SQL Anywhere Studio 9 オフィシャルガイド

使える！ SQL Anywhere®

iAnywhere A SYBASE COMPANY

アイエニウェア・ソリューションズ株式会社
森脇 大悟 著

こんな人におすすめです

エンタープライズDBを
省リソース省コストで運用したい！

部門レベルのDBシステムを
手軽にすばやく構築したい！

SQL Anywhereをバージョンアップしたい！

モバイルアプリケーションを開発したい！

パッケージソフトにDBを組み込みたい！

SE SHOEISHA

iAnywhere
A SYBASE COMPANY

SQL Anywhere Studio 9 オフィシャルガイド

使える！ SQL Anywhere®

アイエニウェア・ソリューションズ株式会社
森脇 大悟　著

SE
SHOEISHA

本書内容に関するお問い合わせについて

このたびは翔泳社の書籍をお買い上げいただき、誠にありがとうございます。弊社では、読者の皆様からのお問い合わせに適切に対応させていただくため、以下のガイドラインへのご協力をお願いしております。下記項目をお読みいただき、手順に従ってお問い合わせください。

● お問い合わせの前に

弊社Webサイトの「正誤表」や「出版物Q&A」をご確認ください。これまでに判明した正誤や追加情報、過去のお問い合わせへの回答（FAQ）、的確なお問い合わせ方法などが掲載されています。

正誤表　http://www.seshop.com/book/errata/
出版物Q&A　http://www.seshop.com/book/qa/

● ご質問方法

弊社Webサイトの書籍専用質問フォーム（http://www.seshop.com/book/qa/）をご利用ください。記載漏れ、お電話や電子メールによるお問い合わせ、本書にはさみ込まれたアンケートはがきなど別紙に記入されたご質問は、お受けしておりません。

※ 質問専用シートのお取り寄せについて

Webサイトにアクセスする手段をお持ちでない方は、ご氏名、ご送付先（ご住所／郵便番号／電話番号またはFAX番号／電子メールアドレス）および「質問専用シート送付希望」と明記のうえ、電子メール（qaform@shoeisha.com）、FAX、郵便（80円切手同封）のいずれかにて“編集部読者サポート係”までお申し込みください。お申し込みの手段によって、折り返し質問シートをお送りいたします。
シートに必要事項を漏れなく記入し、“編集部読者サポート係”までFAXまたは郵便にてご返送ください。

● 回答について

回答は、ご質問いただいた手段によってご返事申し上げます。ご質問の内容によっては、回答に数日ないしはそれ以上の期間を要する場合があります。

● ご質問に際してのご注意

本書の対象を越えるもの、記述個所を特定されていないもの、また読者固有の環境に起因するご質問等にはお答えできませんので、予めご了承ください。

● 郵便物送付先およびFAX番号

送付先住所　〒160-0006　東京都新宿区舟町5
FAX番号　03-5362-3818
宛先　（株）翔泳社 出版局 編集部読者サポート係

※ 本書に記載されたURL等は予告なく変更される場合があります。
※ 本書の出版にあたっては正確な記述に努めましたが、著者および出版社のいずれも、本書の内容に対してなんらかの保証をするものではなく、内容やサンプルに基づくいかなる運用結果に関してもいっさいの責任を負いません。
※ 本書に掲載されている画面イメージなどは、特定の設定に基づいた環境にて再現される一例です。

Introduction

はじめに

SQL Anywhere Studio 9がリリースされ、『SQL Anywhere Studio 8 公式デベロッパーズガイド』に引き続き、本書を発行できる運びとなった[1]。前書は開発に主眼を置いたが、本書はSQL Anywhereの構造や管理に重点を置いた。本書の前半では、データベースを取り上げ、後半では、Mobile Link同期テクノロジやUltra Lightについて解説した。筆者は、SEとして、SQL Anywhereの紹介や導入支援・コンサルティング業務を行っているが、ユーザやパートナーとのやり取りの実体験を踏まえて、SQL Anywhereの見所や間違いやすい点を紹介したつもりである。

本書は、SQL Anywhereに限らずデータベースにある程度触れたことのあるユーザを対象としている。SQLでデータベースのデータを扱ったことはあるが、データベースの内部動作まではよく知らないようなユーザだ。SQL Anywhereは、高機能でありながら、やさしいデータベースなので、データベースの詳しい知識がなくとも利用できる。しかし、知識があれば、より深く利用することができるので、そのような解説を試みた。SQL Anywhereの既存ユーザであれば、細かな動作を知ることにより、今まで以上に楽しくSQL Anywhereに接することができるだろう。SQL Anywhereを初めて動かすユーザであれば、高機能なデータベース機能を簡単に利用できるSQL Anywhereの特徴がわかるであろう。また、Mobile Link同期テクノロジは、ほかに例を見ない機能や柔軟性を備えた誇るべきもので

1 『SQL Anywhere Studio 8 公式デベロッパーズガイド』の全内容は、付属CD-ROMのDisc 1内にPDFで収録されている。

あり、特に詳しく取り上げた。本書を参考に、SQL Anywhereの潜在能力を引き出して利用していただけたら、筆者としてこの上ない喜びである。

本書では、SQLを基礎から説明することはしなかった。SQLについては、数多くの良書があるのでそちらを参考にしてほしい。巻末の参考文献に挙げたように、SQL Anywhereの書籍もある。また、データベース設計やモデリングも本書の範囲外とした。開発言語自体の説明も扱っていないが、『SQL Anywhere Studio 8 公式デベロッパーズガイド』に参考となる部分があるので参照するといいだろう。

なお、本書の執筆にあたっては、日本語版のSQL Anywhere Studio 9.0.1または9.0.2 Windows版を基にした。旧バージョンとの機能比較については、巻末の新機能ガイドやマニュアルを参考にしてほしい。また、Linux版でも同様にSQL Anywhereは機能する。本書ではWindows上の表記を用いたが、特に注意すべき点はLinuxについても併記した。

Acknowledgement

謝辞

前書に引き続き執筆の機会を与えて頂いたアイエニウェア・ソリューションズ社の早川典之氏、Richard Morgan氏にお礼を申し上げます。原稿の査読や校正に協力して頂いたシステムエンジニアリング統括部の磯辺信雄氏、遠藤友二氏、齋藤弘志氏、近藤兼充氏、森山誠氏、小川浩一氏、浅野篤氏、神田佳行氏、宋淑敬氏に感謝します。営業推進室の伊藤沢氏には、出版社との打ち合わせや企画立案にご尽力いただき、大変助かりました。

翔泳社の岩切晃子氏、萩原敬生氏、宮原純子氏、押久保剛氏、佐野あさみ氏からは、優れた編集や素敵なデザインを頂戴し、ありがとうございます。また、執筆が一時中断してしまうなどお待たせしてしまい、大変ご迷惑をおかけしました。無事に発行できて嬉しいです。また、会議室の一角をお借りして執筆に専念できました。その際、田原瑞穂氏には毎朝鍵を開けてもらい助かりました。

最後に、本を生み出す苦しみを私が味わっているそばで、子を産む苦しみを乗り越えた妻の瑞穂に感謝します。机に向かっていると、うしろから玲音が笑顔でハイハイしてきて、気が明るくなりました。すぐ追い返しちゃってごめんね。

2005年8月　森脇 大悟

Contents

Contents

Contents

Contents

第1章 SQL Anywhere Studioとは

1.1 SQL Anywhere Studioとは

iAnywhere Solutions, Inc.（以下、iAnywhere Solutions社）は、企業情報システムを「いつでも、どこでも」「オンラインでも、オフラインでも」利用できるように、そのインフラとなるソフトウェアをパッケージ製品として提供している。データベースや同期ツールを含む「SQL Anywhere Studio」や、PCやPDAにインストールされたソフトウェアを管理する「Manage Anywhere Studio」、Webベースのアプリケーションやコンテンツをモバイルデバイスで利用可能にする「M-Business Anywhere」、RFIDデータを管理する「RFID Anywhere」、自然言語によるGUIや検索を可能にする「Answers Anywhere」などのパッケージ製品がある。また、製品の保守や製品にまつわるコンサルティングサービスも手がける。

本書では、主力製品である「SQL Anywhere Studio」を解説する。まず、開発元のiAnywhere Solutions社と製品の簡単な歴史を紹介してから、製品概要とデータベースの役割について説明する。技術的詳細については、次章以降で解説する。

1.1.1 歴史

SQL Anywhere Studioの歴史は、Watcom SQLというデータベースから始まる。このデータベースを開発したのはWatcom International Corporation（以下、Watcom社）である。Watcom社は、1981年にカナダのWaterloo大学の研究グループが母体となって誕生した。そして1992年、MS-DOSとQNX上で動作する省リソースなデータベースとしてWatcom SQL 3をリリースした[1]。

1　同社は、Watcom Cコンパイラという製品でも有名である。現在、Watcom Cコンパイラはオープンソースとなり、Open Watcomの名称で今日も利用可能である。
Open WatcomのWebサイト：http://www.openwatcom.org/

1994年、Watcom社はPowersoft社に買収され[2]、続く1995年、Powersoft社はSybase社に買収された。Watcom SQLは、Sybase社の製品群に加えられ、「Sybase SQL Anywhere 5.0」と名前を変えて発売された。現在と同じ製品名称が用いられたのは「SQL Anywhere Studio 6」からである。1999年、Sybase社内に、Mobile and Embedded Computing（MEC）事業部が設立され、この事業部がSQL Anywhereの開発・サポートを担当するようになる。2000年、MEC事業部が分社化され、iAnywhere Solutions社が誕生した[3]。

一方、日本では、サイベース株式会社のアイエニウェア・ソリューション事業部がSQL Anywhere Studioの販売を行っていたが、2003年にこの事業部は、iAnywhere Solutions社の100％子会社であるアイエニウェア・ソリューションズ株式会社（以下、アイエニウェア・ソリューションズ社）として独立し、現在に至る[4]。

表1.1：製品年表

1992年	Watcom SQL 3リリース
1994年	Watcom SQL 4
1995年	Sybase SQL Anywhere 5
1996年	Sybase SQL Anywhere 5.5
1998年	SQL Anywhere Studio 6
2000年	SQL Anywhere Studio 7
2001年	SQL Anywhere Studio 8
2003年	SQL Anywhere Studio 9
2004年	SQL Anywhere Studio 9.0.1 日本語版

2　Powersoft社は、統合開発環境であるPowerBuilderの製造元であった。現在、PowerBuilder 10が発売中である。

3　iAnywhere Solution社 Webサイト：http://www.ianywhere.com/

4　アイエニウェア・ソリューションズ社 Webサイト：http://www.ianywhere.jp/

表1.2：会社年表

1981年	Watcom International Corporation設立
1994年	Powersoft社がWatcom社を買収
1995年	Sybase社とPowersoft社が合併
1999年	Sybase社MEC事業部が設立
2000年	iAnywhere Solutions, Inc.設立
2001年	サイベース株式会社アイエニウェア・ソリューション事業部が設立
2003年	アイエニウェア・ソリューションズ株式会社が設立

1.1.2 SQL Anywhere Studioのターゲット市場

SQL Anywhere Studioがターゲットとする市場は3つある。

1. モバイルソリューション市場

現在の企業活動においては、オンライン／オフラインを問わず、「いつでも、どこでも」会社のデータにアクセスして活用できることが求められている。そのためには、企業の情報システムをモバイル化するためのツールが必要だ。

SQL Anywhere Studioは、ノートパソコンやPDAといったモバイル端末でも動作するデータベースをユーザに提供する。また、データ同期テクノロジもある。代表的な活用分野はCRM（Customer Relationship Management）やSFA（Sales Force Automation）だ。顧客情報や商品情報などをデータベース化し、ノートパソコンやPDAに入れて携帯することで、渉外活動中に企業データを利用できる。データが個々のデバイスに分散してしまうことになるが、同期テクノロジによりデータを中央に集約することが可能だ。

■ 2. アプリケーションやハードウェアへの組み込み市場

パッケージソフトなどのアプリケーションでは、データの保存や検索といったデータベース機能が必要なことが多い。SQL Anywhere Studioは、たとえば、企業会計パッケージやPOS端末などに組み込まれて利用されている。このような利用形態の場合、データベースが高性能なマシン上で使用されるとは限らないが、SQL Anywhereならばエンドユーザにデータベースの存在を意識させないほど省リソースで動作する。また、数多くの店舗や代理店に配備するシステムとして大量のローカルデータベースが必要とされるようなケースにも、SQL Anywhereは適している。SQL Anywhereは、省リソースかつ高機能で、アプリケーションに組み込みやすいからだ。

■ 3. 中小企業・中小規模向け（SMB）市場

中小企業や大企業の支店などの中小規模向け（Small to Medium sized Businesses）市場で用いられるデータベースには、エンタープライズ級の高性能・高機能が求められる一方で、専任のIT担当者やデータベース管理者を設けられないため、できるだけ管理コストを抑えることも必要とされる。

SQL Anywhereは、管理の手間を不要とすることを設計コンセプトに掲げている。また、バージョンを重ねるにつれて中・大規模向けの機能が拡充され、エンタープライズレベルの用途にも対応している。クライアント／サーバシステムやWebアプリケーションのバックエンドにあるデータベースサーバを、PCサーバ上で運用することも多いだろう。そのような場合にも、SQL Anywhereの利用価値は高い。

1.1.3 コンポーネント

SQL Anywhere Studioには、3つの代表的なコンポーネントとGUI管理ツールがある。

■ SQL Anywhere（データベース）

SQL Anywhereは、SQL Anywhere Studioのメインのコンポーネントであるデータベースだ。Adaptive Server Anywhere（ASA）が現在の正式名称であるが、いずれSQL Anywhereに統一される予定となっている。このため、本書ではSQL Anywhereという表記を用いることにする。

■ Ultra Light（データベース）

Ultra Lightは、Windows CEやPalm OSに対応する超軽量データベースである。

■ Mobile Link（同期テクノロジ）

Mobile Linkは、SQL AnywhereやUltra Lightとほかのデータベースとを同期するためのコンポーネントである[5]。

■ GUI管理ツール

GUI管理ツールとして、データベースを管理するSybase CentralやSQLを実行するInteractive SQLなどがある。

次章以降では、SQL Anywhereの技術的詳細について解説していくが、Mobile Link とUltra Lightについては独立した章を設けて説明する。

5　このほか、Mobile Link以外の同期テクノロジとしてSQL Remoteと呼ばれるコンポーネントもあるが、本書では説明を割愛する。

1.2 データベースとは

そもそも、どうしてデータベースを使うのだろうか。ファイルにデータを書き込むプログラムを書くのは造作ないにもかかわらず、データ管理専用のソフトウェアをわざわざ用いる利点はなんだろうか。この点について改めて考えてみたい。

データベースと一口に言ってもさまざまな種類がある。SQL Anywhereでも採用しており、最もよく使われているのがRDBMS（Relational DataBase Management System）と呼ばれているものである。

RDBMSは、データを2次元の表（テーブル）のように管理する。テーブルはカラムと行からなり、カラムがデータ項目を意味し、1件1件のデータが行として記録される。一般にテーブルは多数あるので、必要な情報が複数のテーブルにまたがっていることがあるが、テーブル同士を結合するなどして、ほしい情報だけを選択できる。テーブルはエンティティと呼ばれることもあるが、本書ではデータモデルといった概念は扱わないので、単純に「テーブル」と呼ぶことにする。同様に、タプルやレコードやローではなく「行」を使用し、フィールドや属性ではなく「カラム」で統一する。

データベースを用いる利点は、データとアプリケーションとを分離し、別々に扱えるようになることである。その結果、保守管理しやすくなり、開発生産性も向上する。では、もう少し詳細に見ていこう。

■ データの独立性

アプリケーションがデータベースを使わず、たとえば、カンマ区切りのCSVファイルにデータを1件ずつ記録し、その処理をアプリケーション自身が実装するような場合を考えてみよう。この場合、データ保存の物理的性質にアプリケーションが強く依存することになる。

