

構文

REPEAT(*string-expression*, *integer-expression* **)**

パラメータ

string-expression 繰り返される文字列。

integer-expression 文字列を繰り返す回数。*integer-expression* が負の場合は、空の文字列を返します。

備考

実際の結果文字列の長さが戻り値の最大長を超えると、エラーが発生します。この場合、結果は文字列に使用できる最大サイズまでトランケートされます。

または、REPLICATE 関数も使用できます。

参照

- 「REPLICATE 関数 [文字列]」 354 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 repeatrepeatrepeat を返します。

```
SELECT REPEAT( 'repeat', 3 );
```

REPLACE 関数 [文字列]

文字列を別の文字列で置換し、新しい結果を返します。

構文

REPLACE(*original-string*, *search-string*, *replace-string* **)**

パラメータ

いずれかの引数が NULL の場合、NULL を返します。

original-string　検索される文字列。任意の長さの文字列を指定できます。

search-string　検索して *replace-string* に置換する文字列。この文字列は 255 バイトに制限されています。*search-string* が空の文字列の場合は、元の文字列を未変更のまま返します。

replace-string　*search-string* と置き換える置換文字列。任意の長さの文字列を指定できます。*replacement-string* が空の文字列の場合は、すべての *search-string* が削除されます。

備考

この関数はすべてのオカレンスを置換します。

参照

- 「SUBSTRING 関数 [文字列]」 366 ページ
- 「CHARINDEX 関数 [文字列]」 310 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 xx.def.xx.ghi を返します。

```
SELECT REPLACE( 'abc.def.abc.ghi', 'abc', 'xx' );
```

次の文は、ALTER PROCEDURE 文を含む結果セットを生成します。ALTER PROCEDURE を実行すると、名前が変更されたテーブルを参照する格納済みプロシージャが修復されます (テーブル名を一意にすることをおすすめします)。

```
SELECT REPLACE(
  REPLACE( proc_defn, 'OldTableName', 'NewTableName' ),
  'CREATE PROCEDURE',
  'ALTER PROCEDURE')
FROM SYS.SYSPROCEDURE
WHERE proc_defn LIKE '%OldTableName%';
```

LIST 関数にカンマ以外の区切り記号を使用する場合は、次のようになります。

```
SELECT REPLACE( LIST( table_id ), ',', '--')
FROM  SYS.SYSTAB
WHERE table_id <= 5;
```

REPLICATE 関数 [文字列]

文字列を指定された回数だけ連結します。

構文

REPLICATE(*string-expression*, *integer-expression* **)**

パラメータ

string-expression　繰り返される文字列。

integer-expression　文字列を繰り返す回数。

備考

実際の結果文字列の長さが戻り値の最大長を超えると、エラーが発生します。この場合、結果は文字列に使用できる最大サイズまでトランケートされます。

または、REPEAT 関数も使用できます。

参照

- 「REPEAT 関数 [文字列]」 353 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 repeatrepeatrepeat を返します。

```
SELECT REPLICATE( 'repeat', 3 );
```

RIGHT 関数 [文字列]

文字列の一番右側にある文字を返します。

構文

RIGHT(*string-expression*, *integer-expression* **)**

パラメータ

string-expression　左側をトランケートされる文字列。

integer-expression　返される文字列の末尾の文字数。

備考

文字列にマルチバイト文字があり、適切な照合が使用されている場合、返されるバイト数が指定した文字数よりも大きい場合があります。

カラムの値より大きな *integer-expression* を指定できます。この場合、全体の値が返されます。

入力文字が文字長のセマンティックを使用している場合、可能であれば、戻り値は文字長のセマンティックの観点から記述されます。

参照

- 「LEFT 関数 [文字列]」 335 ページ
- 「国際言語と文字セット」 『SQL Anywhere サーバ - データベース管理』
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、Customers テーブルに含まれる各 Surname 値の最後の 5 文字を返します。

```
SELECT RIGHT( Surname, 5) FROM Customers;
```

ROUND 関数 [数値]

integer-expression で指定した小数点以下の桁数に *numeric-expression* を丸めます。

構文

ROUND(*numeric-expression*, *integer-expression* **)**

パラメータ

numeric-expression 関数に渡される、丸めの対象となる数。

integer-expression 正の整数は、丸めを行う小数点の右側の有効桁数を指定します。負の式は、丸めを行う小数点の左側の有効桁数を指定します。

備考

この関数の結果は numeric または double です。数値の結果があり、整数 *integer-expression* が負の値の場合、精度は 1 ずつ増えます。

参照

- 「TRUNCNUM 関数 [数値]」 370 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 123.200 を返します。

```
SELECT ROUND( 123.234, 1 );
```

RTRIM 関数 [文字列]

後続ブランクが削除された文字列を返します。

構文

RTRIM(*string-expression* **)**

パラメータ

string-expression　削除される文字列。

備考

結果の実際の長さは、式の長さから、削除される文字数を引いた値です。すべての文字を削除すると、結果は空の文字列です。

引数が null の場合、結果は null 値になります。

参照

- 「TRIM 関数 [文字列]」 369 ページ
- 「LTRIM 関数 [文字列]」 341 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

  SQL/2003 規格で定義される TRIM 仕様 (LEADING と TRAILING) は、それぞれ SQL Anywhere の LTRIM 関数と RTRIM 関数から指定されます。

例

次の文は、すべての後続ブランクが削除された文字列 Test Message を返します。

```
SELECT RTRIM( 'Test Message     ' );
```

SECOND 関数 [日付と時刻]

指定した日付の秒を返します。

構文

SECOND(*datetime-expression* **)**

パラメータ

datetime-expression　日時の値。

備考

指定した日時の秒に相当する 0 ー 59 の数を返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 25 を返します。

```
SELECT SECOND( '1998-07-13 21:21:25' );
```

SECONDS 関数 [日付と時刻]

この関数の動作は、指定した内容によって変わります。

- 1 つの日付を指定すると、0000-02-29 からの秒数を返します。

  注意
  0000-02-29 は実際の日付を指すための値ではありません。日付アルゴリズムで使用される日付です。

- 2 つのタイムスタンプを指定すると、2 つのタイムスタンプの間の秒数を整数で返します。代わりに、DATEDIFF 関数を使用します。
- 1 つの日付と整数を指定すると、指定したタイムスタンプに、指定した整数の秒数が加算されます。代わりに、DATEADD 関数を使用します。

構文 1：整数

SECONDS([*datetime-expression*,] *datetime-expression* **)**

構文 2：タイムスタンプ

SECONDS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する秒数。*integer-expression* が負の場合、日時の値から適切な分数が引かれます。整数式を指定する場合は、*datetime-expression* を datetime データ型として明示的にキャストしてください。

参照

- 「CAST 関数 [データ型変換]」 308 ページ
- 「DATEADD 関数 [日付と時刻]」 318 ページ
- 「DATEDIFF 関数 [日付と時刻]」 318 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 14400 を返します。これは、2 番目のタイムスタンプが、最初のタイムスタンプの 14400 秒後であることを示します。

```
SELECT SECONDS( '1999-07-13 06:07:12',
  '1999-07-13 10:07:12' );

SELECT DATEDIFF( second,
```

```
    '1999-07-13 06:07:12',
    '1999-07-13 10:07:12' );
```

次の文は、値 63062431632 を返します。

```
SELECT SECONDS( '1998-07-13 06:07:12' );
```

次の文は、日時 1999-05-12 21:05:12.0 を返します。

```
SELECT SECONDS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5);

SELECT DATEADD( second, 5, '1999-05-12 21:05:07' );
```

SHORT_PLAN 関数 [その他]

SQL 文の Ultra Light プランの最適化方法を短い記述の文字列で返します。この記述は、EXPLANATION 関数が返す記述と同じです。

構文

SHORT_PLAN(*string-expression* **)**

備考

クエリによっては、SQL Anywhere に選択したプランと Ultra Light の実行プランが異なる場合があります。

パラメータ

string-expression　SQL 文。通常は SELECT 文ですが、UPDATE 文または DELETE 文も指定できます。

参照

- 「EXPLANATION 関数 [その他]」 327 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、SELECT 文を文字列パラメータとして渡し、クエリを実行するためのプランを返します。

```
SELECT EXPLANATION(
  'SELECT * FROM Departments WHERE DepartmentID > 100' );
```

この情報は、追加するインデックスの決定や、良いパフォーマンスを得るためのデータベース構造の決定に役立ちます。

Interactive SQL では、SQL 文のプランを [結果] ウィンドウ枠の [プラン] タブに表示できます。

SIGN 関数 [数値]

数の符号を返します。

構文

SIGN(*numeric-expression* **)**

パラメータ

numeric-expression　符号が返される数。

備考

負の数を指定すると、SIGN 関数は -1 を返します。

0 を指定すると、SIGN 関数は 0 を返します。

正の数を指定すると、SIGN 関数は 1 を返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 -1 を返します。

```
SELECT SIGN( -550 );
```

SIMILAR 関数 [文字列]

2 つの文字列の類似性を示す数を返します。

構文

SIMILAR(*string-expression-1*, *string-expression-2* **)**

パラメータ

string-expression-1　比較される最初の文字列。

string-expression-2　比較される 2 番目の文字列。

備考

SIMILAR 関数は、2 つの文字列の類似性を表す 0 ー 100 の整数を返します。結果は、2 つの文字列の文字が一致している割合として解釈できます。値が 100 の場合は、2 つの文字列は同じです。

この関数を使用して、名前 (顧客名など) のリストを修正できます。顧客がわずかに異なる名前で重複して登録されている場合があります。テーブルをそのテーブル自体とジョインし、90% より大きく、100% 未満の類似性をすべてレポートします。

SIMILAR 関数で実行される計算は、単に文字数が一致する場合よりも複雑になります。

参照

- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は値 75 を返します。2 つの値が 75 % 前後であることを示します。

```
SELECT SIMILAR( 'toast', 'coast' );
```

SIN 関数 [数値]

数のサインを返します。

構文

SIN(*numeric-expression* **)**

パラメータ

numeric-expression　角度 (ラジアン)。

備考

SIN 関数は、引数のサインを返します。この引数はラジアン値で表現される角度です。SIN 関数と ASIN 関数は逆変換の演算です。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「ASIN 関数 [数値]」 304 ページ
- 「COS 関数 [数値]」 314 ページ
- 「COT 関数 [数値]」 315 ページ
- 「TAN 関数 [数値]」 368 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、0.52 の SIN 値を返します。

```
SELECT SIN( 0.52 );
```

SOUNDEX 関数 [文字列]

文字列の発音を表す数を返します。

構文

SOUNDEX(*string-expression* **)**

パラメータ

string-expression　評価される文字列。

備考

文字列の SOUNDEX 関数の値は、先頭の文字とそれに続く H、Y、W 以外の 3 つの子音がベースになっています。文字列の最初の文字である場合を除き、*string-expression* の母音は無視されます。同じ文字が 2 つ続く場合は 1 文字としてカウントされます。たとえば、apple という単語は、文字 A、P、L、S がベースになります。

マルチバイト文字は、SOUNDEX 関数では無視されます。

完全とは言えませんが、SOUNDEX 関数は通常、類似発音で同じ文字で始まる語句に対して同じ数を返します。

SOUNDEX 関数は、英単語の処理に最も優れています。他の言語の処理は英語の場合よりも劣ります。

参照

- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、それぞれの名前の発音を表す 2 つの同じ数 3827 を返します。

```
SELECT SOUNDEX( 'Smith' ), SOUNDEX( 'Smythe' );
```

SPACE 関数 [文字列]

指定した数のスペースを返します。

構文

SPACE(*integer-expression* **)**

パラメータ

integer-expression　返すスペースの数。

備考

integer-expression が負の場合は、null 文字列を返します。

参照

- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、10 のスペースから成る文字列を返します。

```
SELECT SPACE( 10 );
```

SQRT 関数 [数値]

数の平方根を返します。

構文

SQRT(*numeric-expression* **)**

パラメータ

numeric-expression　平方根が計算される数。

備考

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 3 を返します。

```
SELECT SQRT( 9 );
```

STR 関数 [文字列]

指定した数に相当する文字列を返します。

構文

STR(*numeric-expression* [, *length* [, *decimal*]] **)**

パラメータ

numeric-expression　−1E126 と 1E127 との間の任意の概数 (浮動小数点、実数、または倍精度)。

length　返される文字数 (小数点、小数点の左右のすべての桁、ブランクを含む)。デフォルトは 10 です。

decimal　返される小数点以下の桁数。デフォルトは 0 です。

備考

数の整数部分が指定した長さに合わない場合は、指定した長さの文字列がすべてアスタリスクで埋められて返されます。たとえば、次の文は *** を返します。

```
SELECT STR( 1234.56, 3 );
```

注意
サポートされる最大長は 128 です。1 ー 128 の範囲にない長さでは結果が NULL になります。

参照

- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、6 スペースと 1235 (合計 10 文字) の文字列を返します。

```
SELECT STR( 1234.56 );
```

次の文は、結果 1234.6 を返します。

```
SELECT STR( 1234.56, 6, 1 );
```

STRING 関数 [文字列]

1 つ以上の文字列を連結して 1 つの長い文字列にします。

構文

STRING(*string-expression* **[, ⋯])**

パラメータ

string-expression 評価される文字列。

引数を 1 つだけ指定する場合は、1 つの式に変換されます。複数の引数を指定する場合は、連結されて 1 つの文字列になります。

備考

数値または日付をパラメータとして指定した場合は、文字列に変換されてから連結されます。また、1 つの式を唯一のパラメータとして指定すると、STRING 関数を使用して、その式を文字列に変換できます。

すべてのパラメータが NULL の場合、STRING は NULL を返します。NULL でないパラメータが存在すると、NULL パラメータはすべて空の文字列として処理されます。

参照

- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 *testing123* を返します。

```
SELECT STRING( 'testing', NULL, 123 );
```

STRTOUUID 関数 [文字列]

文字列の値をユニークな識別子 (UUID または GUID) の値に変換します。

> **新しいデータベースでは不要**
> バージョン 9.0.2 より前に作成されたデータベースでは、、UUID 値のバイナリ表現と文字列表現の間を変換するための STRTOUUID 関数と UUIDTOSTR 関数が必要です。
> バージョン 9.0.2 以降を使用して作成されたデータベースでは、UNIQUEIDENTIFIER データ型はネイティブ・データ型であり、。これらのバージョンでは STRTOUUID 関数と UUIDTOSTR 関数を使用する必要はありません。
> 詳細については、「Ultra Light のデータ型」 262 ページを参照してください。

構文

STRTOUUID(*string-expression* **)**

パラメータ

string-expression　フォーマットが *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* の文字列。

備考

フォーマットが *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* の文字列をユニークな識別子の値に変換します。ここで、*x* は 16 進数です。

この関数は、UUID 値をデータベースに挿入するときに役立ちます。

参照

- 「UUIDTOSTR 関数 [文字列]」 372 ページ
- 「NEWID 関数 [その他]」 347 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

```
CREATE TABLE T1 (
   pk UNIQUEIDENTIFIER PRIMARY KEY, c1 INT );
INSERT INTO T1 ( pk, c1 )
VALUES ( STRTOUUID('12345678-1234-5678-9012-123456789012'), 1 );
```


STUFF 関数 [文字列]

1 つの文字列から指定した文字数を削除して、別の文字列に置き換えます。

構文

STUFF(*string-expression-1*, *start*, *length*, *string-expression-2* **)**

パラメータ

string-expression-1　STUFF 関数によって変更される文字列。

start　削除を開始する文字の位置。文字列の先頭文字の位置を 1 とします。

length　削除する文字数。

string-expression-2　挿入する文字列。STUFF 関数を使用して文字列の一部を削除するには、NULL の置換文字列を使用します。

備考

参照

- 「INSERTSTR 関数 [文字列]」 333 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 chocolate pie を返します。

```
SELECT STUFF( 'chocolate cake', 11, 4, 'pie' );
```

SUBSTRING 関数 [文字列]

文字列の部分文字列を返します。

構文

{ **SUBSTRING** | **SUBSTR** } **(** *string-expression*, *start*
[, *length*] **)**

パラメータ

string-expression　部分文字列が返される文字列。

start　文字単位で指定した、返される部分文字列の開始位置。

length　文字単位で指定した、返される部分文字列の長さ。*length* を指定すると、部分文字列は指定した長さに限定されます。

備考

Ultra Light では、データベースに ansi_substring オプションがありません。それでも、SUBSTR 関数は、ansi_substring がデフォルトでオンに設定されているように動作します。つまり、関数の動作は ANSI/ISO SQL/2003 の動作に対応します。

- **Start 値**　文字列の先頭文字の位置を 1 とします。負またはゼロの開始オフセットは、文字列の左側が文字以外で埋められたように扱われます。
- **Length 値**　正の *length* は、部分文字列が開始位置の右側から *length* 文字で終わることを示します。

  負の *length* はエラーを返します。

string-expression が binary データ型の場合、SUBSTRING 関数は BYTE_SUBSTR のように動作します。

SUBSTRING 関数では、開始オフセットを正でない値に設定したり負の長さを指定したりしないことをおすすめします。可能なかぎり、SUBSTRING 関数の代わりに LEFT 関数または RIGHT 関数を使用してください。

入力文字が文字長のセマンティックを使用している場合、可能であれば、戻り値は文字長のセマンティックの観点から記述されます。

参照

- 「BYTE_SUBSTR 関数 [文字列]」 307 ページ
- 「LEFT 関数 [文字列]」 335 ページ
- 「RIGHT 関数 [文字列]」 355 ページ
- 「CHARINDEX 関数 [文字列]」 310 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　コア機能。

例

次の表に、SUBSTRING 関数を SELECT 文で使用して、ansi_substring オプションを On と Off に設定したときに返す値を示します。

例	On に設定された ansi_substring の結果	Off に設定された ansi_substring の結果
SUBSTRING('front yard', 1, 4)	fron	fron
SUBSTRING('back yard', 6, 4)	yard	yard
SUBSTR('abcdefgh', 0, -2)	エラーを返します	gh
SUBSTR('abcdefgh', -2, 2)	空の文字列を返します	gh
SUBSTR('abcdefgh', 2, -2)	エラーを返します	ab

例	On に設定された ansi_substring の結果	Off に設定された ansi_substring の結果
SUBSTR('abcdefgh', 2, -4)	エラーを返します	ab
SUBSTR('abcdefgh', 2, -1)	エラーを返します	b

SUM 関数 [集合]

ロー・グループごとに、指定された式の合計を返します。

構文 1

SUM(*expression* | **DISTINCT** *expression* **)**

パラメータ

expression　合計するオブジェクト。通常はカラム名です。

DISTINCT 式　入力の *expression* で一意の値の合計を計算します。

備考

指定された式が NULL のローは含まれません。

グループにローが含まれていない場合は、NULL を返します。

参照

- 「COUNT 関数 [集合]」 316 ページ
- 「AVG 関数 [集合]」 306 ページ

標準と互換性

- **SQL/2003**　コア機能。構文 2 は T611 です。

例

次の文は、値 3749146.740 を返します。

```
SELECT SUM( Salary )
FROM Employees;
```

TAN 関数 [数値]

数のタンジェントを返します。

構文

TAN(*numeric-expression* **)**

パラメータ

numeric-expression　角度 (ラジアン)。

備考

ATAN 関数と TAN 関数は逆変換の演算です。

この関数は、引数を DOUBLE に変換し、計算を倍精度浮動小数点で行い、結果を DOUBLE で返します。

参照

- 「COS 関数 [数値]」 314 ページ
- 「SIN 関数 [数値]」 361 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、0.52 の tan 値を返します。

```
SELECT TAN( 0.52 );
```

TODAY 関数 [日付と時刻]

現在の日付を返します。

構文

TODAY(*)

備考

従来の CURRENT DATE 関数の位置にこの構文を使用します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、システム・クロックによる現在の日付を返します。

```
SELECT TODAY( * ) ;
SELECT CURRENT DATE
```

TRIM 関数 [文字列]

先行ブランクと後続ブランクを文字列から削除します。

構文

TRIM(*string-expression* **)**

パラメータ

string-expression　削除される文字列。

備考

参照

- 「LTRIM 関数 [文字列]」 341 ページ
- 「RTRIM 関数 [文字列]」 356 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** TRIM 関数は SQL/2003 のコア機能です。

  SQL Anywhere では、SQL/2003 で定義されている追加のパラメータ *trim specification* と *trim character* をサポートしていません。SQL Anywhere の TRIM の実装は、BOTH の TRIM 仕様に対応しています。

  SQL/2003 規格で定義される TRIM 仕様 (LEADING と TRAILING) は、それぞれ SQL Anywhere の LTRIM 関数と RTRIM 関数から指定されます。