システムの利用が長くなり、業務処理の拡大に伴ってデータ形式の変更が迫られ

たとする。記録すべきデータ項目を増やしたり、記録されているデータの型を数値から文字列に変更したり、カンマ区切りをタブ区切りに変更したりしなければならなくなった場合、データファイルだけでなく、アプリケーション上の処理も修正が必要になる。

あるいは、扱うデータ量が増えて、処理速度を向上させる機能を追加することもあるだろう。インデックス検索できるようにしたり、ファイルを分割して管理できるようにしたりするような場合も、相応のロジック変更や追加が必要になる。

アプリケーションとデータとの依存性が高いシステムは、一方の変更が他方に影響するため、保守性や拡張性に劣る。このような事態を避けるため、CSVファイルを扱う場合であっても、データ操作部分をライブラリ化したりして、アプリケーションとデータとの依存度を下げるように努めるだろう。

この「ライブラリ化されたデータ操作」を突き詰めたものがデータベースだ。データベースはデータの保存や管理に専念し、SQLという標準的なアクセス手段を提供する。その際の接続方法もODBCやJDBCといった標準化されたインターフェイスを使う。アプリケーションは、そのようなインターフェイスを利用して、データベースの物理的性質が隠蔽されたままデータにアクセスする。

データベースの導入により、アプリケーションとデータとの独立性が高まり、それぞれを個別に管理することが可能となる。検索速度を向上させようとデータベースファイルやインデックスを追加しても、アプリケーションのコードを変更する必要はない。管理対象となる情報を広げようとテーブルやカラムを増やしても、アプリケーション側の変更は、データベースを使用しない場合よりもずっと少ないだろう。逆に、アプリケーション側でロジックを変更しても、データベースに手を加えることはない。さらに、データベースに接続するアプリケーションの種類が増えても、「標準的な接続方法」に対応してさえいれば、データベースはそのままでよいのである。

■ ポータビリティ

データベースの物理的詳細についてアプリケーションは関知せず、アプリケーションとデータベースは独立した関係になるため、別のOSのマシンにデータベースを移行することも可能となる。ファイル構造や文字コードといったOSごとの差異がデータベースサーバで吸収されるからだ。特に、SQL Anywhereではデータベースファイルが1つで済み、しかもファイルフォーマットがOSに依存しないため、データベースの移行は容易である。

■ データベースの中央管理

複数システムが利用するデータであっても、1つのデータベースに集中させることができるので、バックアップやチューニングといった維持管理を一ヶ所で行うことができ、都合がよい。

逆に言うと、相当量のデータ処理を一ヶ所で行うことになるため、管理責任も増す。規模が大きくなれば、データベース管理者（Database Administrator：DBA）を割り当て、保守にあたらせる必要がある。さらに大規模であれば、DBAチームを編成しなければならないだろう。DBAは、貴重なデータが障害時に失われないよう気を配るのはもちろんであるが、多数のアクセスをこなせるようにデータベースやクエリをチューニングしたりする。このような運用管理だけでなく、データ管理の視点からシステム設計の手助けをする場合もある。

SQL Anywhereは、保守の手間をなるべく省くように設計されたデータベースである。ファイルサイズの拡張や統計情報の更新は自動でなされ、コストベースのオプティマイザによりクエリの実行プランも適宜見直される。また、インデックスの作成を支援するインデックスコンサルタントという機能もある。

■ 信頼性

データベースに求められる信頼性はさまざまな面がある。たとえば、① 意図しな

いデータが記録されない、② データベースがいつでも利用できる、③ 障害が起きても修復可能、などが挙げられる。

データベースでは、データの記録はトランザクション単位で行われる。トランザクションとは、データベースを利用する側にとって、これ以上分割することのできない処理をまとめたものだ。トランザクションの実行結果は、「成功した」「失敗して元に戻った（ロールバックされた）」の2つしかないため（トランザクションの原子性）、トランザクションの途中でたとえ障害が起きたとしても、ユーザが意図しない中途半端なデータの状態にデータベースが陥ることはない。

保存されるデータの値に制約をつけることも可能だ。たとえば、整数型で1から100までの値だけ許可するように条件を設定できる。複数のアプリケーションがデータベース上の同一のデータを利用するときなどはアプリケーション間の整合性を保つのが難しくなるが、データベース上で制限を設ければ、意図しないデータの入力を未然に防げる。

データベースでは、データを複数のファイルに記録して冗長にすることで、耐障害性を高めている。トランザクションログの利用がこれにあたる。また、1つのデータベースに対して複数のファイルを割り当て、記録を分散することもある。ファイルを別々のディスクに置くことは、ディスク障害の影響を減らし、パフォーマンスの向上にもつながる。

たいていのデータベース製品にはバックアップツールが付属しており、データベースが稼働中のままバックアップを行うことができる。データベースサーバに障害が起こったときは、リカバリツールなどを用いて、バックアップから復旧する。

■ セキュリティ

データベースにはセキュリティ機能も必要だ。たとえば、データベースファイルの暗号化やデータ通信の暗号化により、第三者からの盗聴を防ぐ。データベースにアクセスするユーザを設定して、データアクセスに対して認証・承認をかけることも

できる。データベース内で可能な処理の権限をユーザごとに指定するのである。また、誰がどのような操作をしたのか、監査履歴を残すことも可能だ。

■ パフォーマンス

複雑なSQL処理であっても、データベースサーバはインデックスなどを用いて効率的にデータを取得するので、処理の遅いものがあれば、インデックスの追加などでチューニングできる。もちろん、このようなチューニングはアプリケーションの変更を必要としない。また、一度アクセスしたデータは、データベースサーバのメモリ上にキャッシュされるので、頻繁に参照されるデータはメモリ上に残りやすくなり、システム全体のパフォーマンスが向上する。

チューニングを手助けしてくれるツールを利用することもできる。たとえば、SQL Anywhereには、「Index Consultant（インデックスコンサルタント）」というツールがある。どのカラムにどのようなインデックスを作成すべきか判断するのは元来難しいものだが、このツールはその作業を支援・自動化してくれる。

複数のユーザがデータベースにアクセスする場合は、データの共有と同時アクセスという相反する課題を解決する必要がある。データベースは、排他制御の仕組みなどを用いて、論理的な整合性を保ちつつ、並列処理して効率を上げている。SQL Anywhereは、行ロックを用いて、並列処理の整合性を実現している。

■ 開発生産性

これまで見てきたような機能を自分で実装しようとしたら、大変な工数になるはずであり、そこにデータベース製品を利用する価値がある。それだけでなく、データアクセスが標準化されるので、サードパーティ製のツールが利用しやすくなる。さらに、データベースの知識は製品に依存しない部分も多いので、知識や実装の共有につながるといった効果もある。このように、データベースの利用により開発生産性が向上する。

1.3 SQL Anywhereの主な特徴

前節で挙げたようなRDBMSの機能が使いやすくなっていることが、SQL Anywhereの特徴である。

■ 容易なインストール・運用管理

インストールはウィザード形式になっていて、[次へ] ボタンをクリックしていけば完了する。そして、インストールした初期設定のままで高パフォーマンスが発揮・持続されるように設計されているので（優れた自己チューニング機能など）、運用管理コストが低く、専任のデータベース管理者をおく必要はない。データベースファイルサイズは自動拡張し、キャッシュサイズも自動増減する。

アプリケーションに組み込む際は、サイレントインストールやEXE・DLLファイル単位での配備が可能なため、組み込みやすい。さらに、データベースに対するオプションは設定ファイルではなく、起動コマンドやデータベース内に保存されるオプションで指定するため、インストール後の管理も容易だ。また、データベースサーバはサービスとして登録することもできるし、クライアントからの接続時にデータベースを自動起動することもできる。

なお、軽量データベースであるUltra Lightは、アプリケーションのコードと一体化するため、さらに組み込みやすくなっている。

■ エンタープライズレベルのフル機能RDBMS

SQL Anywhereは、およそRDBMSに求められるであろう機能をすべて備えており、正統派のRDBMSと言える。主な機能には、トランザクション処理、リカバリ処理、オンラインバックアップ、参照整合性のサポート、ストアドプロシージャ、トリガ、行ロック、スケジュールとイベントなどがある。

■ GUIによる管理ツール

Sybase CentralやInteractive SQLなど、GUIによる管理ツールが標準で搭載されている。管理コマンドも充実しているので、管理処理をバッチ化することも可能だ。

■ 省リソース

必要最小メモリ量は8MBである[6]。

■ スタンドアロンにもネットワークサーバにも利用可能

パッケージソフトに組み込まれたデータベースとしてスタンドアロンで利用したり、クライアント／サーバシステムやWebシステムのバックエンドでネットワークデータベースとして利用することもできる。

スタンドアロンで利用する場合、アプリケーションの要求に応じて自動的にデータベースの起動や停止が可能だ。この機能は、SQL Anywhereをアプリケーションへ組み込んだときに威力を発揮する。

■ 高性能

省リソースで低い管理コストにもかかわらず、高性能である。マルチCPUに対応し、コストベースの先進的なクエリオプティマイザを持つ。また、パフォーマンスをモニタリングしたりチューニングしたりするツールも備える。

■ 標準的なインターフェイス

業界標準的なプログラミングインターフェイスに対応している。ADO.NET、ODBC、OLE DB、JDBC（Type 2/3/4）、Perl DBD、PHPに対応し、embedded SQLやSybase Open Clientのインターフェイスもある。

6　さらに、1接続あたり4KB（Linux版では8KB）必要となる。

■ クロスプラットフォーム

WindowsやLinux、Mac OS Xなど、多数のOSに対応している。また、SQL Anywhereが使用するデータベースファイルはどのOS版のものでも共通なので、データベースファイルをコピーするだけでデータを移行できる。

■ 先進的で柔軟な同期テクノロジ

分散したデータベースを同期するSQL RemoteやMobile Linkがある。特に、Mobile Linkは、他社のデータベース製品との同期や競合解決などを可能にするなど、機能が豊富だ。

1.4 対応プラットフォーム

SQL Anywhere 9.0.1が対応しているプラットフォームを以下に挙げる。

- Windows 95/98/Me/NT/2000/XP 32bit/XP 64bit（IA64、x64）/XP Embedded/Tablet PC
- Windows Server 2003 32bit/64bit
- Windows Mobile（Pocket PC/Handheld PC）
- Linux 32bit（x86）/64bit（AMD64、EM64T、IA64）
- Mac OS X
- Novell NetWare
- Solaris/SPARC
- HP-UX、IBM AIX、Compaq Tru-64

ただし、日本で発売されているのは、2005年7月現在、Windows版（Windows Mobileを含む）とLinux版だけである。対応プラットフォームに関する最新情報は、アイエニウェア・ソリューションズ社のWebサイトに掲載されている[7]。

7 http://www.ianywhere.jp/sas/os.htmlまたはhttp://www.ianywhere.com/products/supported_platforms.html

第2章 SQL Anywhereの使い方

2.1 インストール

この節では、SQL Anywhere Studioのインストール方法を説明する。インストール自体は簡単なので、ここでは、わかりにくい部分のみ取り上げる。

なお、同梱されているDeveloper Editionの利用には、あらかじめユーザ登録（無償）が必要だ。詳細は付録Cを参照していただきたい。

2.1.1 Windows版のインストール

setup.exeを実行すると、ウィザードに沿ってSQL Anywhere Studioをインストールできる。

■ コンポーネントの選択

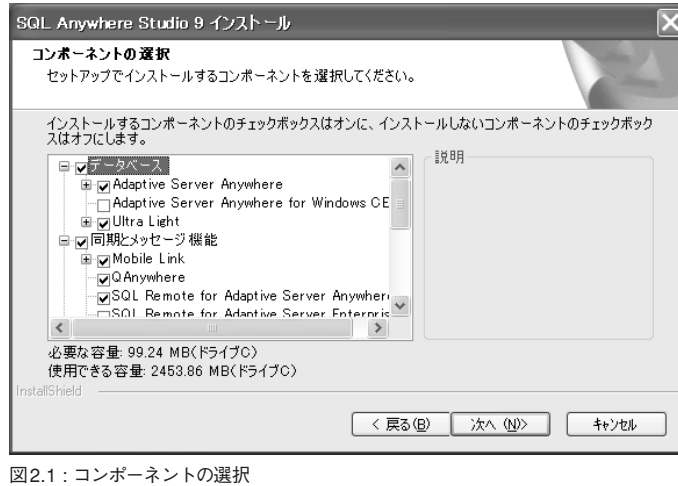


図2.1：コンポーネントの選択

この画面で表示されている、選択可能なコンポーネントの意味は次のとおりである。

表2.1：選択可能なコンポーネント

コンポーネント	説明
Adaptive Server Anywhere	Windows用のデータベース
Adaptive Server Anywhere for Windows CE	Windows CE用のデータベース
Ultra Light	Ultra Lightデータベース。第8章で説明するように、静的インターフェイス版やコンポーネント版がある
Mobile Link	Mobile Link同期コンポーネント
QAnywhere	Mobile Linkをインフラとしたメッセージ通信
SQL Remote for Adaptive Server Anywhere	SQL Remote同期コンポーネント（ASA同士）
SQL Remote for Adaptive Server Enterprise	SQL Remote同期コンポーネント（ASAとASE）
Sybase Central	管理ツール
Install Shield配備コンポーネント	Install Shieldでインストーラを作成する際のテンプレート集
PowerDesigner 9	データベース設計ツール（ASAの動作に必要ない）
InfoMaker 9	レポートやフォームを作成する開発ツール（ASAの動作に必要ない）
アクセシビリティの有効化	米国連邦身体障害者法の第508項に準拠することを目的として、ユーザ補助ツールの使用を可能にするコンポーネント

なお、本書ではデフォルトの状態から［PowerDesigner 9］と［InfoMaker 9］のチェックを外した設定とする。

■ ライセンス情報の入力

有償製品版のインストールではライセンス情報を入力する必要がある。ライセンス数を入力し、ライセンスのタイプを選択する。なお、付属CD-ROM（Disc 1）のDeveloper Editionをインストールする場合は、この画面は表示されない。

図2.2 : ライセンス情報の入力

2.1.2 インストールされる環境変数

インストールされる環境変数として、以下のものがある。

- ASANY9

  SQL Anywhere Studioがインストールされたディレクトリを示す。サンプルプログラムなどで利用される。本書でも、インストールされたディレクトリを%ASANY9%で示す。デフォルトでは、C:¥Program Files¥Sybase¥SQL Anywhere 9になる。

- ASANYSH9

  共有ファイルがインストールされたディレクトリを示す。Interactive SQLやSybase Centralやコンソールコマンドが共有コンポーネントを検索するのに利用される。デフォルトでは、C:¥Program Files¥Sybase¥Sharedになる。

自動的に設定されるわけではないが、知っていると便利な環境変数がある。

- ASTMP

  テンポラリファイルが作成されるディレクトリを示す。

- SQLCONNECT

  デフォルトの接続パラメータを設定できる。

2.1.3 Linux版のインストール

setupスクリプトを実行すると、CUI（Character User Interface）によるインストーラが起動する（図2.3）。デフォルトでは、/opt/sybase以下にインストールされる。Windows版と同様にコンポーネントの選択が可能だが、本書ではデフォルトの設定のままとする。

```
$ cd /cdrom
$ ./setup
```

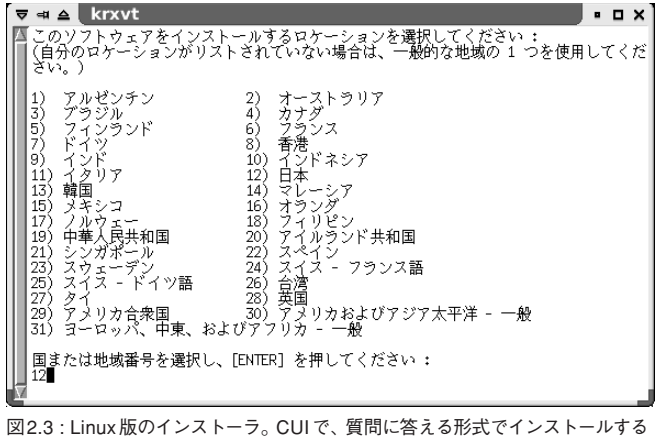


図2.3：Linux版のインストーラ。CUIで、質問に答える形式でインストールする

SQL Anywhereを利用するには、ライブラリへのパスや環境変数に注意する。イ

ンストールしたディレクトリのbinディレクトリ内に設定スクリプトasa_config.sh（またはasa_config.csh）があるので、これを実行すると環境変数などが設定される。

```
$ source ./asa_config.sh
```