例

次の文は、先行ブランクと後続ブランクなしの値 chocolate を返します。

```
SELECT TRIM( '   chocolate   ' );
```

TRUNCNUM 関数 [数値]

指定した桁数で小数点以下を切り捨てます。

構文

{ **TRUNCNUM** | **"TRUNCATE"** }(*numeric-expression*, *integer-expression*)

パラメータ

numeric-expression トランケートされる数。

integer-expression 正の整数は、丸めを行う小数点の右側の有効桁数を指定します。負の式は、丸めを行う小数点の左側の有効桁数を指定します。

備考

いずれかのパラメータが NULL 値の場合、結果は NULL 値になります。

参照

- 「ROUND 関数 [数値]」 356 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 600 を返します。

```
SELECT TRUNCNUM( 655, -2 );
```

次の文は、値 655.340 を返します。

```
SELECT TRUNCNUM( 655.348, 2 );
```

UCASE 関数 [文字列]

文字列中のすべての文字を大文字に変換します。この関数は UPPER 関数と同じです。

構文

UCASE(*string-expression* **)**

パラメータ

string-expression　大文字に変換される文字列。

備考

UCASE 関数は UPPER 関数に似ています。

参照

- 「UPPER 関数 [文字列]」 371 ページ
- 「LCASE 関数 [文字列]」 335 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 CHOCOLATE を返します。

```
SELECT UCASE( 'ChocoLate' );
```

UPPER 関数 [文字列]

文字列中のすべての文字を大文字に変換します。この関数は UCASE 関数と同じです。

構文

UPPER(*string-expression* **)**

パラメータ

string-expression　大文字に変換される文字列。

備考

UCASE 関数は UPPER 関数に似ています。

参照

- 「UCASE 関数 [文字列]」 371 ページ
- 「LCASE 関数 [文字列]」 335 ページ

- 「LOWER 関数 [文字列]」 340 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 CHOCOLATE を返します。

```
SELECT UPPER( 'ChocoLate' );
```

UUIDTOSTR 関数 [文字列]

ユニークな識別子の値 (UUID または GUID) を文字列の値に変換します。

新しいデータベースでは不要
バージョン 9.0.2 より前に作成されたデータベースでは、、UUID 値のバイナリ表現と文字列表現の間を変換するための STRTOUUID 関数と UUIDTOSTR 関数が必要です。
バージョン 9.0.2 以降を使用して作成されたデータベースでは、UNIQUEIDENTIFIER データ型はネイティブ・データ型であり、。これらのバージョンでは STRTOUUID 関数と UUIDTOSTR 関数を使用する必要はありません。
詳細については、「Ultra Light のデータ型」 262 ページを参照してください。

構文

UUIDTOSTR(*uuid-expression* **)**

パラメータ

uuid-expression ユニークな識別子の値。

備考

ユニークな識別子を、フォーマットが *xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx* の文字列値に変換します。ここで、x は 16 進数です。バイナリ値が有効な uniqueidentifier でない場合は、NULL を返します。

この関数は、UUID 値を表示する場合に役立ちます。

参照

- 「NEWID 関数 [その他]」 347 ページ
- 「STRTOUUID 関数 [文字列]」 365 ページ
- 「Ultra Light 文字列関数」 300 ページ

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、2 つのカラムを持つテーブル mytab を作成します。カラム pk は uniqueidentifier データ型で、カラム c1 は integer データ型です。それぞれに値 1 と値 2 を持つ 2 つのローをカラム c1 に挿入します。

```
CREATE TABLE mytab(
   pk UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
   c1 INT );
INSERT INTO mytab( c1 ) values ( 1 );
INSERT INTO mytab( c1 ) values ( 2 );
```

次の SELECT 文を実行すると、新規に作成したテーブルのすべてのデータを返します。

```
SELECT * FROM mytab;
```

2 つのカラムと 2 つのローを持ったテーブルが表示されます。カラム pk に表示される値は、バイナリ値です。

ユニークな識別子の値を読みやすい形式に変換するには、次のコマンドを実行します。

```
SELECT UUIDTOSTR(pk), c1 FROM mytab;
```

UUIDTOSTR 関数は、バージョン 9.0.2 以降で作成されたデータベースでは不要です。

WEEKS 関数 [日付と時刻]

2 つの日付を指定すると、2 つの日付の間の週数を整数で返します。この目的で使用する場合は、「DATEDIFF 関数 [日付と時刻]」 318 ページ を代わりに使用することをおすすめします。

1 つの日付を指定すると、0000-02-29 からの週数を返します。

1 つの日付と整数を指定すると、指定した日付に、指定した整数の週数が加算されます。この目的で使用する場合は、「DATEADD 関数 [日付と時刻]」 318 ページ を代わりに使用することをおすすめします。

構文 1 は、整数を返します。構文 2 は、タイムスタンプを返します。

構文 1

WEEKS([*datetime-expression*,] *datetime-expression* **)**

構文 2

WEEKS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する週数。*integer-expression* が負の場合、日時の値から適切な週数が引かれます。*integer-expression* を指定する場合は、*datetime-expression* を datetime データ型として明示的にキャストしてください。

データ型のキャストの詳細については、「CAST 関数 [データ型変換]」 308 ページを参照してください。

備考

2 つの日付の間の週数は、日曜日の数で計算します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文は、値 8 を返します。これは、2 番目の日付が、最初の日付の 8 週間後であることを示します。2 番目の形式 (DATEDIFF) の使用をおすすめします。

```
SELECT WEEKS( '1999-07-13 06:07:12',
  '1999-09-13 10:07:12' );

SELECT DATEDIFF( week,
  '1999-07-13 06:07:12',
  '1999-09-13 10:07:12' );
```

次の文は、値 104270 を返します。

```
SELECT WEEKS( '1998-07-13 06:07:12' );
```

次の文は、タイムスタンプ 1999-06-16 21:05:07.0 を返します。2 番目の形式 (DATEADD) の使用をおすすめします。

```
SELECT WEEKS( CAST( '1999-05-12 21:05:07'
AS TIMESTAMP ), 5);

SELECT DATEADD( week, 5, '1999-05-12 21:05:07' );
```

YEAR 関数 [日付と時刻]

タイムスタンプ値をパラメータとして受け取り、そのタイムスタンプで指定された年を返します。

構文

YEAR(*datetime-expression* **)**

パラメータ

datetime-expression　日付、時刻、またはタイムスタンプ。

備考

値は short 値として返されます。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の例は、値 2001 を返します。

```
SELECT YEAR( '2001-09-12' );
```

YEARS 関数 [日付と時刻]

2 つの日付を指定すると、2 つの日付の間の年数を整数で返します。この目的で使用する場合は、「DATEDIFF 関数 [日付と時刻]」 318 ページ を代わりに使用することをおすすめします。

1 つの日付を指定すると、その年が返されます。この目的で使用する場合は、「DATEPART 関数 [日付と時刻]」 321 ページ を代わりに使用することをおすすめします。

1 つの日付と整数を指定すると、指定した日付に、指定した整数の年数が加算されます。この目的で使用する場合は、「DATEADD 関数 [日付と時刻]」 318 ページ を代わりに使用することをおすすめします。

構文 1

YEARS([*datetime-expression*,] *datetime-expression* **)**

構文 2

YEARS(*datetime-expression*, *integer-expression* **)**

パラメータ

datetime-expression　日付と時刻。

integer-expression　*datetime-expression* に加算する年数。*integer-expression* が負の場合、日時の値から適切な年数が引かれます。*integer-expression* を指定する場合は、*datetime-expression* を datetime データ型として明示的にキャストしてください。

データ型のキャストの詳細については、「CAST 関数 [データ型変換]」 308 ページを参照してください。

備考

YEARS の値を求めるには、2 つの日付の間に年の最初の日がいくつあるかを計算します。

構文 1 は、整数を返します。構文 2 は、タイムスタンプを返します。

標準と互換性

- **SQL/2003**　ベンダ拡張。

例

次の文はどちらも −4 を返します。

```
SELECT YEARS( '1998-07-13 06:07:12',
        '1994-03-13 08:07:13' );

SELECT DATEDIFF( year,
  '1998-07-13 06:07:12',
  '1994-03-13 08:07:13' );
```

次の文は、1998 を返します。

```
SELECT YEARS( '1998-07-13 06:07:12' )
SELECT DATEPART( year, '1998-07-13 06:07:12' );
```

次の文は、指定した日付の 300 年後を返します。


```
SELECT YEARS( CAST( '1998-07-13 06:07:12' AS TIMESTAMP ), 300 )
SELECT DATEADD( year, 300, '1998-07-13 06:07:12' );
```

YMD 関数 [日付と時刻]

指定した年、月、日に相当する日付値を返します。値は -32768 ー 32767 の small integer です。

構文

YMD(
integer-expression1,
integer-expression2,
integer-expression3 **)**

パラメータ

integer-expression1 年。

integer-expression2 月の数。月が 1 ー 12 の値でない場合は、年が相応に調整されます。

integer-expression3 日の数。任意の整数を指定でき、日付が相応に調整されます。

標準と互換性

- **SQL/2003** ベンダ拡張。

例

次の文は、値 1998-06-12 を返します。

```
SELECT YMD( 1998, 06, 12 );
```

値が通常の範囲から外れている場合は、日付が相応に調整されます。たとえば、次の文は値 2000-03-01 を返します。

```
SELECT YMD( 1999, 15, 1 );
```

第 14 章

Ultra Light SQL 文のリファレンス

目次

Ultra Light SQL 文の概要

この章で説明する SQL 文は Ultra Light SQL でサポートされています。これらの SQL 文は、SQL Anywhere データベースでサポートされている文のサブセットです。

始める前に

- Ultra Light のテーブルでは、所有者の概念がサポートされていません。既存の SQL と、プログラムで生成される SQL のために、Ultra Light では *owner.table-name* の構文が許可されます。ただし、テーブル所有者は Ultra Light ではサポートされていないので、所有者はチェックされません。
- Ultra Light SQL 文の構文の表記規則は、SQL Anywhere の文と同じです。この表記規則について、またこの表記規則を使用して SQL 構文が表されている方法を理解している必要があります。「SQL 構文の表記規則」 『SQL Anywhere サーバ - SQL リファレンス』を参照してください。
- Ultra Light SQL を使用するとトランザクションが作成されます。トランザクションは、最後の ROLLBACK または COMMIT 後のすべての変更内容 (INSERT、UPDATE、DELETE) から構成されます。「Ultra Light でのトランザクション処理と独立性レベル」 16 ページを参照してください。