2.2 GUIツール

2.2.1 Sybase Central

Sybase Centralは、GUIによるデータベース管理ツールで、データベースの作成やテーブルの作成など、SQL Anywhere管理のあらゆる操作が行える（図2.4）。また、テーブルのデータを表示して修正することもできるので、簡単なデータ操作も可能だ（図2.5）。

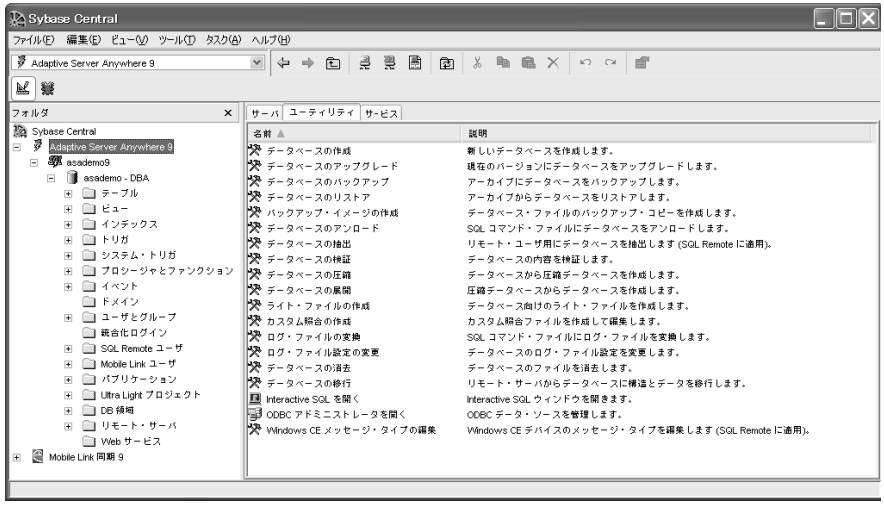


図2.4 : Sybase Centralのユーティリティのメニュー。データベースの作成やバックアップなどのメニューなどが並んでいる

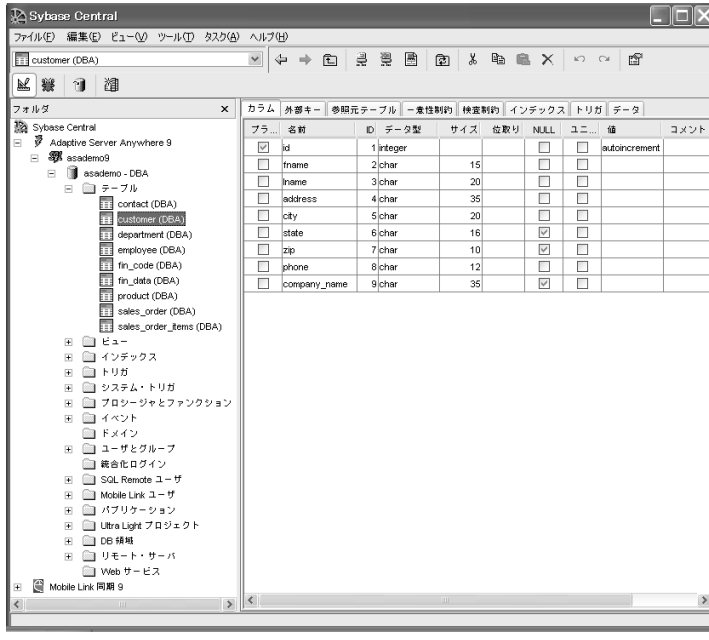


図2.5 : Sybase Central でのスキーマ表示。テーブルのスキーマを表示させたところ。カラムの追加や修正などをデータベース起動中に行える。また、データタブでデータ一覧を表示可能。左のペインに「Mobile Link 同期9」とあるが、これは、統合データベースに接続してMobile Linkの設定を行うための場所である

各オブジェクトの操作は、右ペインのタブや左ペインのオブジェクトを右クリックして表示されるポップアップメニューから行う。なお、左ペインでテーブルを選択してからコピーすると、Sybase Central内でコピーできる。また、メモ帳などにペーストすると、テーブルスキーマのSQL文がテキスト表示される。

2.2.2 Interactive SQL

Interactive SQLは、SQL Anywhereに接続してSQL文を実行するためのツールだ。GUIとコマンドラインの両方で利用できる。GUIで利用すると、SELECT文などのSQLが返す結果セットはテーブル形式で表示される。また、SQL文の実行時間

やオプティマイザが選択したクエリプランも表示できるので、SQL文のチューニングに役立つ。

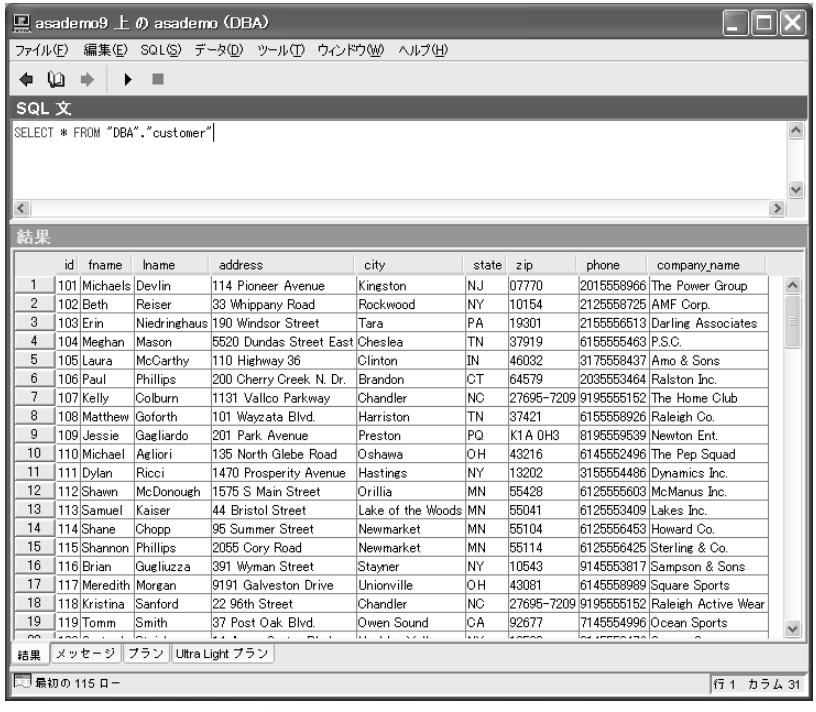


図2.6 : Interactive SQL。テーブルのデータを一覧表示させたところ

Interactive SQLの実行コマンドであるdbisql.exeに対して-noguiオプションを付けて実行すると、対話的にSQLを実行できる。次に実行例を示す。

```
$ dbisql -c "dsn=ASA 9.0 Sample" -nogui
(DBA)> select top 5 id, fname, lname from customer order by id;
実行時間 :0.04 秒
id  fname    lname
-------------------------
101 Michaels Devlin
102 Beth     Reiser
```

```
103 Erin     Niedringhaus
104 Meghan   Mason
105 Laura    McCarthy
(最初の 5 ロー)
(DBA)> exit
$
```

dbisqlコマンドの引数にSQL文指定すると、そのSQL文が実行される。

```
$ dbisql -c "dsn=ASA 9.0 Sample" "select * from customer"
```

■ SQL文のバッチ処理

dbisqlコマンドの引数にファイル名を指定した場合は、そのファイル（コマンドファイル）に書かれたSQL 文が実行される。

```
$ dbisql -c "dsn=ASA 9.0 Sample" a_command_file.sql
```

このとき、コマンドファイル内の変数をパラメータ化して、コマンドファイルを汎用化したい場合もあるだろう。Interactive SQLのREADコマンドを用いると、コマンドファイルにパラメータを渡すことができるので、バッチスクリプトを書く際に便利だ。たとえば、次のようなコマンドファイルselect_product.sqlを用意する。

```
PARAMETERS MIN_QUANTITY;
SELECT * from product
 WHERE quantity > {MIN_QUANTITY};
```

MIN_QUANTITYがパラメータだ。100という数字を渡すには、次のようなコマンドを実行すればよい。

```
$ dbisql -c "dsn=ASA 9.0 Sample" "READ select_product.sql 100"
```

2.2.3 高速ランチャ機能

Sybase CentralとInteractive SQLには、起動を速くする高速ランチャ機能がついている。1度目の起動を終了してもバックグラウンドプロセスが残り、2度目以降の起動が高速になる。この機能はデフォルトでオンになっているので、バックグラウンドプロセスのメモリ使用量などが気になる場合は、［ツール］－［オプション］から、高速ランチャ機能を無効にするとよい。

2.3 SQL Anywhereのアーキテクチャ

2.3.1 データベースのファイル構成

データベースを構成するファイルには、① データベースファイル、②トランザクションログファイル、③ テンポラリファイルの3種類がある（図2.7）。

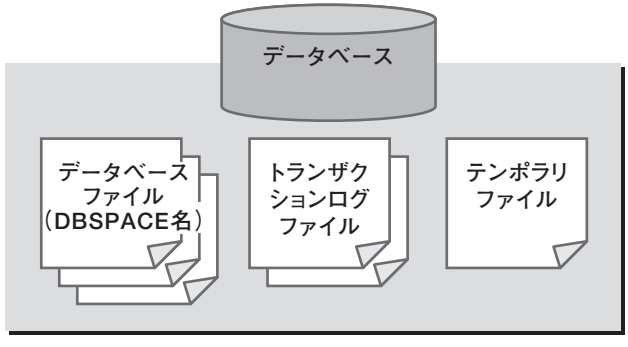


図2.7：データベースのファイル構成

■ データベースファイル（*.db）

データベースファイルには、スキーマ、データ、インデックスなど、すべてのデータベース情報が記録される。デフォルトではデータベースファイルは1つだけだが、テーブルやインデックス単位でデータベースファイルを増やせる[1]。次のような場合に複数のデータベースファイルを持つとよい。

1. ディスクI/Oの分散によるパフォーマンスの向上

複数のディスク（HDD）があるとき、複数のデータベースファイルを用意して別々のディスクに配置すれば、データベースファイルへのディスクI/O（入出力）が分散されるので、パフォーマンスが向上する。

2. OSによるファイルサイズの制限

通常、データベースファイルのサイズは、データが増えるに従って自動的に拡張されるので、日々監視するような必要はない。しかし、データ量が増えてデータベースファイルのサイズが大きくなり、OSのファイルサイズ制限にひっかかりそうなときは、データベースファイルを複数に分けてその制限を回避するとよい。

なお、保存するデータ量があらかじめ把握できる場合は、データベースファイル作成直後にファイルサイズを確保しておくことで、ファイルのフラグメンテーションの発生を抑制できる（第4章「パフォーマンスチューニング」の「4.4　フラグメンテーション」を参照）。

SQL Anywhereのデータベースファイルやトランザクションログファイルの形式はOSによらず共通なので、ファイルをコピーするだけでデータベースを移行できる。たとえば、Windows CE上のデータベースファイルをWindowsにコピーしたり、Windows上のデータベースファイルをLinuxにコピーしたりすれば、移行先でそのデータベースを利用できる。

1　合計12ファイルが上限。

Column　データベース内のオブジェクトの種類

データベースのオブジェクトには次のようなものがあり、これらすべてがデータベースファイルに保存される。

- テーブル
- 主キー、外部キー
- インデックス
- ビュー
- ストアドプロシージャ、トリガ
- ユーザ、グループ
- Javaオブジェクト

それぞれのデータベースファイルは、1：1に対応するDBSPACE名で論理的に区別される。

■ トランザクションログファイル（.log）

トランザクションログとは、データがコミット（COMMIT）されるごとに記録されるログのことで、トランザクションログファイルに保存される。デフォルトでは、1つのトランザクションログファイルがある。設定により、トランザクションログなしで運用したり、（1つ増やして）2つのトランザクションログファイルを利用したりすることも可能だ。

2つのトランザクションログファイルを利用したものを、トランザクションログミラー機能と呼ぶ。2つのファイルに対して同じログを同時に書き込み、冗長化することで耐障害性を高めている[2]。

一方、トランザクションログファイルを使わずにデータベースを運用することも可

2　トランザクションログミラーファイルの拡張子は.mlg。

能だが、パフォーマンスや耐障害性の点から、トランザクションログファイルの利用を推奨する。

■ テンポラリファイル（.tmp）

テンポラリファイルは自動的に作成・削除され、SQLによるデータのソートや結合処理でキャッシュが足りないときなどに利用される。テンポラリファイルが作成される場所は次の環境変数で指定できる。

1. ASTMP
2. TMP
3. TMPDIR
4. TEMP

これらの環境変数を順番に調べ、最初に定義されている場所が選ばれる。いずれも定義されていなければ、データベースサーバのカレントディレクトリが選ばれる。

データベースやトランザクションログファイルと別のディスクにテンポラリファイルを配置すれば、ディスクI/Oが分散され、パフォーマンスの向上が期待できる。

2.3.2 データベースサーバの構成

データベースサーバは、1つ以上のデータベースを管理できる（上限は255）。データベースサーバは1つのプロセスであり、クライアントからの接続処理、SQL文の解析や最適化、ファイルアクセスなどのすべての処理をマルチスレッドで実行する。マルチCPUに対応しているので、CPUの数を増やすことでパフォーマンスを向上させることができる（図2.8）。

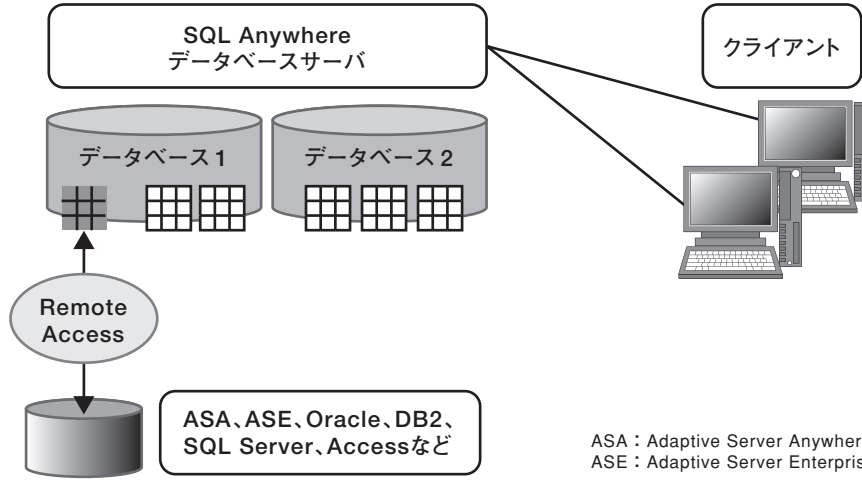


図2.8：データベースサーバの構成

データベースサーバには、次の2種類がある。

1. ネットワークサーバ
2. パーソナルサーバ

ネットワークサーバは全機能を搭載している。一方、ネットワーク接続など一部の機能を制限したのがパーソナルサーバで、次のような制限がある。

- 同一マシンからの接続のみを受け付ける（ネットワーク接続不可）
- 同時接続は10台まで
- 1CPUのみ利用可能（マルチCPU機能には対応しない）

このような制限があるため、パーソナルサーバはソフトウェア組み込み時や開発時などで使用されることが多い。

2.3.3 キャッシュ割り当て

SQL Anywhereのプロセスは1つなので、考慮すべきキャッシュ領域も1つだけだ。デフォルトでは、SQL Anywhereのキャッシュサイズは自動的に増減する。データベース処理にもっとキャッシュが必要であれば増やし、必要なければ減らす。上限下限のサイズを指定することもできるので、その範囲内でサイズが調節される。なお、デフォルトの上限サイズは、Windowsでは256MBである。大量のメモリを使用可能なときは、データベースサーバの起動時に上限サイズを大きく設定するとよい。自動調整機能を使わずに、キャッシュサイズを固定することも可能だ。設定方法などについては、本章の「2.5 動的キャッシュ割り当て」で解説する。

2.4 データベースの作成と起動

この節では、データベースの作成・削除、および接続方法などについて説明する。主にコマンドラインによる方法を説明するので、GUIなどそのほかの方法についてはマニュアルを参照していただきたい。

2.4.1 作成

データベースを作成するにはいくつか方法がある。

- Sybase CentralからGUIで行う（ウィザード）
- 既存のデータベースサーバに対してSQL文で行う（CREATE DATABASE文）
- コマンドラインで行う（dbinitコマンド）

コマンドラインでデータベースを作成するには、dbinitコマンドを使用する。

```
$ dbinit test.db
```

この文を実行すると、test.dbというデータベースファイルが作成される。トランザクションログファイルは、データベースを起動したときに自動的に生成される。

ページサイズを明示してデータベースを作成するには、-pオプションでページサイズを指定する。

```
$ dbinit -p 4096 test.db
```

■ DBSPACEのサイズを拡張する

データベースファイル（DBSPACE）のサイズは自動的に拡張するので、特に注意する必要はない。しかし、まとまった量をあらかじめ作成しておけば、ファイルのフラグメンテーションが起きにくいので、必要なサイズが概算できるならば指定しておくとよい（フラグメンテーションについては第4章の「4.4　フラグメンテーション」を参照）。

DBSPACEのサイズを増やすには、データベースを作成して起動したあとで、ALTER DBSPACE文を実行する。デフォルトのDBSPACE名はsystemなので、200MB増やす場合、次のようになる。

```
ALTER DBSPACE system ADD 200MB
```

■ データベースファイルの追加

データベースファイルを追加し、テーブルやインデックス単位でDBSPACEに振り分けることができる。次のSQL例では、既存のデータベースに対して、books.dbとbooks_index.dbとの2つのデータベースファイルを新たに作り、テーブルとインデックスとを別々に置いている。

```
-- books.db
CREATE DBSPACE books
AS 'd:¥¥books.db';
CREATE TABLE Books (
  title   char(100) PRIMARY KEY,
  author  char(50),
  isbn    char(30),
) IN books;
-- books_index.db
CREATE DBSPACE books_index
AS 'd:¥¥books_index.db';
CREATE INDEX books_isbn
    ON Books (isbn)
    IN books_index;
```

2.4.2 起動

データベースサーバを起動するコマンドは、サーバの種類ごとにある。

- ネットワークサーバ ―― dbsrv9コマンド
- パーソナルサーバ ―― dbeng9コマンド

コマンド引数の形式は同じだ。引数にデータベースファイル名を指定すると、データベースサーバが立ち上がるとともにデータベースが起動する。

```
$ dbeng9 test.db
```

複数のデータベースを起動するには、データベースファイルを列挙する。

サーバ名やデータベース名をそれぞれ明示したい場合は、-nオプションを利用する。たとえば、図2.9のような場合は次のように実行する。

```
$ dbeng9 -n server_name test1.db -n test1_db_name -n test2.db
test2_db_name
```

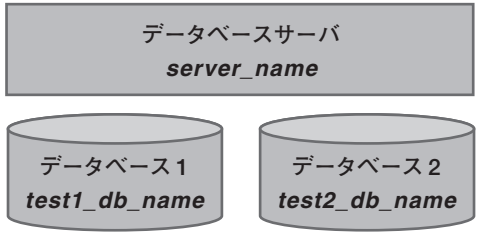


図2.9：データベースサーバ名とデータベース名

Column　データベース名やサーバ名の省略

データベース名を省略した場合は、データベースファイル名（拡張子を除く）がデータベース名となる。また、サーバ名を省略した場合は、1つ目のデータベース名がサーバ名となる。

2.4.3　接続

SQL Anywhereに接続するには、ユーザ名とパスワードが必要だ。接続先にデータベースサーバやデータベースが複数ある場合は、データベースサーバ名やデータベース名も指定しなければならない。これらのパラメータは、GUIツールであれば［接続］ダイアログボックスに入力し、コマンドラインでは接続パラメータを用いる。

■ ユーザIDとパスワード

Sybase CentralやInteractive SQLでデータベースに接続しようとすると、まず［接続］ダイアログボックスが表示されるので、接続先やユーザID・パスワードを指定する（図2.10）。あらかじめ作成されているユーザは、ユーザID「DBA」・パスワ

ード「SQL」で、DBA 権限（すべての権限）を持つ[3]。

図2.10：［接続］ダイアログボックス

■ 接続先の指定

接続先となるデータベースを指定する方法はいくつかある。

1. ODBC データソース名を指定する

   データソース作成時に接続設定をあらかじめ定義しておく。

2. データベースファイル名を指定する

3　IDやパスワードの文字列は、デフォルトでは大文字と小文字が区別されない。パスワードについては、大文字と小文字を区別するように設定可能で、CREATE DATABASE文のPASSWORD CASEキーワードで指示する。

3. サーバ名を指定する

ただし、同一ネットワークドメイン内にサーバがない場合は、IPやホスト名も指定する必要がある。また、1つのサーバに複数のデータベースがある場合は、サーバ名だけでなくデータベース名もあわせて指定する。

1. ODBCデータソース名による指定

「ASA 9.0 Sample」というデータソースがデフォルトでサンプルとしてインストールされるので、これを利用してみよう。[接続] ダイアログボックスのIDタブにおいて、[ODBCデータ・ソース名] に「ASA 9.0 Sample」と入力する。データソースのほうにユーザID・パスワードが指定されているので、このダイアログボックスで入力する必要はない。

2. データベースファイル名による指定

サンプルデータベースのファイルは%ASANY9%¥asademo.dbなので、このパスを、[接続] ダイアログボックスの [データベース] タブにある [データベース・ファイル] に入力する。[ID] タブでは、ユーザID・パスワードとして「DBA」「SQL」をそれぞれ入力する。なお、[ODBCデータ・ソース名] では [なし] が選択されていることを確認しておく。なお、SQL AnywhereのODBCドライバの機能により、デフォルトでは接続時にデータベースが起動していればそこに接続し、起動していなければ自動起動後に接続する。また、切断時に接続しているクライアントがなくなれば、自動的にデータベースが停止する。「1. ODBCデータソース名による指定」の場合でも、データソースの設定でデータベースファイル名が指定されていれば、この機能が働く。

3. サーバ名による指定

あらかじめデータベースサーバが起動していれば、サーバ名（とデータベース名）

で接続先を指定できる。

接続をテストするため、あらかじめデータベースを起動する。スタートメニューから［SQL Anywhere 9］－［SQL Anywhere］－［パーソナル・サーバのサンプル］を選択すると、サンプルデータベースが起動する。

［接続］ダイアログボックスの［ID］タブでは、ユーザID・パスワードとして「DBA」「SQL」をそれぞれ入力する。なお、［ODBCデータ・ソース名］では［なし］が選択されていることを確認しておく。［データベース］タブのサーバ名で、「asademo9」を入力すると、起動しているデータベースに接続する[4]。もし、データベースサーバに複数のデータベースが起動している場合は、データベース名も入力する必要がある。

■ ネットーワークドメインを超えた接続

接続にあたって、SQL AnywhereのODBCドライバは、ブロードキャストパケットをUDPでネットワークに送信して、データベースサーバやサーバ名を探そうとする。ファイアウォールを超えて接続するような場合、このパケットが遮断されてしまい、うまく接続できない場合がある。そのようなときは、ブロードキャストを抑制すればよい。dbisqlコマンドを使う場合は、-cで接続パラメータを与える。

```
$ dbisql -c
"eng=test;uid=dba;pwd=sql;CommLinks=tcpip(HOST=192.168.1.10;DOBROADCA
ST=NONE)"
```

主な接続パラメータは次のとおり。

4　サーバ名を知るには、［パーソナル・サーバのサンプル］のプロパティから、リンク先に書かれている起動コマンドを読めばわかる。あるいは、起動後にシステムトレイに現れるアイコンをマウスで指すと、サーバ名が表示される。

表2.2：主な接続パラメータ

接続パラメータ	説明
eng	データベースサーバ名
dbn	データベース名
dsn	ODBCのデータソース名
dbf	データベースファイルのパス
uid	ユーザ名
pwd	パスワード
CommLinks	接続プロトコルの詳細設定。CommLinksパラメータにさらにパラメータを指定できる

2.4.4 停止

データベースサーバを停止するには、Sybase Centralから切断するか、dbstopコマンドで行う。dbstopコマンドでは、以下のように接続パラメータを指定する。

```
$ dbstop -c "eng=asademo9;uid=dba;pwd=sql"
```

2.4.5 削除

データベースの削除はdberaseコマンドで行う。

```
$ dberase test.db
```

この例では、データベースファイルtest.dbと使用されていたトランザクションファイルが削除される。なお、データベースファイルとトランザクションログファイルを手動で（つまり、OSのファイル操作で）削除してもよい。

2.4.6 Windows サービスに登録する方法

データベースサーバをWindowsサービスに登録するには、Sybase Centralから行うか、dbsvcコマンドで行う。

■ Sybase Central から行う方法

左ペインで［SQL Anywhere 9］を選択してから、右ペインの［サービス］タブを選択すると、サービスの管理が行える。

■ dbsvc コマンドによる方法

たとえば、ネットワークサーバを自動起動するようにサービスに登録するには、次のようにする。

```
$ dbsvc -as -s auto -t network -w mynetworkserv "C:¥Program
Files¥Sybase¥SQL Anywhere 9¥win32¥dbsrv9.exe" -x tcpip -c 8m
"C:¥Program Files¥Sybase¥SQL Anywhere 9¥sample.db"
```

2.4.7 ODBC データソースの作成

ODBCデータソースの登録は、スタートメニューの［SQL Anywhere 9］内にある［ODBCアドミニストレータ］から行うか、dbdsnコマンドで行う。

dbdsnコマンドを使って、サンプルデータベースを「test」というデータソース名で登録するには、次のように行う。

```
$ dbdsn -w test -y -c "uid=dba;pwd=sql;dbf=%ASANY9%¥asademo.db"
```

■ ODBC によるデータベースの自動起動・自動停止

デフォルトでは接続時にデータベースが起動していなければ自動起動し、切断時

にほかの接続がなければ自動終了する。これを抑制するには、接続パラメータで調整する。次に実行例を示す。

```
$ dbdsn -w test -y -c "uid=dba;pwd=sql;dbf=%ASANY9%¥asademo.db;
AutoStart=NO;AutoStop=NO"
```

■ ODBCドライバによるログの保存

ODBCマネージャを使えば、ODBCのログを取得できる。デバッグ時などに利用するとよい。ODBCアドミニストレータの［トレース］タブで、トレースの開始・終了を操作する。トレース結果はファイルに保存される（図2.11）。

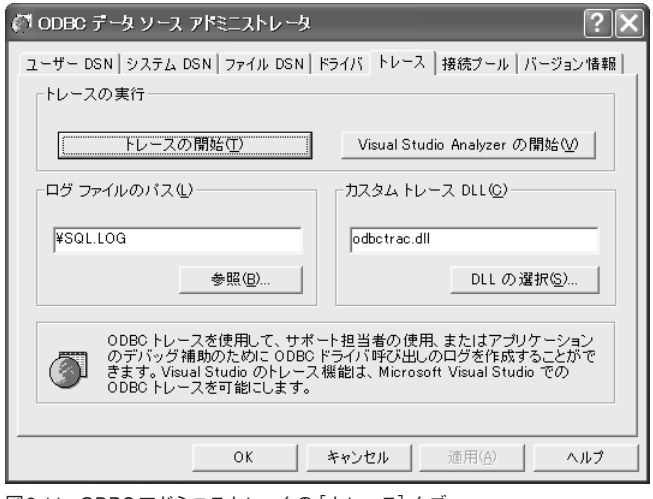


図2.11：ODBCアドミニストレータの［トレース］タブ

なお、クライアントアプリケーションから送信されたSQLのログをとる別の方法として、SQL Anywhereのログ機能を利用する方法もある（通常の使用であれば、こちらを推奨する）。これについては、第5章「バックアップとリカバリ」で紹介する。

2.5 動的キャッシュ割り当て

SQL Anywhereでは、必要に応じて、メモリ上のキャッシュサイズが動的に変更される。通常1分間隔で見直され、キャッシュが増減する。ただし、起動直後や急激にデータが増加した直後は、見直し間隔が短縮されて5秒ごとになり、これが30秒間持続する。変動しすぎることを避けるため、大幅に増減させる必要がある場合でも徐々に変わるようになっている。キャッシュの増減にあたっては、システムの負荷状況を参照し、ほかのプロセスがメモリを必要とするようであればSQL Anywhereで使用するキャッシュの割り当てサイズは控えめになる。

ただし、Linux上では、キャッシュの「動的割り当て」の意味が若干異なる。物理メモリとスワップ領域の合計をここではシステムリソースと呼ぶことにすると、SQL Anywhereは起動時に、システムリソースに対して一定量を確保する。この量は変化しないが、物理メモリとスワップ領域との構成比は動的に変わる。これが「動的割り当て」の意味である。

なお、Windows CEでは、キャッシュを動的に割り当てる機能はない。

■ キャッシュサイズの指定

キャッシュに割り当てられる初期値や最大値は、デフォルトでは、物理メモリの量やデータベースサイズなどから算出される。ここで注意すべき点は、デフォルトの最大値が次のように決まっている点だ。

表2.3：キャッシュサイズの最大値

プラットフォーム	キャッシュサイズの最大値
Windows	256MB
Windows CE	600KB
Linux	256MB

そのため、大きなキャッシュサイズが必要な場合は、初期値や最大値を指定したり、キャッシュサイズを固定したりするとよい。キャッシュに割り当てる初期値は-cオプションで指定する。最大値・最小値は、それぞれ-ch・-clオプションで指定する。

例：初期サイズ100MB、最大値500MB、最小値50MB

```
$ dbsrv9 -c 100M -ch 500M -cl 50MB asademo.db
```

キャッシュサイズを固定するには、-caオプションでゼロ（0）を指定する。

例：400MBで固定

```
$ dbsrv9 -ca 0 -c 400M asademo.db
```

キャッシュサイズを指定する際には、G（ギガバイト）・M（メガバイト）・K（キロバイト）という単位を使用できる。さらに、物理メモリに対する百分率で指定することも可能で、単位にPを用いる。

■ どのくらい割り当てるべきか

キャッシュサイズの見当をつけるには、次のように実験してみるとよい。まず、できるだけ大きなサイズ［A］（たとえば物理メモリ量）を最大値として、データベースを起動し、アプリケーションを実行するなどして負荷をかける。しばらく様子を見てから、どの程度のキャッシュが実際に割り当てられたかを確認する。それには、PeakCacheSizeデータベースオプションで、割り当てられた最大値［B］を調べる。次のコマンドを実行する。

```
SELECT property ( 'PeakCacheSize' )
```

BがAよりも小さい場合は、B程度のキャッシュがあれば十分なので、それを初期値や最大値としてデータベースを運用する。BがAと同程度のときは、B（=A）程度のキャッシュを設定するしかないが、物理メモリが不足している可能性がある。すなわち、もっとキャッシュがあれば、パフォーマンスの向上が望めるので、メモリの増設を検討するとよい。

■ AWEによるメモリ割り当て

Windows 2000/XPやWindows Server 2003では、大容量のメモリを使用するためにAWE（Address Windowing Extensions）という機能がある。たとえば、Windows Server 2003 Enterprise Editionでは、OSとして32GBのメモリをサポートするが、AWEを使わない場合、2.7GBまでしかデータベースサーバに割り当てられない。

-cwオプションを指定すると、AWEによるメモリの割り当てが行われ、大きなサイズを確保することができる。ただし、キャッシュサイズの動的な割り当ては行われず、初期値で指定した量に固定される。