  変更内容は、COMMIT を実行することによって確定できます。ROLLBACK 文を実行すると、変更内容が削除されます。「Ultra Light COMMIT 文」 386 ページと「Ultra Light ROLLBACK 文」 400 ページを参照してください。

Ultra Light の文のカテゴリ

SQL 文は、文の最初の語で分類されています。この語はほとんど常に動詞です。したがって、言語の性質が、データベースに対する一連の命令文 (コマンド) になります。Ultra Light でサポートされている SQL 文は次のように分類できます。

- **データ検索文**　これらの文はクエリとも呼ばれ、テーブルからデータ式のローを選択できます。データ検索は SELECT 文を使用して行います。「Ultra Light SELECT 文」 401 ページを参照してください。
- **データ操作文**　データベースの内容を変更できます。データ操作は次の文を使用して行います。
  - 「Ultra Light DELETE 文」 395 ページ
  - 「Ultra Light INSERT 文」 399 ページ
  - 「Ultra Light UPDATE 文」 412 ページ
- **データ定義文**　データベースの構造またはスキーマを定義できます。スキーマの変更には次の文を使用します。
  - 「Ultra Light CREATE INDEX 文」 387 ページ
  - 「Ultra Light CREATE TABLE 文」 390 ページ

- 「Ultra Light DROP INDEX 文」 396 ページ
- 「Ultra Light DROP TABLE 文」 398 ページ
- 「Ultra Light ALTER TABLE 文」 380 ページ
- 「Ultra Light TRUNCATE TABLE 文」 409 ページ

- **トランザクション制御文**　Ultra Light アプリケーション内でトランザクションを制御できます。トランザクション制御には次の文を使用します。

  - 「Ultra Light CHECKPOINT 文」 385 ページ
  - 「Ultra Light COMMIT 文」 386 ページ
  - 「Ultra Light ROLLBACK 文」 400 ページ

- **同期管理**　Mobile Link サーバとの同期を一時的に制御できます。同期管理には次の文を使用します。

  - 「Ultra Light START SYNCHRONIZATION DELETE 文」 407 ページ
  - 「Ultra Light STOP SYNCHRONIZATION DELETE 文」 408 ページ
  - 「Ultra Light CREATE PUBLICATION 文」 389 ページ
  - 「Ultra Light ALTER PUBLICATION 文」 384 ページ
  - 「Ultra Light DROP PUBLICATION 文」 397 ページ

参照

- 「Ultra Light の式」 274 ページ
- 「Ultra Light の演算子」 287 ページ

Ultra Light ALTER TABLE 文

テーブル定義を変更します。

構文

ALTER TABLE *table-name*
{ *add-clause* | *modify-clause* | *drop-clause* |
rename-clause }

add-clause :
ADD { *column-definition* | *table-constraint* }

modify-clause :
ALTER *column-definition*

drop-clause :
DROP { *column-name* | **CONSTRAINT** *constraint-name* }

rename-clause :
RENAME { *new-table-name* | *old-column-name* **TO** *new-column-name*
| **CONSTRAINT** *old-constraint-name* **TO** *new-constraint-name* }

column-definition :
column-name data-type
[[**NOT**] **NULL**]
[**DEFAULT** *column-default*]
[*column-constraint* ⋯]

column-constraint :
[**UNIQUE**]

column-default :
{ **GLOBAL AUTOINCREMENT** [(*number*)] |
AUTOINCREMENT | **CURRENT DATE** |
CURRENT TIME | **CURRENT TIMESTAMP** |
NULL |
NEWID() |
constant-value }

table-constraint :
[**CONSTRAINT** *constraint-name*]
{ *foreign-key-constraint*|*unique-key-constraint* }
[**WITH MAX HASH SIZE** *value*]

foreign-key-constraint :
[**NOT NULL**] **FOREIGN KEY** [*role-name*] **(** *ordered-column-list* **)**
REFERENCES *table-name* [**(** *column-name*, ⋯ **)**]
[**CHECK ON COMMIT**]

unique-key-constraint :
UNIQUE(*ordered-column-list* **)**

ordered-column-list :
[**(** *column-name* [**ASC** | **DESC**], ⋯, **)**]

パラメータ

add-clause　テーブルに新しいカラム制約またはテーブル制約を追加します。

- **ADD *column-definition***　テーブルに新しいカラムを追加します。カラムにデフォルト値がある場合は、新しいカラムのすべてのローにそのデフォルト値が移植されます。
- **ADD *table-constraint***　テーブルに制約を追加します。オプションの制約名を使用すると、テーブル全体の制約を修正することなく、後で個々の制約を修正したり削除したりできます。

> **注意**
> Ultra Light では、プライマリ・キーを追加することはできません。

column-definition と *table-constraint* の詳細については、「Ultra Light CREATE TABLE 文」 390 ページを参照してください。

modify-clause　1 つのカラム制約を変更します。

column-definition と *table-constraint* の詳細については、「Ultra Light CREATE TABLE 文」 390 ページを参照してください。*column-definition* が ALTER 文の一部の場合には、プライマリ・キーは使用できません。

drop-clause　カラム制約またはテーブル制約を削除します。

- **DROP *column-name***　テーブルからカラムを削除します。カラムがインデックス、一意性制約、外部キー、またはプライマリ・キーに含まれている場合は、インデックス、制約またはキーを削除してからカラムを削除してください。
- **DROP CONSTRAINT *table-constraint***　テーブル定義から指定された制約を削除します。

> **注意**
> Ultra Light では、プライマリ・キーを削除することはできません。

table-constraint の詳細については、「Ultra Light CREATE TABLE 文」 390 ページを参照してください。

rename-clause　テーブル名、カラム名、制約名を変更します。

- **RENAME *new-table-name***　テーブルの名前を *new-table-name* に変更します。古いテーブル名を使用しているアプリケーションを修正する必要があることに注意してください。古いテーブル名を自動的に割り当てられた外部キーは、名前を変更しません。
- **RENAME *old-column-name* TO *new-column-name***　カラムの名前を *new-column-name* に変更します。古いカラム名を使用しているアプリケーションを修正する必要があることに注意してください。

- **RENAME** *old-constraint-name* **TO** *new-constraint-name*　制約の名前を *new-constraint-name* に変更します。古い制約名を使用しているアプリケーションを修正する必要があることに注意してください。

> **注意**
> Ultra Light では、プライマリ・キーの名前を変更することはできません。

column-constraint　カラム制約は、データベース内のデータの整合性を保つために、カラムに格納できる値を制限します。カラム制約は UNIQUE である必要があります。

UNIQUE　テーブル内の各ローをユニークに識別する 1 つまたは複数のカラムを識別します。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。

備考

1 つの ALTER TABLE 文では、1 つの *table-constraint* または *column-constraint* だけを追加、変更、または削除できます。

役割名は外部キーの名前です。*role-name* の主な機能は、同じテーブルに対する 2 つの外部キーを区別することです。また、**CONSTRAINT** *constraint-name* を使用して外部キーに名前を付けることができます。ただし、両方のメソッドを使用して外部キーに名前を付けないでください。

カラムに一意性制約、外部キー、またはプライマリ・キーが設定されている場合は、制約またはキーを削除してからカラムを修正してください。テーブルまたはカラムの制約は MODIFY (変更) できません。制約を変更するには、古い制約を DELETE (削除) し、新しい制約を ADD (追加) します。

名前の末尾が **nosync** のテーブルは、末尾が **nosync** のテーブル名に変更する必要があります。「Ultra Light の nosync テーブル」 『Mobile Link - クライアント管理』を参照してください。

テーブルを対象とした文が、別の要求やクエリですでに参照されている場合、ALTER TABLE は実行できません。同様に、テーブルの変更中は、そのテーブルを参照する要求は処理されません。また、データベースにアクティブなクエリやコミットされていないトランザクションがある場合は ALTER TABLE を実行できません。

Ultra Light.NET ユーザの場合、すべてのデータ・オブジェクト （たとえば ULDataReader） に対して Dispose メソッドも呼び出さないと、この文を実行できません。「Dispose メソッド」 『Ultra Light - .NET プログラミング』を参照してください。

スキーマの変更が同時に開始された場合、文は解放されません。「DDL 文を使用したスキーマ変更」 11 ページを参照してください。

参照

- 「Ultra Light CREATE TABLE 文」 390 ページ
- 「Ultra Light DROP TABLE 文」 398 ページ
- 「Ultra Light のデータ型」 262 ページ
- 「テーブルの変更」 『SQL Anywhere サーバ - SQL の使用法』
- 「テーブル制約とカラム制約の使い方」 『SQL Anywhere サーバ - SQL の使用法』
- 「オートインクリメント・カラムの分割サイズの上書き」 『Mobile Link - クライアント管理』

- 「最後に割り当てられた GLOBAL AUTOINCREMENT の値の割り出し」 『Mobile Link - クライアント管理』

例

次の例は、employee テーブルから office カラムを削除します。

```
ALTER TABLE employee
DROP office
```

次の例は、テーブルに最大 50 文字を格納できるようにします。

```
ALTER TABLE customer
MODIFY address CHAR(50)
```

Ultra Light ALTER PUBLICATION 文

パブリケーションを変更します。パブリケーションは、Ultra Light リモート・データベース内の同期されたデータを示します。

構文

ALTER PUBLICATION *publication-name alterpub-clause*

alterpub-clause :
  ADD TABLE *table-name* [**WHERE** *search-condition*]
| **ALTER TABLE** *table-name* [**WHERE** *search-condition*]
| { **DELETE** | **DROP** } *table-name*
| **RENAME** *publication-name*

パラメータ

WHERE 句 この文で WHERE 句を変更した場合、同期時に、*search-condition* を満たすローだけが、関連付けられたテーブルからのアップロード対象になります。「Ultra Light の探索条件」 280 ページを参照してください。

備考

WHERE 句を指定しなかった場合は、最後の同期後にテーブル内で変更されたすべてのローがアップロード対象になります。

関連する動作

オートコミット

参照

- 「Ultra Light での同期の設計」 『Mobile Link - クライアント管理』
- 「Ultra Light CREATE PUBLICATION 文」 389 ページ
- 「Ultra Light DROP PUBLICATION 文」 397 ページ
- 「Ultra Light START SYNCHRONIZATION DELETE 文」 407 ページ
- 「Ultra Light STOP SYNCHRONIZATION DELETE 文」 408 ページ

例

次の文は、customers テーブルを pub_contact パブリケーションに追加します。

```
ALTER PUBLICATION pub_contact
  ADD TABLE customers
```