例：AWEで20GBで使用

```
$ dbsrv9 -c 20G -cw asademo.db
```

2.5.1 Cache Warming機能

データベース起動直後のキャッシュにはデータがなく、データが保存され、安定したパフォーマンスが出るまで多少の時間がかかる。この問題に対処するのがCache Warming機能だ。これにより、データベース終了時点などで、キャッシュの状態がデータベースに保存され、次回起動時に再利用される。そのため、起動直後からキャッシュヒットを期待できる。

Cache Warming機能はデフォルトでオンになっている。使用したくない場合は、データベースサーバの -ccオプションや-crオプションでオフにする。

2.6 ユーザの作成

データベースにはユーザを作成できる。データベース内のテーブルやストアドプロシージャは、作成したユーザがオーナーとなり、オーナーがほかのユーザへアクセス権限を与える。テーブルを参照する権限を特定のユーザのみに限定したり、ストアドプロシージャの実行権限を制限したりすれば、データベースのセキュリティが高まる。ユーザの数が多いときは、ユーザをグループに分類し、グループ単位で管理するとよい。

なお、DBA権限という特殊な権限があり、この権限を持つユーザは（特別な設定なしに）データベース内のあらゆる権限を持つことができる。SQL Anywhereでは、デフォルトで、「DBA」という名前のユーザがあり（パスワードは「SQL」）、このユーザはDBA権限を持つ。データベース作成後、DBAユーザで必要なユーザやグループを作成し、DBAユーザを削除するか最低限パスワードを変更するのがセキュリティの観点からは望ましい[5]（パスワード変更方法は後述する）。

■ ユーザの作成

ユーザを作成するには、GRANT CONNECT文を用いる[6]。

```
GRANT CONNECT TO user1 IDENTIFIED BY password1;
```

5　DBA権限のユーザのパスワードを忘れて接続できなくなったという問い合わせを受けることがあるが、隠しユーザなどはないので、パスワードは忘れないこと。

6　Windowsのログオン認証を用いる「統合化ログイン」機能もある。その際は、GRANT INTEGRATED LOGIN文を用いる。

これで、user1が作成され、このユーザはpassword1というパスワードで接続できる。だが、このままではuser1は接続しかできないので、ほかの適切な権限を与える。たとえば、テーブルを作成する権限を与えるには、RESOURCE権限を与えればよい。

```
GRANT RESOURCE TO user1;
```

ほかのユーザother_userが所有するテーブルをuser1が参照できるようにするには、そのテーブルの参照権限があればよいので、オーナーであるother_user（もしくはDBA権限を持つユーザ）が、次のように権限を与える。

```
GRANT SELECT ON other_user.some_table TO user1;
```

権限には以下のものがある（表2.4）。参照や更新などが別々の権限となっているので、細かく管理できる。

表2.4：権限の種類

権限名	可能な構文
SELECT	SELECT文
INSERT	INSERT文
UPDATE	UPDATE文
DELETE	DELETE文
ALTER	ALTER TABLE文
REFERENCE	インデックスや外部キーの作成
ALL	上記すべての権限

■ 権限の委譲

ユーザに権限を与える権限はオーナーの特権であるが（もちろんDBAも可能）、ユーザに委譲することも可能だ。

```
GRANT SELECT ON other_user.some_table TO user1
WITH GRANT OPTION;
```

このようにWITH GRANT OPTIONを付加すれば、user1は、自分に与えられたother_user.some_tableテーブルに対するSELECT権限を別のユーザに付与できる。

■ DBA権限の付与

ユーザにDBA権限を与えるには、GRANT DBA文を用いる。

```
GRANT DBA TO user1;
```

■ デフォルトユーザのパスワード変更

前述したように、SQL Anywhereではデフォルトで DBA権限を持つ「DBA」ユーザが、「SQL」というパスワードで作成されている。データベースを運用する際は、セキュリティの面から、DBA権限を持つ別のユーザを作成し、「DBA」ユーザを削除（もしくは接続不可能）することを推奨する。少なくとも、次のようにしてパスワードを変更しておくとよい。

```
GRANT CONNECT TO dba IDENTIFIED BY new_password;
```

■ ストアドプロシージャの実行権限の付与

ストアドプロシージャの実行権限を与えるには、GRANT EXECUTE文を用いる。

```
GRANT EXECUTE ON Calculate_Report TO user1;
```

2.6.1 グループの作成

グループの機能は、GROUP権限として実装されている。GROUP権限を持つユーザは、ほかのユーザの親となることができ、「グループ」として振る舞う。グループ自体に権限を与えれば、そのグループに所属するユーザもその権限を与えられたことになる。

```
GRANT CONNECT TO group1;
GRANT GROUP TO group1;
```

GRANT CONNECTのときにパスワードを指定していないので、このグループ（ユーザ）名で接続はできない。

グループにユーザを追加するには、以下のように実行する。

```
GRANT MEMBERSHIP IN GROUP group1 TO user1;
```

2.7 主キーの生成

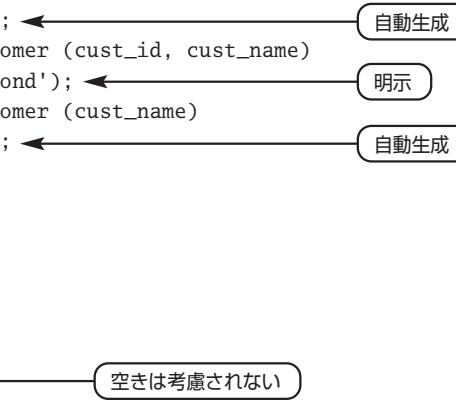
主キーとして使うデータは、顧客名や商品名など、アプリケーションロジックとして決まる場合もあるが、データベース上で一意の値を自動生成したい場合もあるだろう。排他制御の点から、アプリケーション側で一意の値を作るのは面倒であるが、SQL Anywhereの機能を使えば簡単だ。SQL Anywhereには、① 連番を生成するAUTOINCREMENT機能、② 一意の値を生成するUUID（Universally Unique Identifier）機能がある。

2.7.1 AUTOINCREMENT

カラムのデフォルト値としてAUTOINCREMENTを指定すると、正整数の連番が自動的に生成される。カラムに値を明示して行を追加すればその値が使用され、そうでなければ、一意の値が自動生成される。生成される値は、そのカラムで使用されている最大の正整数（なければゼロ）に1を足した値となる。なお、数字の並びに空きがあったとしても、それが考慮されることはない。

```
CREATE TABLE customer (
   cust_id    INTEGER NOT NULL DEFAULT AUTOINCREMENT,
   cust_name  VARCHAR(256),
   PRIMARY KEY ( cust_id )
);
INSERT INTO customer (cust_name)
VALUES ('first');
INSERT INTO customer (cust_id, cust_name)
VALUES (10, 'second');
INSERT INTO customer (cust_name)
VALUES ('third');
commit;
SELECT *
  FROM customer;

実行結果
1,'first'
10,'second'
11,'third'
```



■ 自動生成された値の取得

使用中のコネクション内で直近に自動生成された値は、グローバル変数の@@identityで取得できる。

```
SELECT @@identity;

実行結果
11
```

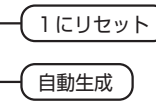
■ 自動生成される値のリセット

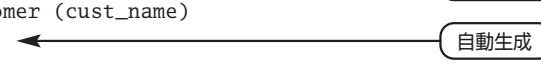
次回に自動生成される値を、sa_reset_identityプロシージャで設定できる。ここで設定された値に1を足したものが次に自動生成される。ただし、重複する値を避けて自動生成されることはないので、重複した場合は追加時にエラーとなる。

なお、TRUNCATE TABLE文で全行削除した場合は、ゼロにリセットされて、次回は1が生成される。

```
DELETE customer
 WHERE cust_id >= 10;
CALL sa_reset_identity('customer', 'dba', 1);
INSERT INTO customer (cust_name)
VALUES ('forth');
commit;
SELECT *
  FROM customer;

実行結果
1,'first'
2,'forth'
```





2.7.2 UUID

UUID（Universally Unique Identifier）はGUID（Globally Unique Identifier）とも呼ばれ、実行プロセスやマシンによらない一意の値（16バイト）を生成する機能である。そのため、Mobile Link同期環境など、データベースが分散するときの主キーとして便利だ。

SQL Anywhereでは、NEWID関数でUUIDを生成でき、その値を文字列と相互

変換する関数（UUIDTOSTRやSTRTOUUID）も用意されている。

```
CREATE TABLE orders (
  order_id    UNIQUEIDENTIFIER NOT NULL DEFAULT newid(),
  order_date DATE,
  PRIMARY KEY (order_id)
);
INSERT INTO orders (order_date)
VALUES ('2005-1-1');
commit;
SELECT UUIDTOSTR(order_id), order_date
  FROM orders;

実行結果
'36a18c0a-d17d-4f79-b5cc-04432b25c1c8','2005-01-01'
```

2.8 スケジュールとイベント

SQL Anywhereには、スケジュールとイベントと呼ばれる機能があり、指定されたタイミングで特定の処理を実行させることができる。スケジュール機能は、OSの時刻をもとに実行される。一方、イベント機能は、ユーザからの接続があったなどのシステムイベントをもとに実行される。なお、SQL Anywhere内部では、スケジュールもイベントの一種とみなされているので、混乱のない限り、スケジュールにも「イベント」という語を使用して説明する。

スケジュールは、定まった時刻や間隔で処理を実行できるので、データベースのバックアップやインデックスの再構築などデータベースのメンテナンス処理に都合がよい。一方、イベントは、ユーザからの接続を調べたり、ディスクの空き容量を調べたりできるので、管理者へ警告メールを送信するなどといったデータベースの監視に適している。

2.8.1 スケジュール

処理が実行される時刻を時間や曜日で指定できる。毎週月曜日であるとか、3時間ごとといった指定も可能だ。たとえば、毎日深夜1時にバックアップを実行するには、次のようにイベントを作成する。

```
CREATE EVENT IncrementalBackup
SCHEDULE
   START TIME '1:00 AM' EVERY 24 HOURS
HANDLER
BEGIN
   BACKUP DATABASE DIRECTORY 'c:¥¥backup'
   TRANSACTION LOG ONLY
   TRANSACTION LOG RENAME MATCH
END
```

CREATE EVENT文でイベント名を定義し、SCHEDULE部で時刻などを指定する。続くHANDLER部以下に処理のスクリプトを書く。処理は、SQLスクリプトとして書ければ、どのような処理であってもよい（バックアップについては第5章「バックアップとリカバリ」で解説するので、ここではスクリプトの詳細には触れない）。

SCHEDULE部では、上記の例以外にも、次のような時刻指定が可能だ。

```
BETWEEN '8:00AM' AND '6:00PM'
EVERY 1 HOURS ON
   ('Monday','Tuesday','Wednesday','Thursday','Friday')
```

これは、平日の朝8時から夕方6時まで1時間ごとに実行することを意味している。

スケジュールでは次の点に注意する。

- スケジュールで指定された時刻にデータベースが動いていないと実行されない。
- 前の処理が終了してから次の処理が予約される。たとえば、処理に70分かかるスケジュールを1時間おきに実行されるように指定しても、実際には2時間おきに実行される。

2.8.2 イベント

イベントは、SQL Anywhereのシステムイベントがトリガとなって実行される。次の例では、トランザクションログファイルが10MBを超えると、バックアップを実行し、トランザクションログファイルを切り替える。

```
CREATE EVENT LogLimit
TYPE GrowLog
WHERE event_condition( 'LogSize' ) > 10
HANDLER
BEGIN
   BACKUP database
   DIRECTORY 'c:¥¥logs'
   TRANSACTION LOG ONLY
   TRANSACTION LOG RENAME MATCH
END
```

スケジュールと同様に、CREATE EVENT文でイベントを定義し、HANDLER部に処理を記述する。

スケジュールと異なる点は、SCHEDULE部で時刻を指定するのではなく、TYPEでイベント種別を指定し、必要ならWHEREで条件を指定する点だ。上記の例では、トランザクションログファイルのサイズが増加したことを意味する「GrowLog」イベントが起こったとき、ファイルサイズが10MBを超えたなら、処理が実行されることを意味している。

イベント種別には以下のものが用意されている。

表2.5：イベント種別

イベント種別	説明
BackupEnd	バックアップ実行後
DatabaseStart	データベース起動後
Connect	ユーザからの接続時
ConnectFailed	接続失敗時
Disconnect	ユーザの接続が切断されたとき
DBDiskSpace	DB ファイルがあるディスクの空き容量をポーリング 指定可能条件：DBFreePercent、DBFreeSpace
LogDiskSpac	トランザクションログファイルがあるディスクの空き容量をポーリング 指定可能条件：LogFreePercent、LogFreeSpace
TempDiskSpace	テンポラリファイルがあるディスクの空き容量をポーリング 指定可能条件：TempFreePercent、TempFreeSpace
GrowDB	データベースファイルのサイズが増えたとき 指定可能条件：DBSize
GrowLog	トランザクションログファイルのサイズが増えたとき 指定可能条件：LogSize
GrowTemp	テンポラリファイルのサイズが増えたとき 指定可能条件：TempSize
GlobalAutoIncrement	Global Auto Increment の数が消費されたとき 指定可能条件：RemainingValues
RAISERROR	SQL エラーが起こったとき 指定可能条件：ErrorNumber
ServerIdle	データベースサーバのアイドル状態をポーリング 指定可能条件：IdleTime

なお、すべてのイベント種別に対して、前回実行時からの間隔を指定する条件であるIntervalを指定できる。

2.9 Remote Data Access

SQL Anywhereでは、クライアントが現在接続しているデータベースから別のデータベースにアクセスできる。SQL Anywhereとほかのデータベースとを連携でき、データの移行（ほかのデータベースからSQL Anywhereへ、またはその逆）に都合がよい。

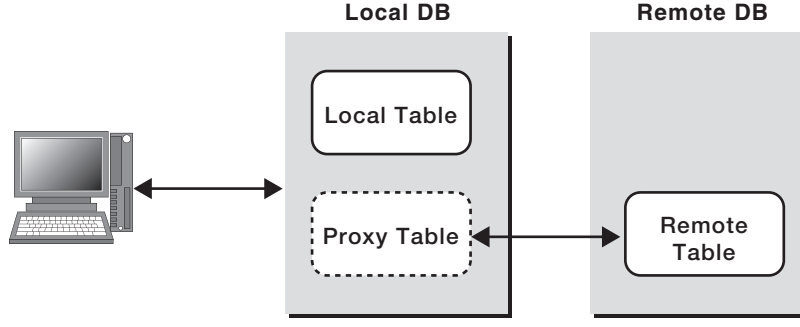


図2.12 : Local DBとRemote DBの概念図

クライアントが現在接続しているデータベースを「Local DB」と呼び、それ以外のデータベースを「Remote DB」と呼ぶことにする（図2.12）。また、Local DB内とRemote DB内の通常のテーブルを、それぞれ「Local Table」、「Remote Table」と呼ぶ。このとき、Local DB上でRemote Tableへのアクセスを可能にするのがRemote Data Accessと呼ばれる機能だ。

サポートするRemote DBは、SQL Anywhereに限らず、Sybase Adaptive Server Enterprise、Oracle、IBM DB2、Microsoft SQL Server/Excel/Access、そのほかODBC接続できるものである。また、Remote DBは、Local DBと同じマシン上で動作しているものでもよいし、別のマシン上であってもよい[7]。

7　複数のSQL Anywhereデータベースが同じデータベースサーバで管理されていたとしても、それらは別々で、Remote DBとLocal DBの関係である。

SQL Anywhereでは、Remote Tableに対してProxy Tableと呼ばれるテーブルをLocal DBに定義できる。クライアントからみれば、Proxy TableはLocal Tableのように振る舞うので、クライアントはあたかもRemote TableがLocal DBにあるかのようにそのデータにアクセスできる。Proxy Table自体がデータを保持することはないが、Proxy Tableに対するクエリがRemote Table用に書き換えられて再送信されるので、このような仕組みが実現できるのである。

Proxy Tableに対するアクセスは、読み書きともに可能だ。しかし、パフォーマンスはLocal Tableのアクセスよりも劣る。リモート通信が介在し、クエリの最適化が十分に行えないからだ。

実際に、Proxy Tableを作成するための手順を見ていこう。Sybase CentralからGUIでの操作も可能だが、以下ではSQL文でのやり方を説明する。なお、これから示す手順はすべてLocal DBに対する操作であり、Remote DBにあらかじめRemote Tableがあることを前提としている。

2.9.1 リモートサーバの定義

まず、CREATE SERVER文で、Remote DBをリモートサーバとして定義する。

```
CREATE SERVER remoteasa
CLASS 'ASAODBC'
USING 'remote_dsn'
```

remoteasaが、Local DB内でRemote DBを識別するための名前である。

CLASS句で、Remote DBの種別を指定する。指定可能なものは、以下の8つだ。製品名と接続方法とからなる文字列なので説明は不用だろう。

- ASAJDBC

- ASEJDBC
- ASAODBC
- ASEODBC
- DB2ODBC
- MSSODBC
- ORAODBC
- ODBC

USING句では、Remote DBへの接続情報を指定する。ただし、あらかじめ、Remote DBに対するODBCデータソースがremote_dsnという名前で定義されているものとする。

2.9.2 ログイン情報の設定

Remote DBにアクセスするにあたり、デフォルトでは、Local DBへのアクセスで用いたのと同じユーザID・パスワードが利用される。Local DBとRemote DBとでユーザID・パスワードが一致するならログイン情報の設定は不要だが、異なる場合は設定が必要だ。

```
CREATE EXTERNLOGIN fred
TO remoteasa
REMOTE LOGIN frederick
IDENTIFIED BY banana
```

上記の例では、Local DBのユーザfredがリモートサーバremoteasaにアクセスするときは、frederick/bananaというユーザ名・パスワードを使用することを指定している。

2.9.3 Proxy Tableの定義

いよいよProxy Tableを定義する。Remote DB上のemployeeテーブルに対するProxy Tableを、p_employeeという名前でLocal DB上に定義するには、以下のようにする。

```
CREATE EXISTING TABLE p_employee
AT 'remoteasa..DBA.employee'
```

AT句で、リモートサーバ名やテーブル名を指定する（「..」は表記ミスではないので注意）。ここでは、remoteasaサーバ上のDBAユーザのemployeeテーブルとなっている。構文の詳細はヘルプを参照していただきたい。

これで、employeeテーブルに対するProxy Tableがp_employeeとして定義され、p_employeeテーブルに対する読み書きや、ほかのテーブルとのJOINなどが可能となる。Proxy Table同士もJOIN可能だ。

2.9.4 ほかのデータベースからのデータ移行

Proxy Tableの機能を使えば、ほかのデータベース上のデータをSQL Anywhere上に容易に移行できる。そのためには、上記の手順をたどってProxy Tableを作成し、そこからSELECT文で抽出した結果をLocal Tableに挿入（INSERT）すればよい。しかし実は、これらの手順をまとめたsa_migrateプロシージャがあらかじめ用意されているので、それを利用するだけでデータの移行が可能だ。

CREATE SERVER文でリモートサーバを定義した後、sa_migrateプロシージャを実行する。

```
CALL sa_migrate( 'local_user', 'remote_server', NULL, 'remote_user',
NULL, 1, 1, 1 )
```

この文の意味は、「remote_server」リモートサーバ上の「remote_user」ユーザが所有するすべてのテーブルを、データを含めてローカルデータベースに移行し、「local_user」ユーザの所有とする、となる。

引数の最後の3つはフラグになっていて、① データを移行するかどうか、② 実行中に生成されたproxy tableを削除するかどうか、③ 外部キーを生成するかどうか、を指定できる。

2.10 暗号化

SQL Anywhereには、暗号化機能として、データベースファイルの暗号化と通信の暗号化とがある。

たとえば、データベースをモバイルデバイスに入れて持ち歩く際、デバイスを紛失したり盗難にあったりすることもある。この場合、データベースファイルをダンプすればデータを見ることができるため、貴重な情報が漏洩する恐れがある。しかし、データベースファイルを暗号化しておけば、ダンプしたデータを解読することはできず、また、暗号化キーがなければデータベースを起動できないのでセキュリティが高まる。

一方、ネットワーク経由でデータベース通信をする際、通信データが盗聴される危険があるが、通信を暗号化しておけばそのような恐れはない。

2.10.1 データベースファイルの暗号化

AES（Advanced Encryption Standard）アルゴリズムを用いてデータベースファイルとトランザクションログファイルとを暗号化する。暗号化オプションはデータベース作成時に指定するので、既存のデータベースを暗号化する場合はデータベース

を再構築しなければならない（再構築に関しては「2.11 データベースファイルの再構築」で説明する）。暗号化するにはキーとなる文字列が必要となる。dbinitコマンドの-ekオプションで文字列として指定する。もしくは、-epオプションを指定すると、キー入力ダイアログが現れるので、そこに入力する。

```
$ dbinit -ek "0kZ2o56AK#" "encrypted.db"
```

暗号化されたデータベースを起動するには、同じキーが必要になる。

```
$ dbeng9 encrypted.db -ek "0kZ2o56AK#"
```

これで、データは暗号化されてファイルに記録される。

2.10.2 通信の暗号化

ODBC、OLE DB、Embedded SQLによるクライアント・サーバ間のTCP/IP通信を暗号化できる。ただし、TDS（Tabular Data Stream。jConnect、Sybase Central、Interactive SQLで利用できる）やOpen Clientによる通信は暗号化できない。暗号アルゴリズムはRSAと楕円曲線暗号から選択する[8]。

■ サーバ側の設定

サーバ側では次のように指定する。

```
$ dbsrv9 -x tcpip -ec RSA_TLS asademo.db
```

8 暗号化オプションは別ライセンスとなっている。SQL Anywhereのインストール時に、対応するライセンスキーを登録することで、暗号化オプションが利用可能となる。

-ecオプションでは、表2.6に示す5つの中から、接続を受け入れる通信方式をカンマで区切って列挙する。

表2.6：通信方式

通信方式	説明
none	暗号化されていない通信のみ可能
simple	簡易的な暗号のみ可能[9]
ECC_TLS	ECC暗号のみ可能
RSA_TLS	RSA暗号のみ可能
all	暗号化にかかわらず、すべて可能（デフォルト）

前出の例では、RSA暗号オプションだけが指定されているので、すべての通信がRSA暗号化される。後述するように、クライアント側でも接続する通信方式を指定できるので、クライアント側の設定にRSA_TLSが含まれていなければ、接続が拒否される。

なお、ここでの指定にかかわらず、TDSやOpen Clientによる通信は受け入れられ、暗号化されずにやり取りされる。

■ クライアント側の設定

Encryption接続パラメータで、利用したい通信方式を1つ指定する。サーバ側の設定で説明したものと同じ5つの値がある。

```
"ENG=asademo; LINKS=tcpip; Encryption=RSA_TLS"
```

クライアント側の設定とサーバ側の設定を照らし合わせ、接続可能な方式がある

9　「簡易的な暗号」とは、カモフラージュ程度の暗号を意味し、素人には解読されない程度のもの。きちんとしたセキュリティを保つには、強度な暗号であるRSAやCerticomを利用する。

場合のみ接続が確立する。なお、サーバ証明書を用いて、サーバを認証することも可能だ。

2.11 データベースファイルの再構築

データベースファイルの再構築とは、既存のデータベースをテキストファイルにダンプし、それを新しいデータベースに適用し、データベースファイルを作り直すことである。再構築のメリットとして、主に次の点が挙げられる。

1. データベースファイル形式の更新
   SQL Anywhereのデータベースサーバは、過去のバージョンで作成されたデータベースファイルも管理可能だ。しかし、新機能の中にはデータベースファイルの構造に依存するものがあるため、新しいファイル形式でないと有効にならない機能もある。したがって、SQL Anywhereをアップグレード（特にメジャーバージョンアップ）した際は、データベースファイルを再構築するのが原則だ。
2. データベースファイルのサイズの縮小
   SQL Anywhereでは、削除（DELETE）されたデータが占めていた領域は再利用されるものの、物理的に削除されることはないので、データベースファイルが小さくなることはない。しかし、再構築により、新しいデータベースではそのような領域が解消される。
3. データベースパラメータの変更（ページサイズや暗号化など）
   ページサイズを指定できるのはデータベース作成時だけであり、既存のデータベースのページサイズを変更することはできない。しかし、再構築時に、新し

いデータベースを異なるページサイズで作成すれば、ページサイズの“変更”が可能だ。このように、データベース作成時のみに指定可能なパラメータは、再構築時に変更する。データベースファイルの暗号化も同様だ。

再構築している間はデータベースを停止する必要があるため、頻繁に行うわけにはいかないが（データベースのデータ量や利用のしかたに依存するが）、上記のような問題はまれであろうから、頻繁に再構築する必要もない。必要となったときに実行すればよい。

再構築の手順は、以下のようにunloadとreloadの2つの過程からなる。

1. unloadでは、既存のデータベースのスキーマやデータをテキストファイルにダンプする。
2. reloadでは、ダンプされた情報を新しいデータベースに適用する。

これらの作業は、Sybase CentralからGUIで操作することも可能だが、ここではコマンドラインによる手順を紹介する。

2.11.1 既存データベースのunload

既存データベースをダンプするには、dbunloadコマンドを用いる。

```
$ dbunload -c "dsn=ASA 9.0 Sample" c:¥unload
```

指定されたデータベースに接続し、引数のディレクトリにダンプされたファイルを生成する。生成されるファイルには2種類あり、①reload.sqlというreload用のSQL文が書かれたファイルと②datの拡張子を持つデータファイルからなる。unload終了後、古いデータベースは停止させてかまわない。

2.11.2 新しいデータベースの作成

reload先となる新しいデータベースを作成する。データベースの作成時でしか変更できない項目は、ここで設定する。

```
$ dbinit -p 8192 new_asademo.db
```

2.11.3 データのreload

新しいデータベースを起動し、unload時に作成されたreload.sqlを新しいデータベースに対して実行する。

```
$ dbeng9 -n newasademo new_asademo.db
$ dbisql -c "eng=new_asademo;uid=dba;pwd=sql" reload.sql
```

2.11.4 その他のテクニック

以上が再構築の一連の手順であるが、実は、これらを自動化する便利なオプションがdbunloadコマンドにある。

-an

前ページ以降で説明した作業と同じことをdbunloadコマンドの-anオプションで実行できる（ページサイズの指定は後述）。-anで指定された新しいデータベースを自動作成し、さらにreloadも実行される。ダンプファイルは自動削除されるので残らない。

```
$ dbunload -c "dsn=ASA 9.0 Sample" -an new_asademo.db
```

-ar

-arオプションでは、“同じ”データベースに対して再構築できる。“同じ”の意味は、物理的に同じファイルという意味ではなくて、新規に作成されたデータベースが既存のデータベースと置換されるという意味である。すなわち、新データベースへのreloadが完了した後、新データベースは停止し、旧データベースが削除されて、新データベースのファイル名が書き換わる。

```
$ dbunload -c "dsn=ASA 9.0 Sample" -ar
```

-ap

-anや-arオプションと共に使われ、新しいデータベースのページサイズを指定できる。

同期に関係するとき

Mobile Link同期クライアントは、トランザクションログを用いて同期するため、再構築の前後でトランザクションログを維持する必要がある。それには、dbunloadコマンドの-arオプションのあとにディレクトリを指定する。そうすると、同期に必要な部分のトランザクションログが指定されたディレクトリに保存されるので、同期クライアントの引数にこのディレクトリを指定すれば、同期が継続できる。

2.12 SQL Anywhereのアップデート

SQL Anywhereのアップデートは小さな修正から順に、以下の3つがある。

- EBF（Express Bug Fix）によるバグフィックス

- アップグレードパッチによるマイナーバージョンアップ
- メジャーバージョンアップ

EBFは、主にバグの修正で、ビルドナンバーが更新される。修正履歴がWebサイトに掲示されるので、必要があればインストールする。EBFファイルには、同一マイナーバージョン内における過去の修正がすべて含まれるので、最新のものを1つインストールすればよい。EBFファイルは、アイエニウェア・ソリューションズ社のWebサイトからダウンロードでき、インストーラを使ってアップデートする[10]。

アップグレードパッチではマイナーバージョンが更新され、バグの修正だけでなく、新機能も追加される。パッチファイルは、アイエニウェア・ソリューションズ社のサイトからダウンロードでき、インストーラを使ってアップグレードする。新機能を利用するには、「2.11　データベースファイルの再構築」で説明した方法でデータベースファイルを再構築したほうがよい場合がある。

メジャーバージョンアップは、大幅な機能追加で、新製品に位置づけられる。既存のデータベースは、原則、データベースファイルを再構築しなければならない。

■ バージョンの確認

現在インストールされているSQL Anywhereのバージョンは、以下のコマンドで確認できる。

```
$ dbeng9 -v
```

また、データベースサーバの起動ウィンドウなどにも、利用しているバージョンが表示される。バージョンは「9.0.2.2451」のように記述され、次のような意味である。

メジャーバージョン.0.マイナーバージョン.ビルドナンバー

10　http://www.ianywhere.jp/。ただし、EBFの取得は有償。

2.13 サイレントインストール

SQL Anywhereをインストールする際、通常、CD-ROMからウィザードを起動して、オプションを選択しながらインストールする。1台のマシンにインストールするならこれでよいが、多数のマシンにインストールしたり、アプリケーションに組み込んだりする場合には、インストールの始めから終わりまで人間がつきっきりで見ていなければならず面倒だ。そのようなときは、あらかじめ指定したインストールオプションでインストールする「サイレントインストール」機能がある。

まず、設定オプションのマスターを作るため、適当なマシンにSQL Anywhereをインストールする。このとき、CD-ROM内のsetup.exeを-rオプション付きで起動する。

```
$ setup.exe -r
```

ウィザードに従ってインストールすると、設定したオプションが*.issという応答ファイルとして作成される。

次に、実際にインストールしたいマシンにインストールする。作成された応答ファイルをコピーしておき（たとえばsetup.iss）、CD-ROM内のsetup.exeを次のように起動する。

```
$ setup.exe -s f1"setup.iss"
```

これで、記録された設定どおりに自動でインストールされる。

2.14 各種コマンド

表2.7にSQL Anywhereにおけるデータベース管理コマンド名とその用途を簡単に記した。コマンドオプションなどの詳細はマニュアルを参照していただきたい。

表2.7 : SQL Anywhereのデータベース管理コマンド

コマンド名	説明
dbbackup	データベースをバックアップする
dbcollat	文字の照合（文字の並び順を指定したもの）をファイルに書き出す。照合をカスタマイズする際に利用する
dbdsn	データソース名の管理
dbconsole	データベース接続をモニタリングするツールを起動する
dberase	データベースファイルを削除する
dbfhide	dbsrv9などのコマンドラインはファイルとして指定することもできるが、そのファイルを暗号化する
dbhist	データベースのヒストグラムをファイルに書き出す
dbinfo	データベース情報を表示する
dbinit	データベースファイルを作成する
dbisql	Interactive SQLを起動する
dblang	ログメッセージなどの言語を変える
dblic	ライセンス数を変える
dbltm	Sybase Replication Serverに参加する
dbtran	トランザクションログをSQLに変換する
dbping	データベースサーバへの接続を試みる
rebuild	データベースを再構築する。dbunloadやdbinitコマンドのラッパー
dblocate	ネットワーク上にあるデータベースサーバを表示する
dbsvc	データベースサーバのサービス登録を管理する

コマンド名	説明
dbspawn	データベースサーバをバックグラウンドで起動する
dbstop	データベースサーバを停止する
dblog	トランザクションログファイルを管理する
dbunload	データベースをアンロードする
dbupgrade	データベースのスキーマをアップグレードする
dbvalid	データベースを監査する

第3章 SQL Anywhereの内部動作

本章では、SQL Anywhereの内部動作を解説する。どのようにSQL Anywhereが動いているかを知らなくても利用に支障はないが、パフォーマンスチューニング（第4章）やバックアップ（第5章）の仕組みを考察する知識の土台となる。まず、データベースの基礎構造であるページを取り上げ、ページにデータを記録する様子を説明する。その後、インデックスやトランザクションログの働きをページの視点から見ていく。なお、ここで取り上げる内容はあくまでも概略であり、実際のSQL Anywhereの仕組みはもっと複雑であることをお断りしておく。