Ultra Light CHECKPOINT 文

この文は、データベースにチェックポイントを実行させるために使用します。

構文

CHECKPOINT

備考

CHECKPOINT 文をコミット・フラッシュのトリガとして使用できます。コミット・フラッシュが実行されると、コミットされていないトランザクションが記憶領域に書き込まれます。

この文の代わりに、Embedded SQL API を使用している場合には、ULCheckpoint メソッドを使用できます。または、C++ コンポーネント・アプリケーションを作成している場合には、接続オブジェクトで Checkpoint メソッドを使用できます。その他すべての API では、この文を使用する必要があります。

関連する動作

この文により、保留中のコミットされたトランザクションが記憶領域にすべてフラッシュされますが、現在のトランザクションはコミットもフラッシュもされません。

参照

- 「単一のトランザクションまたはグループ化されたトランザクションのフラッシュ」 47 ページ
- 「Ultra Light COMMIT_FLUSH 接続パラメータ」 172 ページ
- Ultra Light for Embedded SQL：「ULCheckpoint 関数」『Ultra Light - C/C++ プログラミング』
- Ultra Light for C/C++：「Checkpoint 関数」『Ultra Light - C/C++ プログラミング』

Ultra Light COMMIT 文

データベースへの変更を絵確定します。

構文

COMMIT [**WORK**]

パラメータ

WORK はオプションのキーワードです。

備考

トランザクションは、COMMIT または ROLLBACK 文の間にデータベースへの 1 回の接続について行われる挿入、更新、削除です。COMMIT 文は、トランザクションを終了し、このトランザクション中で行われたすべての変更内容をデータベースに継続保管します。

ALTER、CREATE、DROP の各文は、自動的にコミットされます。

参照

- 「Ultra Light ROLLBACK 文」 400 ページ

Ultra Light CREATE INDEX 文

指定したテーブルにインデックスを作成します。「Ultra Light クエリ・パフォーマンスの最適化」 38 ページを参照してください。

構文

CREATE [**UNIQUE**] **INDEX** [*index-name*]
ON *table-name* **(** *ordered-column-list* **)**
WITH MAX HASH SIZE *x*

ordered-column-list :
[**(** *column-name* [**ASC** | **DESC**], ⋯,] **)**]

パラメータ

UNIQUE　UNIQUE パラメータによって、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。

テーブル上の一意性制約とユニーク・インデックスの間には違いがあります。ユニーク・インデックスのカラムは NULL を使用できますが、一意性制約のカラムには使用できません。外部キーは、複数の NULL のインスタンスを内包できるため、一意性制約があるプライマリ・キーまたはカラムを参照できます。

ordered-column-list　カラムの順序リストです。順序リストは、昇順または降順にソートできます。

WITH MAX HASH SIZE　デフォルトのインデックス・ハッシュ・サイズをバイト単位で設定します。この値を設定しなかった場合は、デフォルト・サイズの 4 バイトが使用されます。「Ultra Light max_hash_size プロパティ」 145 ページを参照してください。

備考

インデックスによって、Ultra Light が特定のローを検索しやすくし、クエリのパフォーマンスを向上させることができます。ただし、インデックスを管理する必要があるので、INSERT、DELETE、UPDATE の各文や同期は低速になる可能性があります。

インデックスは、データベースに対して発行されたクエリのパフォーマンスを向上させたり、ORDER BY 句を使用してクエリをソートするときに自動的に使用されます。インデックスはいったん作成されると、DROP INDEX を使用して削除するときを除いて、SQL 文では二度と参照されません。

インデックスはデータベース内の領域を占有します。また、インデックスの管理に必要な追加作業は、データ修正操作のパフォーマンスに影響する場合があります。このため、クエリのパフォーマンスが向上しないインデックスの作成は避けてください。

CREATE INDEX 文の処理中は、そのインデックスを参照する要求やクエリは処理されません。また、データベースにアクティブなクエリやコミットされていないトランザクションがある場合は CREATE INDEX を実行できません。

Ultra Light では、クエリ・アクセス・プランを使用してクエリを最適化することもできます。「Ultra Light のクエリ・アクセス・プラン」 291 ページを参照してください。

Ultra Light.NET ユーザの場合、すべてのデータ・オブジェクト （たとえば ULDataReader） に対して Dispose メソッドも呼び出さないと、この文を実行できません。「Dispose メソッド」 『Ultra Light - .NET プログラミング』を参照してください。

Ultra Light では、プライマリ・キーと一意性制約のインデックスは自動的に作成されます。

スキーマの変更が同時に開始された場合、文は解放されません。「DDL 文を使用したスキーマ変更」 11 ページを参照してください。

関連する動作

- オートコミット

参照

- 「Ultra Light DROP INDEX 文」 396 ページ
- 「Ultra Light のインデックスの操作」 100 ページ

例

次の例は、employee テーブルで 2 カラムのインデックスを作成します。

```
CREATE INDEX employee_name_index
ON employee
( emp_lname, emp_fname )
```

次の例は、prod_id カラム用に sales_order_items テーブル上でインデックスを作成します。

```
CREATE INDEX item_prod
ON sales_order_items
( prod_id )
```


Ultra Light CREATE PUBLICATION 文

パブリケーションを作成します。

構文

CREATE PUBLICATION *publication-name*
(**TABLE** *table-name* [**WHERE** *search-condition*], ...)

パラメータ

WHERE 句 WHERE 句を指定した場合、同期時に、*search-condition* を満たすローだけが、関連付けられたテーブルからのアップロード対象になります。「Ultra Light の探索条件」 280 ページ を参照してください。

WHERE 句を指定しなかった場合は、最後の同期後にテーブル内で変更されたすべてのローがアップロード対象になります。

備考

パブリケーションは、単一の同期操作で同期されるテーブルを設定します。パブリケーションによって、Mobile Link サーバにアップロードされるデータが決定します。Mobile Link サーバのダウンロード・セッション中には、これらのテーブルだけのローが送り返される可能性があります。ダウンロードされるローは、テーブルの WHERE 句を満たす必要はありません。

テーブル全体だけをパブリッシュできます。Ultra Light では、テーブルの特定のカラムはパブリッシュできません。

関連する動作

- オートコミット

参照

- 「Ultra Light クライアント」『Mobile Link - クライアント管理』
- 「Ultra Light DROP PUBLICATION 文」 397 ページ
- 「Ultra Light ALTER PUBLICATION 文」 384 ページ
- 「Ultra Light の探索条件」 280 ページ

例

次の文は、2 つのテーブルのすべてのカラムとローをパブリッシュします。

```
CREATE PUBLICATION pub_contact (
  TABLE contact,
  TABLE company
)
```

次の文は、customer テーブルの status カラムをテストする WHERE 句を組み込んで、アクティブな customer ローのみをパブリッシュします。

```
CREATE PUBLICATION pub_customer (
  TABLE customer
  WHERE status = 'active'
)
```


Ultra Light CREATE TABLE 文

データベースに新しいテーブルを作成します。

構文

CREATE TABLE *table-name*
({ *column-definition* | *table-constraint* }, ⋯ **)**

column-definition :
column-name data-type [[**NOT**]
NULL] [**DEFAULT** *column-default*] [*column-constraint* ⋯]

column-constraint :
[**UNIQUE** | **PRIMARY KEY**]

table-constraint :
[**CONSTRAINT** *constraint-name*] *spec*

column-default :
{**GLOBAL AUTOINCREMENT** [(*number*)] |
AUTOINCREMENT | **CURRENT DATE** |
CURRENT TIME | **CURRENT TIMESTAMP** |
NULL |
NEWID( ) |
constant-value}

spec :
{ **PRIMARY KEY** (*ordered-column-list* |
[**NOT NULL**] **FOREIGN KEY** [*role-name*](*ordered-column-list*)
REFERENCES *table-name* (*column-name*, . . .)
[**CHECK ON COMMIT**] |
UNIQUE (*ordered-column-list*) } [**WITH MAX HASH SIZE** *number*]

ordered-column-list :
[**(** *column-name* [**ASC** | **DESC**] ⋯,] **)**]

パラメータ

column-definition テーブル内のカラムを定義します。この句には、次のパラメータを使用できます。

- ***column-name*** カラム名は識別子です。同じテーブル内の 2 つのカラムが同じ名前を持つことはできません。「Ultra Light の識別子」 253 ページを参照してください。
- ***data-type*** カラムのデータ型です。「Ultra Light のデータ型」 262 ページを参照してください。
- **[NOT] NULL** NOT NULL が指定されているか、カラムが UNIQUE または PRIMARY KEY 制約下にある場合、カラムはいずれのローでも NULL を持つことはできません。それ以外の場合は、NULL は許可されます。

- ***column-default* と *column-constraint*** 式の作成に使用される複数のパラメータを含む句です。「Ultra Light の式」 274 ページを参照してください。

column-constraint 同等のテーブル制約の短い形式で、現在のローだけがあります。カラム制約は次のいずれかです。

- **UNIQUE** テーブル内の各ローをユニークに識別する 1 つまたは複数のカラムを識別します。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。NULL 値は使用できません。
- **PRIMARY KEY** これは、1 つのカラムが 1 つのプライマリ・キー制約のみを持つことができるという点を除いて、一意性制約と同じです。プライマリ・キーは、通常、ローの識別子に最も適しています。たとえば、顧客番号は顧客テーブルのプライマリ・キーです。

  プライマリ・キーに含まれるカラムは、自動的に NOT NULL になります。

table-constraint テーブル制約は、データベース内のデータの整合性を保つために、テーブル内の 1 つまたは複数のカラムに格納できる値を制限します。整合性制約の違反を起こす文は、実行が完了しません。このような文がエラー検出の前に行った変更はロールバックされ、エラーがレポートされます。

column-default DEFAULT 値を指定する場合、カラムの値を指定しない INSERT 文のカラムの値としてこのデフォルト値が使用されます。INSERT 文はカラムの値を指定しません。DEFAULT を指定しない場合、DEFAULT NULL と等しくなります。デフォルトのオプションを以下に示します。

- **AUTOINCREMENT** 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。テーブルに挿入する場合、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。

> **ヒント**
> Ultra Light 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。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。

- **GLOBAL AUTOINCREMENT** ドメインが分割されるという点を除いて、AUTOINCREMENT と同じです。「Ultra Light での GLOBAL AUTOINCREMENT の使用」『Mobile Link - クライアント管理』を参照してください。