3.1 ページ

SQL Anywhereのデータベースファイル（*.db, *.logなど）は、固定長の「ページ」単位で構成される。1つのページに複数の行を記録できるが、異なるテーブルに属する行が1つのページに混在することはない。行を1つも持たない空のテーブルであっても、少なくとも1つのページが割り当てられる。テーブルに行を追加するとき、既存のページに空きがあればそこに記録し、空きがなければ新たにページを作成して（このときデータベースファイルのサイズが大きくなる）記録する。

データの読み書き（ディスクアクセス）は、必ずこのページ単位で行われる。ある行のデータをディスクから読むとき、その行の部分だけにアクセスするということはない。その行を含むページ全体をまずディスクから読み、次にメモリ上で該当する行を検索する。ディスクへの書き出しも同様で、メモリ上のページ内のある行が更新されてディスクに記録されるとき、その行だけでなく、ページ全体がディスクに書き出される。このように、ページはデータベースの物理的な基本要素である。

3.1.1 ページサイズ

ページサイズは、データベース作成時に指定する[1]。サイズの単位（バイト）としては、1,024・2,048・4,096・8,192・16,384・32,768の6つから選択できる。デフォルトは2,048バイトだ[2]。ページサイズはパフォーマンスに大きな影響を及ぼすことがあるので、極端に大きなサイズ（16Kや32K）や小さなサイズ（1K）の適用は慎重に行う。

1つのデータベースに複数のページサイズを割り当てることはできない。1つのデータベースに対して複数のデータベースファイルを割り当てることはできるが、ファイルごとにページサイズを変えるということはできない。

データベースサーバは複数のデータベースを管理できるので、それぞれのデータベースが別々のページサイズであってもよい。しかし、このようなページサイズの混在は推奨しない。なぜなら、データベースサーバのメモリ管理は単一のページサイズであるので（管理するデータベースの中で最大のページサイズが利用される）、異なるページサイズのデータベースが混在した場合、小さなデータベースのページがメモリ上では大きなページに1つ収まるため、メモリスペースに無駄が生じるからである。よって、データベースサーバが管理するデータベースのページサイズは統一されているのが望ましい。

3.1.2 ランダムアクセスとブロックI/O

ハードディスクドライブ（HDD）の性質上、1つのページへのアクセスにかかる時間は、ページサイズによらない。HDDへのアクセス時間を構成する要素はいろいろ

1　第2章でも述べたように、dbinitコマンドの-pオプションでページサイズを指定する。既存のデータベースのページサイズを変更することはできない。ページサイズを変えるには、unload/reloadを用いてデータベースを再構築する（第2章の「2.11　データベースファイルの再構築」を参照）。

2　ただし、一般には、4,096バイトをお勧めする。

あるが、ディスクのヘッドを目的のセクタに移動させるシーク時間が多くを占めるため、読み書きするデータ量が多少変わってもアクセス時間は変わらない。よって、アクセスするデータ量よりもアクセスする回数のほうが、ディスクI/Oのパフォーマンス上は重要となる。

複数のページにアクセスするとき、ページがディスク上に散在していれば、ページ数分のディスクI/Oが必要となる。これをランダムアクセスという。

一方、ページが連続して記録されていた場合、それらを一度にまとめて読む（または書く）ことができるので、1回のディスクI/Oで複数ページにアクセス可能だ。これをブロックI/Oと言う。

Column　I/O処理

データベースのパフォーマンスを考える際、データのI/O処理として、遅い順に、① ネットワーク、② ディスク、③ CPUの3つの部分がある。よって、ネットワーク処理を減らし、ディスクI/Oを減らすことが肝要となる。

3.1.3 ページに行が記録される様子

実際に、ページに行が記録される様子を見てみよう。まず、2,048バイトのページサイズを持つデータベースを作成する。

```
$ dbinit -p 2048 fragtest.db
```

次に、テスト用のテーブルを用意する。

```
CREATE TABLE fragtest (
  pk INT PRIMARY KEY,
  s1 CHAR(100),
```

```
  s2 CHAR(100)
);
```

このテーブルに、初期データとして1万件の行を追加（INSERT）する。このとき、pkカラムとs1カラム（全100文字＝100バイト）にはデータを入れ、s2カラムはNULLとする。

```
BEGIN
  DECLARE @i int;
  SET @i = 0;
  lp: LOOP
    INSERT INTO fragtest (pk, s1)
    VALUES (@i, repeat('x',100));
    SET @i = @i + 1;
    IF @i >= 10000 THEN LEAVE lp END IF;
  END LOOP;
  COMMIT;
END;
```

テーブルが使用しているページ数は、dbinfoコマンドを使って調べることができる。dbinfoコマンドの出力結果にはシステムテーブルを含めた全テーブルの情報が表示されるが、ここではfragtestテーブルの部分だけを抜粋した。

```
$ dbinfo -u -c "eng=fragtest;uid=dba;pwd=sql"

結果（fragtestの部分を抜粋）：
     Table   Table             Index            Percent  Name
      id     Pages    %used   Pages    %used   of File
     -----  --------  -----  --------  -----  -------  ----
       436       648  ( 84)        71  ( 91)     45    fragtest
           ========          ========
Total: 1024 table + 407 index + 153 free + 5 other = 1589 pages
```

実データを記録するテーブルページとして、fragtestテーブルは648ページ

（Table Pages）使用している。そのうち実際にデータが記録されている割合は84％である（空きが生じる理由は後述する）。また、インデックスのデータを記録するのに71ページ（Index Pages）使用している。また、fragtestテーブルの識別番号（Table ID）は436になっている。

1ページに含まれる行数を計算すると、以下のようになる。

10,000（行）/ 648（ページ）= 15.4（行／ページ）

つまり、1ページあたり平均およそ15行記録されている。

行を追加するとき、その行のサイズがページサイズよりも小さい場合、行は必ず1つのページに書かれる[3]。このとき、既存のページの空きスペースを利用しようとするが、空きスペースが足りなければ、新たにページを作成してそこに記録する。一方、追加される行のサイズが1つのページに収まらなければ、複数のページにまたがって記録される。

■ 行の削除

では、いま追加した1万行をすべて削除してみよう。

```
$ dbisql -c "eng=fragtest;uid=dba;pwd=sql" "TRUNCATE TABLE fragtest"
$ dbinfo -u -c "eng=fragtest;uid=dba;pwd=sql"

結果（fragtestの部分を抜粋）：
     Table   Table            Index           Percent  Name
      id     Pages    %used   Pages    %used  of File
     -----  --------  -----  --------  -----  -------  ----
       436         1  (  2)         1  (  1)      0    fragtest
           ========          ========
Total: 377 table + 337 index + 870 free + 5 other = 1589 pages
```

3　後述するように、その後の更新（UPDATE）で複数のページにまたがって記録される可能性はある。

fragtestテーブルのテーブルページが647ページ減り（Table Pages）、インデックスページが70ページ減っている（Index Pages）。しかし、データベースの合計ページサイズ1,589ページに変化はない。その代わり、647 + 70 = 717ページだけ空きページ（free pages）が増えている。

このように、SQL Anywhereでは、行が削除されて空となったページがあったとしても、そのページが物理的に削除されることはなく、データベースサイズは減らない。空となったページは、以降の追加および更新時に使い回される[4]。

■ ページの内部構造

ここでいったん、ページの内部構造に目を向けよう。ページは、ヘッダー部・データ部からなる。ヘッダー部には、このページが属するテーブルのIDやページ番号・前後のページの番号などが記載される。データ部には、行のデータが1つずつ記録される。行には、データそのものだけでなく、行を識別するためのページ番号と行番号の組が含まれる。

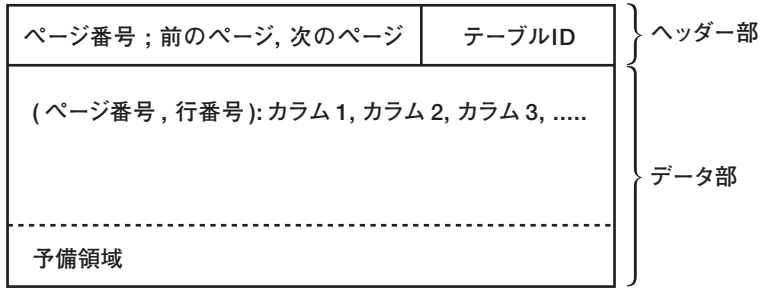


図3.1：ページの内部構造の概念図

ページに行が追加されるとき、ページのサイズがすべて消費されることはない。の

4　空きスペースを物理的に削除してデータベースのサイズを縮めるには、unload/reloadでデータベースを再構築する。詳細については、第2章の「2.11　データベースファイルの再構築」を参照。

ちのち、行が更新されてデータが増加することに備え、一定量のスペースを予備領域として残す。デフォルトの予備サイズは、200バイトだ[5]。

なお、ページサイズの大小にかかわらず、1つのページに記録可能な行数は最大255である。したがって、行のサイズが小さいにもかかわらず大きなページサイズを採用すると、ページの空間に無駄が生じるので注意したい。

1万行が追加されたあとのページの様子を図示すると、図3.2のようになる。ここではテーブル内の相対的なページ番号をPで表し、ページ内の相対的な行番号をRで表す。

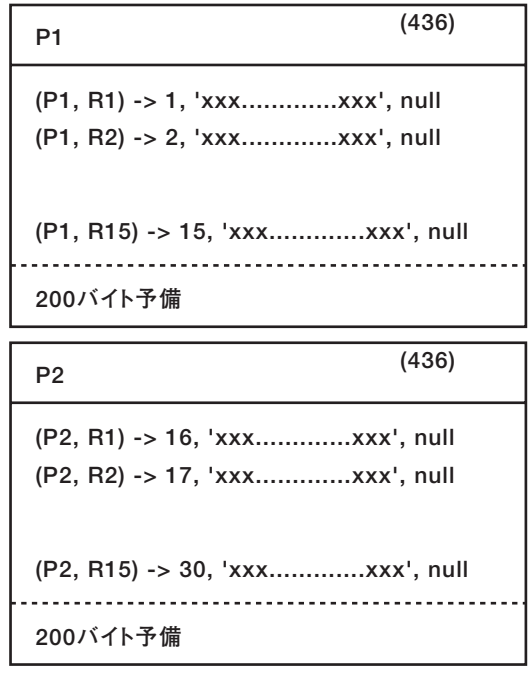


図3.2：行が追加されたあとのページの様子。ここでは最初の2ページを示したが、同様のものが648ページある

5　ただし、1Kのページサイズのときは100バイト。予備サイズは、CREATE TABLE文などのPCTFREEで変更可能である。

P1テーブルにはR1からR15まで15行が記録されている。同様に、P2テーブルにもR1からR15まで15行記録されている（行番号はページ内の相対番号である点に注意）。

実際に追加されたデータは、INTEGER型4バイトと文字列100バイトの計104バイトであるので、1つのページを占める正味のデータサイズは、以下の式で求めることができる。

104（バイト）× 15（行／ページ）= 1,560（バイト）

ヘッダー部分などページや行を管理するオーバーヘッドとなる量は、ページサイズが2,048 バイトで200バイトの予備サイズがあることを考慮すると、以下のようになる。

2048 −（1560 + 200）= 288（バイト）

したがって、300バイト程度と推測される。

3.1.4 行の更新

1万行が追加された直後の状態に話を戻そう。このとき、複数のページにまたがって記録されている行はなく、1つの行は必ず1つのページに記録されている。しかし、行を更新（UPDATE）して行のサイズが大きくなるとき、複数のページにまたがって記録されることがある（これを「行分割」と呼ぶ）。その様子を見ていこう。

まずは、分割度合いを計る「行セグメント（row segment）」という単位を導入する。行セグメントとは、1つのページ内において、1つの行の一部または全体のことを指す。図3.2を例にとると、P1内の行セグメントの数は15個だ。数え方は簡単で、「（ページ番号, 行番号）」の数を数えればよい。たとえば、pk = 1の行は、すべてのデータがP1に収まっているので（P1, R1）、1つの行セグメントからなる。このように、

行が1つのページ内に必ず収まっていれば、行の数と行セグメントの数は一致する。

テーブル内の行セグメントの個数は、sa_table_fragmentationプロシージャで確認できる。1万行が追加された直後のfragtestテーブルの状態を次に示す。

```
call sa_table_fragmentation('fragtest')
結果：
TableName rows  row_segments segs_per_row
----------------------------------------
fragtest  10000 10000        1.0
```

このように、予想どおり1万個の行セグメントを持つ。

■ ページの更新による行分割

さて、NULLだったカラム2（全100文字＝100バイト）を更新して、行のサイズを大きくするとどうなるのだろうか。200バイトが増加用の予備としてあるので、ページ内の15行のうち、はじめの2行に対する更新では予備領域が使用されるはずだ。そして、3行目に対する更新は、ページに内にスペースが足りないので、別のページに書かれるだろう。

実際にはじめの2行を更新してみよう[6]。

```
$ dbisql -c "eng=fragtest;uid=dba;pwd=sql" "UPDATE fragtest SET s2 =
repeat('y',100) WHERE pk = 1;"
$ dbisql -c "eng=fragtest;uid=dba;pwd=sql" "UPDATE fragtest SET s2 =
repeat('y',100) WHERE pk = 2;"
$ dbisql -c "eng=fragtest;uid=dba;pwd=sql"
"sa_table_fragmentation('fragtest')"
結果：
TableName rows  row_segments segs_per_row
----------------------------------------
fragtest  10000 10000        1.0
```

6　repeat関数は、文字列を指定された回数だけ連結した文字列を返す。

行セグメント（row_segments）に変化がないので、たしかに予備領域が使われているようだ。このときのページは図3.3のようになっている。

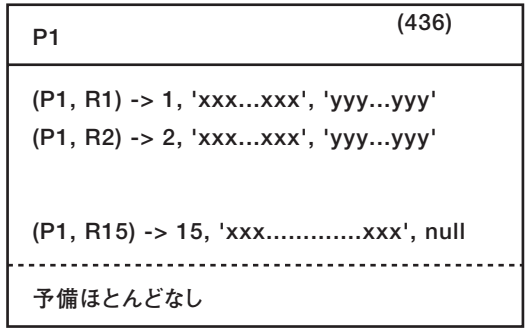


図3.3：はじめの2行を更新したところ

次に、3行目を更新してみる。

```
$ dbisql -c "eng=fragtest;uid=dba;pwd=sql" "UPDATE fragtest SET s2 =
repeat('y',100) WHERE pk = 3;"
$ dbisql -c "eng=fragtest;uid=dba;pwd=sql"
"sa_table_fragmentation('fragtest')"
結果：
TableName rows  row_segments segs_per_row
----------------------------------------
fragtest  10000 10001        1.0001
```

行セグメント（row_segments）が1つ増えているので、1つの行が2つの断片（セグメント）に分割され、別のページに記録されたことがわかる。このときのページの状態は、図3.4のようになっている。

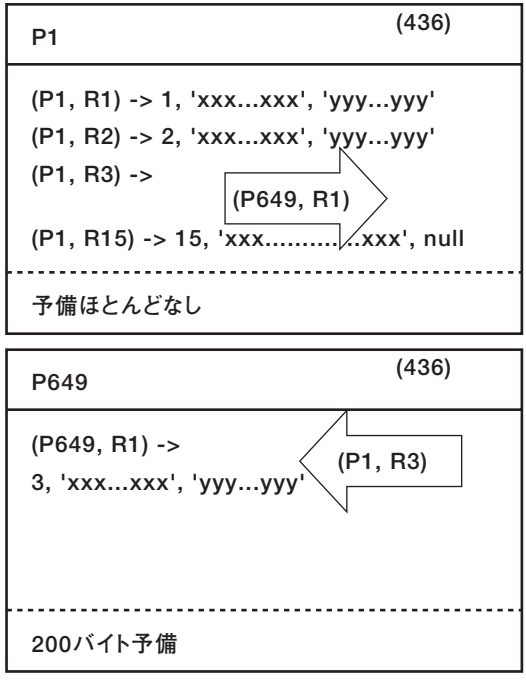


図3.4：3行目を更新し、行分割が起こったところ

更新時にスペースが足りなければ、スペースに余裕がある別のページに行の実データが書き込まれ（P649, R1）、元の場所（P1, R3）には移動先を示す情報のみが残る[7]。このように、SQL Anywhereでは、更新処理が行われても、行の先頭は行が最初に追加されたページにとどまり（P1, R3）、移動されることはない。pk＝3のデータを読むには、この行の先頭（P1, R3）にアクセスし、そこに飛び先が書かれていることを知り、（P649, R1）を読む。このように、行分割が起きるとディスクI/Oが増加しやすい。