  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 番号を割り当てます。「Ultra Light global_database_id オプション」 162 ページを参照してください。

  Ultra Light では、データベースのデフォルト値は、そのデータベース番号でユニークに識別された分割からのみ設定されます。

- **NULL**　カラムに NULL を格納できます。
- **NEWID()**　ユニークな識別子の値を生成する関数です。「NEWID 関数 [その他]」 347 ページを参照してください。
- **CURRENT TIMESTAMP**　CURRENT DATE と CURRENT TIME を結合して、TIMESTAMP 値を形成します。年、月、日、時、分、秒、秒の小数位で構成されます。秒は小数第 3 位まで格納されます。精度はシステム・クロックの精度によって制限されます。「CURRENT TIMESTAMP 特別値」 259 ページを参照してください。
- **CURRENT DATE**　CURRENT DATE は、現在の年、月、日を返します。「CURRENT DATE 特別値」 258 ページを参照してください。
- **CURRENT TIME**　現在の時、分、秒 (小数位あり) で構成される時刻を返します。「CURRENT TIME 特別値」 258 ページを参照してください。
- ***constant-value***　カラムのデータ型の定数です。通常、この定数は数値または文字列です。

spec　追加仕様を指定します。*spec* には次のいずれかを指定してください。

- **プライマリ・キー**　プライマリ・キーは、通常はテーブル内のローをすぐに識別するためのカラムのコレクションを識別します。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。
- **外部キー**　1 組のカラムの値を別のテーブル (プライマリ・テーブル) のプライマリ・キーまたは (あまり一般的ではありませんが) 一意性制約と一致するものだけに制限します。
- **role-name**　役割名は外部キーの名前です。*role-name* の主な機能は、同じテーブルに対する 2 つの外部キーを区別することです。また、**CONSTRAINT** *constraint-name* を使用して外部キーに名前を付けることができます。ただし、両方のメソッドを使用して外部キーに名前を付けないでください。
- **NOT NULL**　外部キー・カラムを NULL 入力不可にします。外部キー内の NULL は、外部テーブルのこのローに該当するローがプライマリ・テーブル中にないことを意味します。

  複数カラムの外部キー内の少なくとも 1 つの値が NULL である場合、キーの他のカラムに保持できる値に関する制限はありません。
- **CHECK ON COMMIT**　データベース・サーバで、COMMIT を待ってから、外部キーが使用されていることが確認されます。

  このため、データベースの変更は任意の順序で適用できます。指定がなかった場合、プライマリ・キー (または UNIQUE 制約の値) がデータベースになければ、これらの外部キーの値を持つローは追加できません。
- **UNIQUE**　テーブル内の各ローをユニークに識別する 1 つまたは複数のカラムを識別します。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。
- **ordered-column-list**　カラムの順序リストです。順序リストは、昇順または降順にソートできます。

- **WITH MAX HASH SIZE**　デフォルトのインデックス・ハッシュ・サイズをバイト単位で設定します。この値を設定しなかった場合、デフォルト・サイズの 8 バイトがインデックスのハッシュに使用されます。「Ultra Light max_hash_size プロパティ」 145 ページを参照してください。

備考

通常、カラム制約を使うのは、制約がテーブル内で複数の他カラムを参照しない場合です。複数のカラムを参照する場合は、テーブル制約を使用します。整合性制約の違反を起こす文は、実行が完了しません。このような文がエラー検出の前に行った変更は取り消され、エラーがレポートされます。

テーブル内の各ローは、ユニークなプライマリ・キー値を持ちます。

役割名を指定しない場合、役割名は以下のように設定されます。

1. テーブル名と同じ役割名を含む外部キーが存在しない場合、テーブル名が役割名として割り当てられます。
2. テーブル名がすでに使用されている場合、役割名は、0 が埋め込まれた、テーブルに固有の 3 桁の数字と結合されたテーブル名になります。

スキーマの変更　スキーマの変更が同時に開始された場合、文は解放されません。「DDL 文を使用したスキーマ変更」 11 ページを参照してください。

CREATE TABLE 文の処理中は、そのテーブルを参照する要求やクエリは処理されません。また、データベースにアクティブなクエリやコミットされていないトランザクションがある場合は CREATE TABLE を実行できません。

Ultra Light.NET ユーザの場合、すべてのデータ・オブジェクト （たとえば ULDataReader） に対して Dispose メソッドも呼び出さないと、この文を実行できません。「Dispose メソッド」 『Ultra Light - .NET プログラミング』を参照してください。

関連する動作

オートコミット

参照

- 「Ultra Light DROP TABLE 文」 398 ページ
- 「CREATE TABLE 文」 『SQL Anywhere サーバ - SQL リファレンス』
- 「Ultra Light のデータ型」 262 ページ
- 「オートインクリメント・カラムの分割サイズの上書き」 『Mobile Link - クライアント管理』

例

次の例は、図書データベース用にテーブルを作成し、図書情報を保持します。

```
CREATE TABLE library_books (
  isbn CHAR(20)      PRIMARY KEY,
  copyright_date     DATE,
  title              CHAR(100),
  author             CHAR(50),
  location           CHAR(50),
```



```
  FOREIGN KEY location REFERENCES room
)
```

次の例では、図書データベース用にテーブルを作成して、貸し出し図書の情報を保持します。date_borrowed のデフォルト値は、エントリが作成される日に本が貸し出されることを示します。date_returned カラムは、本が返却されるまでは NULL です。

```
CREATE TABLE borrowed_book (
  loaner_name CHAR(100)     PRIMARY KEY,
  date_borrowed DATE NOT NULL DEFAULT CURRENT DATE,
  date_returned       DATE,
  book              CHAR(20)
  FOREIGN KEY book REFERENCES library_books (isbn),
)
```

次の例は、販売データベース用にテーブルを作成し、注文と注文項目情報を保持します。

```
CREATE TABLE Orders (
  order_num INTEGER NOT NULL PRIMARY KEY,
  date_ordered DATE,
  name CHAR(80)
);
CREATE TABLE Order_item (
  order_num       INTEGER NOT NULL,
  item_num        SMALLINT NOT NULL,
  PRIMARY KEY (order_num, item_num),
  FOREIGN KEY (order_num)
  REFERENCES Orders (order_num)
)
```

Ultra Light DELETE 文

データベースのテーブルからローを削除します。

構文

DELETE
[**FROM**] *table-name*
[**WHERE** *search-condition*]

パラメータ

WHERE 句　WHERE 句を指定すると、*search-condition* を満たすローだけが削除されます。「Ultra Light の探索条件」 280 ページを参照してください。

また、WHERE 句では、非確定の関数 (RAND など) や変数を使用できません。この句では、カラムも制限されません。カラムをサブクエリで使用する場合、別のテーブルを参照する必要がある場合があります。

備考

DELETE 文は、指定したテーブルから、探索条件を満たすすべてのローを削除します。

Ultra Light では、独自の方法でローのステータスがトレースされます。削除やローのステータスの意味を理解している必要があります。「Ultra Light でのローのステータス」 14 ページを参照してください。

参照

- 「Ultra Light START SYNCHRONIZATION DELETE 文」 407 ページ
- 「Ultra Light STOP SYNCHRONIZATION DELETE 文」 408 ページ

例

次の文は、データベースから従業員 105 を削除します。

```
DELETE
FROM employee
WHERE emp_id = 105
```

次の文は、fin_data テーブルから、2000 年より前のデータをすべて削除します。

```
DELETE
FROM fin_data
WHERE year < 2000
```


Ultra Light DROP INDEX 文

インデックス定義をデータベースから永続的に削除します。

構文

DROP INDEX [*table-name.*]*index-name*

備考

プライマリ・インデックスは削除できません。

DROP INDEX 文の処理中は、そのインデックスを参照する要求やクエリは処理されません。また、データベースにアクティブなクエリやコミットされていないトランザクションがある場合は DROP INDEX を実行できません。

Ultra Light.NET ユーザの場合、すべてのデータ・オブジェクト（たとえば ULDataReader）に対して Dispose メソッドも呼び出さないと、この文を実行できません。「Dispose メソッド」『Ultra Light - .NET プログラミング』を参照してください。

スキーマの変更が同時に開始された場合、文は解放されません。「DDL 文を使用したスキーマ変更」 11 ページを参照してください。

参照

- 「Ultra Light CREATE INDEX 文」 387 ページ
- 「Ultra Light のインデックスの操作」 100 ページ

Ultra Light DROP PUBLICATION 文

パブリケーションを削除します。

構文

DROP PUBLICATION *publication-name1*, ..., *publication-nameX*

備考

Mobile Link サーバにアップロードするデータを示すパブリケーションが古くなった場合に使用します。

参照

- 「Ultra Light での同期の設計」『Mobile Link - クライアント管理』
- 「Ultra Light ALTER PUBLICATION 文」 384 ページ
- 「Ultra Light CREATE PUBLICATION 文」 389 ページ

例

次の文は、pub_contact パブリケーションを削除します。

```
DROP PUBLICATION pub_contact
```

Ultra Light DROP TABLE 文

テーブル定義とテーブル内のすべてのデータをデータベースから永続的に削除します。

構文

DROP TABLE *table-name*

備考

DROP TABLE 文は、指定されたテーブルの定義を削除します。テーブル内のすべてのデータは、削除プロセスの一部として自動的に削除されます。また、テーブルのすべてのインデックスとキーは DROP TABLE 文によって削除されます。

DROP TABLE 文の処理中は、そのテーブルやテーブルのインデックスを参照する要求やクエリは処理されません。また、データベースにアクティブなクエリやコミットされていないトランザクションがある場合は DROP TABLE を実行できません。

Ultra Light.NET ユーザの場合、すべてのデータ・オブジェクト（たとえば ULDataReader）に対して Dispose メソッドも呼び出さないと、この文を実行できません。「Dispose メソッド」『Ultra Light - .NET プログラミング』を参照してください。

スキーマの変更が同時に開始された場合、文は解放されません。「DDL 文を使用したスキーマ変更」 11 ページを参照してください。

参照

- 「Ultra Light ALTER TABLE 文」 380 ページ
- 「Ultra Light CREATE TABLE 文」 390 ページ

Ultra Light INSERT 文

テーブルに 1 つのローを挿入するか、クエリの結果セットからローを挿入します。

構文

INSERT [**INTO**]
table-name [**(** *column-name*, ⋯ **)**]
{ **VALUES** (*expression, ...*) | **SELECT** *statement* }

備考

オプションのカラム名のリストを指定すると、指定したカラムの中に値が 1 つずつ挿入されます。カラム名のリストを指定しないと、作成順 (SELECT * を使って取り出すときと同じ順序) に値がテーブル・カラムの中に挿入されます。ローは、テーブル内の任意の位置に挿入されます。

VALUES 式または SELECT 文のいずれかを使用する必要があります。両方を使用することはできません。

テーブルに挿入した文字列は、データベースが大文字と小文字を区別するかどうかに関係なく、常に入力された大文字と小文字の状態のままで格納されます。

参照

- 「Ultra Light SELECT 文」 401 ページ

例

次の文は、データベースに Eastern Sales 部を追加します。

```
INSERT
INTO department ( dept_id, dept_name )
VALUES ( 230, 'Eastern Sales' )
```


Ultra Light ROLLBACK 文

トランザクションを終了し、最後に行った COMMIT または ROLLBACK 以降の変更を元に戻します。

構文

ROLLBACK [WORK]

パラメータ

WORK はオプションのキーワードです。

備考

トランザクションは、COMMIT または ROLLBACK 文の間にデータベースへの 1 回の接続について行われる挿入、更新、削除です。ROLLBACK 文は、現在のトランザクションを終了し、前回の COMMIT または ROLLBACK 以降にデータベースに対して行われたすべての変更を元に戻します。

参照

- 「Ultra Light COMMIT 文」 386 ページ

Ultra Light SELECT 文

データベースから情報を取得します。

構文

SELECT [**DISTINCT**] [**FIRST** | **TOP** *n* [**START AT** *m*]]
select-list
[**FROM** *table-expression*, ⋯ *table-expression*]
[**WHERE** *search-condition*]
[**GROUP BY** *group-by-expression*,...*group-by-expression*]
[**HAVING** *search-condition*]
[**ORDER BY** *order-by-expression*,...*order-by-expression*]
[**FOR** { **UPDATE** | **READ ONLY** }]
[**OPTION (FORCE ORDER)**]

select-list :
{ *column-name* | *column-expression* } [**AS**]
alias-name

order-by-expression :
{ *integer* | *expression* } [**ASC** | **DESC**]

パラメータ

select-list　カンマで区切られた式のリストであり、データベースから何を取り出すかを指定します。アスタリスク (*) を使用すると、FROM 句に記述された全テーブルのすべてのカラムを選択できます。

select-list の式に続いて、この式を表すエイリアス名を定義できます。これによって、エイリアス名をクエリ内の別の位置 (WHERE 句や ORDER BY 句など) で使用できるようになります。

DISTINCT 句　DISTINCT を指定しない場合は、SELECT 文の句を満たすすべてのローを返します。DISTINCT を指定すると、重複した出力ローが削除されます。多くの場合、DISTINCT を指定すると、文の実行に時間が非常に長くかかります。したがって、DISTINCT を使用するのは、必要な場合だけにしてください。

FIRST、TOP、または START AT 句　結果セットの最初のローまたは結果セットの最初の *n* 個のローだけを明示的に取得できます。TOP 句と START AT 句を指定すると、特定の方法で結果セットをパラメータ化することにより、結果セットを明示的に制限するクエリの中でさらに柔軟性を高めることができます。パラメータ化では、TOP 句は返されるロー数を指定し、START AT 句は開始ポイントの結果セットを設定します。

FROM 句　*table-expression* の中で指定されるテーブルとビューからローを取り出します。「FROM 句」 403 ページを参照してください。

WHERE 句　WHERE 句を指定すると、*search-condition* を満たすローだけが選択されます。「Ultra Light の探索条件」 280 ページを参照してください。

GROUP BY 句　クエリの結果には、GROUP BY 式の中の個別の値の各セットに対し 1 つのローが入ります。テーブル・リストのローの各グループに対する結果にはローが 1 つずつ含まれるため、結果ローはグループとして頻繁に参照されます。これらのグループ内のローには、集合関数を適用できます。NULL があった場合、ユニークな値と見なされます。

HAVING 句　この句は、個々のロー値ではなくグループ値に基づいてローを選択します。HAVING 句を使用できるのは、文に GROUP BY 句があるか、select-list が集合関数のみから成る場合だけです。探索条件は集約式である必要があります。集約式と、GROUP BY 句内にある式だけを指定できます。HAVING 条件は、候補のローが完全にグループ化されてからテストされます。

ORDER BY 句　この句で指定した式に従って、クエリの結果がソートされます。ORDER BY リストの各項目には、昇順の場合 (デフォルト) は ASC、降順の場合は DESC のラベルを付けることができます。式が整数 *n* である場合、クエリの結果は select リストの *n* 番目の項目でソートされます。

特定の順序でローが返されるようにする唯一の方法は ORDER BY を使用することです。ORDER BY 句がない場合は、Ultra Light が最も効率のよい順序でローを返します。

FOR 句　この句には、クエリの動作を制御する 2 つの形式があります。

- **FOR READ ONLY**　この句は、クエリが更新に使用されないようにします。クエリが更新に使用されないことがわかっている場合、Ultra Light のパフォーマンスが向上することがあるので、この句はできるだけ使用してください。たとえば、この句が有効になっていると、Ultra Light でダイレクト・ロー・スキャンを使用する判断が可能になります。「ダイレクト・ページ・スキャンの使用」 45 ページを参照してください。
- **FOR UPDATE**　この句はデフォルトで適用されます。この句によってクエリを更新に使用できるようになります。

OPTION (FORCE ORDER) 句　この句は一般的には使用しないことをおすすめします。この句は、Ultra Light で選択されたテーブル・アクセス順序を上書きし、クエリに出現する順序でテーブルにアクセスするよう Ultra Light に要求します。この句は、クエリの順序が Ultra Light の順序よりも確実に効率的である場合にのみ使用してください。

Ultra Light では、クエリ・アクセス・プランを使用してクエリを最適化することもできます。「Ultra Light のクエリ・アクセス・プラン」 291 ページを参照してください。

備考

クエリは必ず閉じてください。そうしないと、メモリが解放されず、存在し続けるテンポラリ・テーブルの数が不必要に増加することになります。

参照

- 「Ultra Light クエリ・パフォーマンスの最適化 」 38 ページ
- 「SELECT 文」 『SQL Anywhere サーバ - SQL リファレンス』
- 「クエリ：テーブルからのデータの選択」 『SQL Anywhere サーバ - SQL の使用法』

例

次の文は、employee テーブルから従業員の数を選択します。

```
SELECT COUNT(*)
FROM employee
```

次の文は、employee テーブルで 40 番目から 49 番目までの 10 個のローを選択します。

```
SELECT TOP 10 START AT 40 * FROM employee
```

FROM 句

SELECT 文に必要なデータベース・テーブルまたはビューを指定します。

構文

FROM *table-expression*, ⋯

table-expression :
[*table-name*
[[**AS**] *correlation-name*]
| (*SELECT-expression*)
[**AS**] *derived-table-name* **(** *column-name*, ⋯ **)**
| (*table-expression*)
| *table-expression join-operator table-expression* [**ON** *search-condition*]

join-operator :
, | **CROSS JOIN** | **INNER JOIN** | **LEFT OUTER JOIN** | **JOIN**

パラメータ

table-name　ベース・テーブルまたはテンポラリ・テーブル。Ultra Light では、異なるユーザがテーブルを所有できません。ユーザ ID でテーブルを修飾した場合、ID は無視されます。

correlation-name　文の他の部分のオブジェクトを参照するときに使用する識別子。

同じ相関名をテーブル式の同じテーブルに対して 2 回使用する場合、そのテーブルは、一度だけリストされたものとして扱われます。たとえば、次の 2 つの SELECT 文は同じです。

```
SELECT *
FROM sales_order
CROSS JOIN sales_order_items,
sales_order
CROSS JOIN employee
```

```
SELECT *
FROM sales_order
CROSS JOIN sales_order_items
CROSS JOIN employee
```

これに対して、次の文では Person テーブルの 2 つのインスタンスとして扱われ、異なる相関名 HUSBAND と WIFE を使います。

```
SELECT *
FROM Person HUSBAND, Person WIFE
```

derived-table-name　抽出テーブルは、SELECT 文として指定するテーブル式です。カッコで囲んだ SELECT 文に続いて、抽出テーブルの名前と、カッコで囲んだ抽出カラム名のリストを指定します。SELECT 式ごとに抽出カラム名を抽出テーブルに指定する必要があります。

抽出テーブルの Select リストの項目は、抽出テーブル名 (オプション) に続いてピリオド (.) とカラム名を指定して参照します。カラム名があいまいではない場合は、カラム名だけを使用できます。

SELECT 文内で抽出テーブル名を参照することはできません。この参照は、「**内部参照**」とも呼ばれます。「式のサブクエリ」 278 ページを参照してください。

備考

FROM 句を指定しない場合、SELECT 文の式は定数式である必要があります。

> **抽出テーブル**
> この説明はテーブルについてのものですが、特に注意書きがなければ抽出テーブルにも適用します。

FROM 句は、指定した全テーブルのすべてのカラムで構成される結果セットを作成します。最初に、コンポーネント・テーブルのすべてのローの組み合わせが結果セットの中に入ります。次に、JOIN 条件か WHERE 条件、またはその両方の分だけ、通常は組み合わせの数が減ります。

ジョイン演算子は、共通のカラム名に基づいて 2 つのテーブルを連結します。Ultra Light では、次の演算子がサポートされています。

- ,
- **CROSS JOIN (クロス・ジョイン)**
- **INNER JOIN (内部ジョイン)**
- **LEFT OUTER JOIN (左外部ジョイン)**
- **JOIN (ジョイン)**

これらの演算子には、前述の条件を指定できます。ON 条件は、単一のジョイン演算に対して指定され、ジョインを使用して結果セットにローを作成する方法を示します。JOIN 演算子には常に ON 条件が必要です。

WHERE 句は、ジョインによって潜在的なローが作成された後に結果セット内のローを制限するために使用します。

カンマ・ジョインは CROSS JOIN と同じです。この演算子でも ON 句を使用できません。

INNER ジョインの場合、ON または WHERE を使用した制限は同じです。OUTER ジョインの場合、これらは同じではありません。

> **注意**
> Ultra Light では、KEY JOIN と NATURAL JOIN の各ジョインはサポートされていません。

参照

- 「ジョイン：複数テーブルからのデータ検索」 『SQL Anywhere サーバ - SQL の使用法』
- 「Ultra Light DELETE 文」 395 ページ
- 「Ultra Light SELECT 文」 401 ページ
- 「Ultra Light UPDATE 文」 412 ページ

例

次は、有効な FROM 句です。

```
...
FROM employee
...

...
FROM employee NATURAL JOIN department
...

...
FROM customer
CROSS JOIN sales_order
CROSS JOIN sales_order_items
CROSS JOIN product
...
```

次のクエリは、クエリ内での抽出テーブルの使い方を示します。

```
SELECT lname, fname, number_of_orders
FROM customer JOIN
    ( SELECT cust_id, COUNT(*)
      FROM sales_order
       GROUP BY cust_id )
    AS sales_order_counts( cust_id,
                        number_of_orders )
ON ( customer.id = sales_order_counts.cust_id )
WHERE number_of_orders > 3
```


Ultra Light SET OPTION 文

この文は、データベース・オプションの値を変更するために使用します。

構文

SET OPTION *option-name*=*option-value*

option-name: identifier

option-value: string, identifier, or number

備考

この文で設定できるのはデータベース・オプションのみです。データベースの作成後にプロパティを変更することはできません。

Ultra Light では、オプションが永続的かどうかの指定はできません。オプションが一時的か永続的かは、Ultra Light でのオプションの実装方法で決まります。永続的なオプションだけが、データベースの sysuldata システム・テーブルに格納されます。Ultra Light が一時的なオプションを使用するのは、データベースの実行が停止するまでです。

参照

- 「sysuldata システム・テーブル」 247 ページ
- 「Ultra Light の設定可能なオプション」 160 ページ
- 「DB_PROPERTY 関数 [システム]」 324 ページ

例

次の文を実行すると、global_database_id オプションが 100 に設定されます。

```
SET OPTION global_database_id=100
```

Ultra Light START SYNCHRONIZATION DELETE 文

Mobile Link 同期で行われた削除のロギングを再起動します。

構文

START SYNCHRONIZATION DELETE

備考

Ultra Light データベースでは、同期が必要なローへの変更が自動的に追跡されます。Ultra Light では、変更されたテーブルの次の同期時に、これらの変更が統合データベースにアップロードされます。この文を使用すると、削除されたローの追跡を一時的に再開できます。

STOP SYNCHRONIZATION DELETE 文を実行すると、その接続に対してそれ以降に実施された削除操作は同期されません。この効果は、START SYNCHRONIZATION DELETE 文が実行されるまで継続します。STOP SYNCHRONIZATION DELETE を繰り返してもそれ以上効果はありません。

STOP SYNCHRONIZATION DELETE 文の実行回数に関わらず、START SYNCHRONIZATION DELETE 文を 1 回実行すれば、ロギングが再起動します。

アプリケーションでデータを同期しない場合は、START SYNCHRONIZATION DELETE を使用しないでください。

Ultra Light では、独自の方法でローのステータスがトレースされます。削除やローのステータスの意味を理解している必要があります。「Ultra Light でのローのステータス」 14 ページを参照してください。

参照

- 「Ultra Light STOP SYNCHRONIZATION DELETE 文」 408 ページ

例

次の一連の SQL 文は、START SYNCHRONIZATION DELETE と STOP SYNCHRONIZATION DELETE の使用方法を示します。

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM PROPOSAL
WHERE last_modified < months( CURRENT TIMESTAMP, -1 )
START SYNCHRONIZATION DELETE;
COMMIT;
```


Ultra Light STOP SYNCHRONIZATION DELETE 文

Mobile Link 同期で行われた削除のロギングを一時的に停止します。

構文

STOP SYNCHRONIZATION DELETE

備考

Ultra Light データベースでは、同期が必要なローへの変更が自動的に追跡されます。これらの変更は次の同期時に統合データベースにアップロードされます。この文を使用すると、削除されたローの追跡を一時的に停止できます。

STOP SYNCHRONIZATION DELETE 文を実行すると、その接続に対してそれ以降に実施された削除操作は同期されません。この効果は、START SYNCHRONIZATION DELETE 文が実行されるまで継続します。

STOP SYNCHRONIZATION DELETE を繰り返してもそれ以上効果はありません。STOP SYNCHRONIZATION DELETE 文の実行回数に関わらず、START SYNCHRONIZATION DELETE 文を 1 回実行すれば、ロギングが再起動します。

このコマンドは、リモート・データベースに対して訂正を行うには便利ですが、Mobile Link 同期を事実上無効にしてしまうので、使用の際には注意してください。

アプリケーションでデータを同期しない場合は、STOP SYNCHRONIZATION DELETE を使用しないでください。

Ultra Light では、独自の方法でローのステータスがトレースされます。削除やローのステータスの意味を理解している必要があります。「Ultra Light でのローのステータス」 14 ページを参照してください。

参照

- 「Ultra Light START SYNCHRONIZATION DELETE 文」 407 ページ

Ultra Light TRUNCATE TABLE 文

テーブル定義を削除せずに、テーブルからすべてのローを削除します。

構文

TRUNCATE TABLE *table-name*

備考

TRUNCATE TABLE 文によってテーブル内のローがすべて削除され、この後の同期時に Mobile Link サーバにこの削除について通知されません。これは、次の文と同義です。

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM TABLE;
START SYNCHRONIZATION DELETE;
```

> **注意**
> この文は、同期またはレプリケーション対象のデータベースに対して慎重に使用する必要があります。Mobile Link サーバに通知されないので、この削除によって整合性が失われ、その結果、同期またはレプリケーションに失敗する可能性があります。

TRUNCATE TABLE 文の後、テーブル構造体とインデックスのすべては、DROP TABLE 文が発行されるまで存在し続けます。カラムの定義と制約はそのまま残ります。

テーブルを対象とした文が、別の要求やクエリですでに参照されている場合、TRUNCATE TABLE は実行できません。同様に、テーブルの変更中は、そのテーブルを参照する要求は処理されません。また、データベースにアクティブなクエリやコミットされていないトランザクションがある場合は TRUNCATE TABLE を実行できません。

Ultra Light.NET ユーザの場合、すべてのデータ・オブジェクト（たとえば ULDataReader）に対して Dispose メソッドも呼び出さないと、この文を実行できません。「Dispose メソッド」『Ultra Light - .NET プログラミング』を参照してください。

スキーマの変更　スキーマの変更が同時に開始された場合、文は解放されません。「DDL 文を使用したスキーマ変更」 11 ページを参照してください。

関連する動作

テーブルに DEFAULT AUTOINCREMENT または DEFAULT GLOBAL AUTOINCREMENT と定義されたカラムがある場合、TRUNCATE TABLE はそのカラムの次に使用可能な値をリセットします。

TRUNCATE TABLE でローが削除済みとマーク付けされると、この処理を実行したユーザはローにアクセスできなくなります。ただし、他の接続からはアクセスできます。ROLLBACK を使用すると、現在のユーザが再びアクセスできるようになります。COMMIT を使用するとローが物理的に削除され、すべての接続からデータにアクセスできなくなります。

トランケート対象のテーブルを同期すると、テーブルに適用されているすべての INSERT 文が、TRUNCATE TABLE 文より優先されます。

参照

- 「Ultra Light DELETE 文」 395 ページ
- 「Ultra Light START SYNCHRONIZATION DELETE 文」 407 ページ
- 「Ultra Light STOP SYNCHRONIZATION DELETE 文」 408 ページ

例

Departments テーブルからすべてのローを削除します。

```
TRUNCATE TABLE Departments
```

Ultra Light UNION 文

2 つ以上の SELECT 文の結果を結合するために使用します。

構文

select-statement-without-ordering
[**UNION** [**ALL** | **DISTINCT**] *select-statement-without-ordering*]...
[**ORDER BY** [*number* [**ASC** | **DESC**] , ...]

備考

いくつかの SELECT 文の結果を、UNION を使ってより大きな結果へと結合することができます。コンポーネントの SELECT 文には、それぞれの select リストに同じ数の項目を指定します。各文には ORDER BY 句を含めることはできません。

UNION ALL の結果は、単にコンポーネントの SELECT 文の結果を結合したものです。UNION の結果は UNION ALL と同じですが、重複ローが削除されている点が異なります。重複ローを削除するには余分な処理が必要なため、可能であれば UNION の代わりに UNION ALL を使用してください。UNION DISTINCT は UNION と同じです。

2 つの select リスト中の対応する項目が異なるデータ型の場合、Ultra Light は結果の中から対応するカラムのデータ型を選択し、各コンポーネント SELECT 文のカラムを自動的にそれぞれ変換します。

表示されるカラム名は、最初の SELECT 文に対して表示されるカラム名と同じです。結果セットのカラム名をカスタマイズするには、SELECT 文で WITH 句を使用するという方法もあります。

ORDER BY 句では、整数を使用して順序が確立されます。ここで整数は結果をソートするクエリ式を示します。

参照

- 「Ultra Light SELECT 文」 401 ページ

例

次の例は、従業員と顧客のそれぞれの姓すべてをリストします。

```
SELECT emp_lname
FROM Employee
UNION
SELECT lname
FROM Customer
```


Ultra Light UPDATE 文

データベース・テーブルにある既存のローを変更します。

構文

UPDATE *table-name*
SET *column-name* **=** *expression*, ...
[**WHERE** *search-condition*]

パラメータ

table-name *table-name* は、更新するテーブルの名前を指定します。使用できるのは単一テーブルのみです。

SET 句 それぞれ指定したカラムを、等号の右側の式の値に設定します。式には制限がありません。式が *column-name* である場合は、古い値が使われます。

SET 句で指定されたカラムの値のみ変更されます。特に、UPDATE を使用して、カラムの値をデフォルトに設定することはできません。

WHERE 句 WHERE 句を指定すると、*search-condition* を満たすローだけが更新されます。「Ultra Light の探索条件」 280 ページを参照してください。

備考

UPDATE 文は、テーブル内の値を修正します。

テーブルに挿入した文字列は、データベースが大文字と小文字を区別するかどうかに関係なく、常に入力された大文字と小文字の状態のままで格納されます。

参照

- 「Ultra Light INSERT 文」 399 ページ
- 「Ultra Light DELETE 文」 395 ページ
- 「Ultra Light の探索条件」 280 ページ

例

次の例は、従業員 Philip Chin (従業員番号 129) を営業部からマーケティング部 (部門 400) に移動します。

```
UPDATE employee
SET dept_id = 400
WHERE emp_id = 129
```

索引

記号

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

あ

い

う

え

お

か

こ

す

せ

そ

た

ち

つ

て

と

な

に

は

ひ

ふ

へ

ほ

ま

め

も

ゆ