行分割が発生した状態を、テーブルのフラグメンテーションと呼ぶ。テーブルの

7　ここでは、649番目の新しいページが作成されているが、既存のページに十分な空きがあればそこに書かれる。

フラグメンテーションを解消するには、REORGANIZE TABLE文を用いてテーブルを再構築する（詳細は第4章の「4.4 フラグメンテーション」で説明する）。fragtestテーブルを再構築したあとに行セグメントを確認すると、以下のようになっている

```
REORGANIZE TABLE  fragtest
確認：
call sa_table_fragmentation('fragtest')
結果：
TableName rows  row_segments segs_per_row
----------------------------------------
fragtest  10000 10000        1.0
```

たしかに、rowsとrow_segmentsの値は一致しており、テーブルのフラグメンテーションが解消されている。

3.1.5 拡張ページ

SQL Anywhereでは、文字列型やバイナリ型による大きなデータは、拡張ページ（extension page）と呼ばれるページに記録され、通常のページ（table page：テーブルページ）とは区別される。具体的には、255バイト目までのデータは通常のページに書かれるが、256バイト以降のデータは、拡張ページに記録される。

データベースを作成して、実験してみよう。まず、2,048バイトのページサイズを持つデータベースを作成する。

```
$ dbinit -p 2048 blobtest.db
```

次に、テスト用のテーブルを用意する。

```
CREATE TABLE blobtest (
    pk INT PRIMARY KEY,
```

```
    c1 CHAR(500),
    c2 CHAR(500),
    c3 CHAR(100)
);
```

データベース中の3つのカラムをすべてデータで満たす行を追加する。

```
INSERT INTO blobtest (pk, c1, c2, c3)
VALUES (1,
     repeat('x', 500),
     repeat('y', 500),
     repeat('z', 100));
```

このとき、ページは図3.5のようになっている。

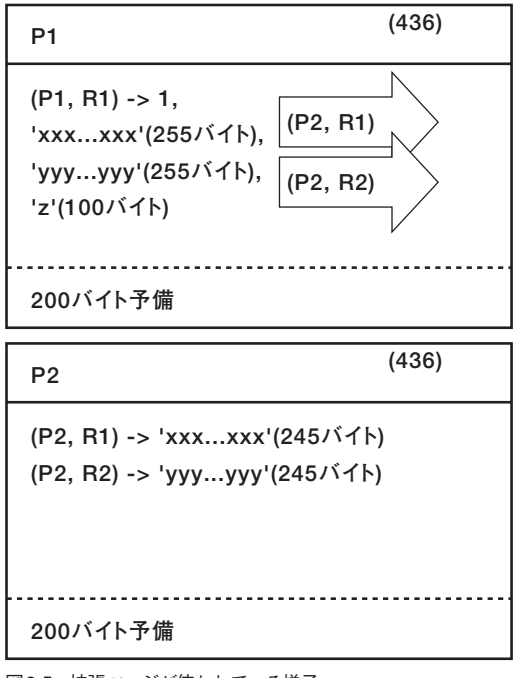


図3.5 : 拡張ページが使われている様子

カラムc1の255バイト目までのデータは（P1, R1）に書かれているが、（P2, R1）に続くという印があり、そこに256バイト目以降が記録されている（特に図示されていないが、ページP2は拡張ページである）。カラムc2についても同様だ。

テーブルが使用しているページ数を見てみると、以下のようになっている。

```
$ dbinfo -u -c "eng=blobtest;uid=dba;pwd=sql"
結果（fragtestの部分を抜粋）：
     Table   Table             Index            Percent  Name
      id     Pages    %used    Pages    %used   of File
     -----  --------  -----  --------  -----  -------  ----
       436          2  ( 29)         1  (  2)      0    blobtest
```

たしかに2ページ使用している。しかし、行セグメント（row_segments）の数は1である。

```
call sa_table_fragmentation('blobtest')
結果：
TableName rows  row_segments segs_per_row
----------------------------------------
blobtest  1     1            1.0
```

このように、sa_table_fragmentationの結果は、拡張ページについては考慮されず、テーブルページ（table page）についてのフラグメンテーションのみが示される。

拡張ページに関する情報は、sa_table_statsプロシージャで確認できる[8]。

8　sa_table_statsプロシージャは、本来はキャッシュの状態を確認するための内部機能なので、マニュアルには載っていない。

```
table_id creator table_name count table_page_count table_page_cached
-------------------------------------------------------------------
436      DBA     blobtest   1     1                1

table_page_reads ext_page_count ext_page_cached ext_page_reads
--------------------------------------------------------------
2                1              (NULL)          (NULL)
```

拡張ページ（ext_page_count）が1つ使用されていることがわかる。

3.1.6 まとめ

データベースファイルはページ単位に構成される。行のデータをページに記録する際、行分割によりテーブルのフラグメンテーションが起きることがある。sa_table_fragmentationプロシージャでテーブルのフラグメンテーションを確認する。ただし、文字列型やバイナリ型の大きなデータについては、拡張ページに記録されるので、sa_table_fragmentationプロシージャの結果には現れない。

3.2 インデックス

インデックスは検索効率を上げるのに有効な機能だが、万能というわけではない。インデックスの構造を理解し、その特徴を知ろう。SQL Anywhereのインデックスは、B+ツリーという構造を持つ。

3.2.1 B+ツリーの構造

連番の一意の値を持つカラム（たとえば主キー）に対するB+ツリーインデックスの様子を示す（図3.6）。

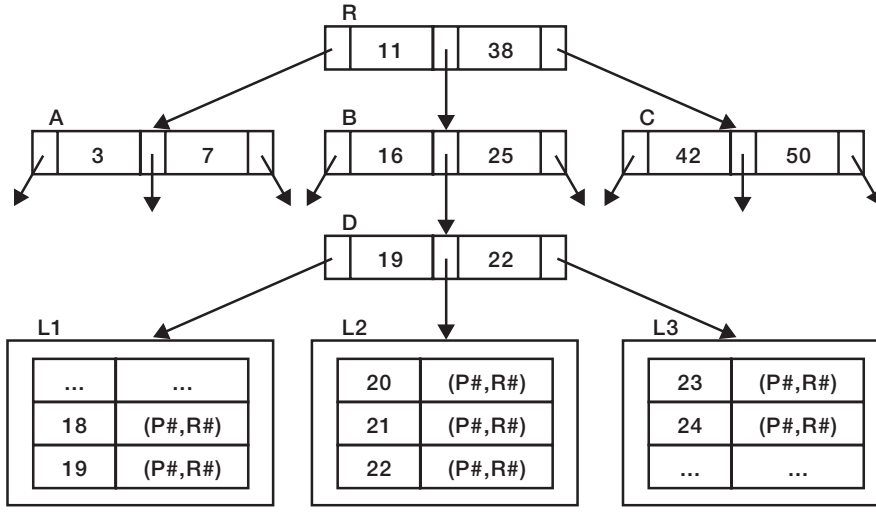


図3.6 : B+ ツリーインデックスの概念図

頂上にある1つのノード（ルートノード）から中間のブランチノードを経由して、最下層のノード（リーフノード）までたどれるようになっている。ルートノードは必ず1つだ。ブランチノードの階層はここでは2段になっているが、エントリーの数によって増減する。しかし、B+ ツリーでは、ツリーの一部分だけが深くなることはなく、全体の高さが均一になるという特徴がある。

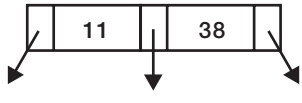


図3.7 : B+ ツリーのルートノード

1つのノードが1つのページを構成する。このページを、実際のデータが記録されるテーブルページに対比して、インデックスページと呼ぶ。インデックスページの個々のエントリーは、リーフノードとその他のノード（ルートノードとブランチノー

ド）では意味が異なる。

ルートノードとブランチノードは、キーの値と子ノードへのポインタからなる。キーは順番に並べられ、キー間にはポインタが位置する。ただし、左右の両端にはポインタが位置するので、キーの数より1つだけ多い数のポインタが存在する。ポインタが示す先のサブツリーには、左のキー値以上で右のキー値未満のエントリーを持つノードがある。ただし、左端のポインタは右のキー値未満を指し、右端のポインタは左のキー値以上を指す。

たとえば、図3.6のルートノード（R）では、キーが11以下のエントリーは、左のポインタの示す先（A）のサブツリー内に含まれ、キーが11以上38未満のエントリーは真ん中のポインタが示す先（B）に含まれ、キーが38以上のエントリーは右のポインタの先（C）に含まれる。ブランチノードでも同様だ。

一方、リーフノード（L1, L2, L3）のエントリーは、キー値と、実際の行へのポインタからなる。図3.6では、行のポインタは、ページ番号と行番号との組として表現されている。

インデックスを用いてキー値が21の行を検索する場合は、R→B→D→L2とたどることで、行へのポインタが得られる。このようにして、ルートノードからブランチノード・リーフノードを経て、行のデータに到達できる。

■ 高さとファンアウト

ルートノードからリーフノードへたどるときのパスの数をツリーの高さと呼ぶ。図3.6のB+ツリーの高さは3である。ノードが持つ子ノードの数の平均をファンアウト（fan-out）と呼ぶ。

高さがhでファンアウトがnであるツリーのリーフノード数は、n^hになる。実際のインデックスのファンアウト数は100程度であるので、高さが2であれば1万のリーフノードを持ち、高さが3であれば100万のリーフノードを持つ。よって、行数が増えてもツリーの高さはあまり増加せず、ほとんどの場合において、インデックスの高さ

は3ないし4で済む。

なお、sa_index_levelsシステムプロシージャで、インデックスの高さを調べられる。

■ イコール検索

ある値のキーをもつ行を検索するときのページI/Oの数を考えてみよう[9]。ルートノードのページからリーフノードのページを経てテーブルページに到達するので、ツリーの階層数に1（テーブルページ）加えた数のページI/Oが必要となる。テーブルに100万行あったとしても、ツリーの高さが3であれば、必ず5つのページI/Oでデータを取得できる[10]。

もし、インデックスがなければ、テーブルページをはじめから終わりまで検索する必要がある[11]。運良く最初のページに目的の行がある場合もあれば、運悪く最後のページまで探す場合もある。平均すれば、テーブルページ数の半分のページI/Oが検索に必要だ。よって、I/Oの数は行数に比例する。73ページの例であれば1万行に対して648ページ使用されているので、平均のI/O数は324となる。

このように、インデックスを使うと、行数によらないパフォーマンスでイコール検索を処理できる。

■ 範囲検索

次に、範囲検索時のページI/Oを考えてみよう。たとえば、21以上120以下のキーを持つ行（100件）を検索するとする。まず、21のキーを持つリーフノードを探す（R→B→D→L2）。図には表現されていないが、リーフノードは双方向リストになって

9 CPUの計算コストやブロックI/Oの効果などがあるので、実際のパフォーマンスが決まる過程はもっと複雑である。

10 インデックスページ（特にルートノード）はキャッシュされている確率が高いので、実際のページI/Oはこれよりも少ないだろう。

11 テーブルページはランダムに並べられていると想定している。

いるので、隣同士のリーフノードをたどることができる。L2→L3→…と、120のキーを持つリーフノードまでたどり、100件分の行のポインタを得る。

これらの行が同じテーブルページに含まれる可能性もあるが、最悪の場合、すべての行が別々のページに存在することもある。このときのページは、ディスク上で隣り合っているとは限らないため、ディスクにランダムアクセスするしかなく、100回のI/Oが必要になる。

このように、範囲検索では、検索対象となる行が増えるにつれてページI/Oも増える。検索対象が多いときは、インデックスを使わずに、テーブルページをすべて読むテーブルスキャンのほうが効率が良い。テーブルページは連続していることが多いので、ブロックI/Oが可能だからだ。インデックス検索のほうがテーブルスキャンよりも効率的になる分岐点は、10％程度の選択性が目安だと言われる。ただし、次のクラスタードインデックスではこの点が改善される。

■ クラスタードインデックス

クラスタードインデックスを作成すると、インデックスエントリーの並び順と同じような並びで、テーブルページに行が保存される。すなわち、リーフノード内の2つのインデックスエントリーの示す先は、同じテーブルページか近隣のものになる。よって、検索結果として多くのインデックスエントリーが該当する場合、それらが指し示す行を取得するためのページは重複するため、実際に必要なページ数は減り、ディスクI/Oも少なくて済む。さらに、ブロックI/Oの効果も期待できる。

クラスタードインデックスはテーブルページの順序に作用するので、1つのテーブルに対して複数のクラスタードインデックスを定義することはできない。

一般に、次のような検索でクラスタードインデックスが有効だ。

- WHERE句で、大小関係で取得する場合

  たとえば、年齢による選択で、「50歳以上」や「20歳以上30歳未満」など

- イコール検索で、多数の行が該当する場合
  たとえば、部門コードで従業員テーブルから検索するなど（同じ部門に多数の従業員が所属する）。これは、WHERE句だけでなく、JOINの結合条件として検索される場合もある
- ソート時
  ORDER BYなどでソートする際、インデックスの並びとテーブルページの並びとが対応するため、インデックスを使ってソートできる

このように、選択の幅が大きく、通常のインデックスが役に立たないような検索であっても、クラスタードインデックスならば有効であることが多い。ただし、行の追加や更新時には、より多くのコストを必要とするので、クラスタードインデックスが万能ということではない。

■ 行の追加

行の追加に伴い、インデックスにもエントリーが追加される。このとき、リーフノードはスペースに余裕を持って作成されるので、リーフノードだけの修正で済む場合もあるが、ツリー構造の変更が必要な場合もある。ツリー構造が変更される際、B+ツリーインデックスでは、ツリーが1つの方向に伸びるのではなく、全体の高さが揃うように変更される。

■ 行の削除

行の削除に伴い、インデックスのエントリーも削除される。このとき、SQL AnywhereのB+ツリーインデックスでは、削除されたエントリーが空きスペースになるだけで（後の行の追加時に再利用される）、ツリー構造が変更されることはない。多数の行が削除されると、リーフノードに空きスペースが多くなり、インデックスの性能が落ちることがある。そのようなときは第4章の「4.4 フラグメンテーション」で示

すように、インデックスを再構築する。

3.2.2 インデックスを利用可能な検索

WHERE句で検索条件を記述するとき、検索しようとするカラムにインデックスがあったとしても、SQL文の書き方によってはインデックスを利用できないことがある。

具体例を挙げて説明しよう。employeeテーブルのemp_idが主キーだとする。主キーなのでインデックスが自動的に作成されている。

● インデックス利用可

```
SELECT *
  FROM employee
 WHERE employee.emp_id = 123
```

この文では、インデックスが利用可能だ。しかし、次のSQL文のように<>（等価ではない）を用いた比較ではインデックスは利用されず、テーブルスキャンとなる。

● インデックス利用不可

```
SELECT *
  FROM employee
 WHERE employee.emp_id <> 123
```

sales_orderテーブルでは、order_dateがインデックスされている。しかし、次のSQL文のように、カラムの計算結果（YEAR関数：日付から年を返す）との比較ではインデックスは利用できない。

● インデックス利用不可

```
SELECT *
  FROM sales_order
 WHERE year(order_date)='2000'
```

インデックスを使うように書き直すと、以下のようになる。

● インデックス利用可

```
SELECT *
  FROM sales_order
 WHERE order_date > '1999-12-31'
   AND order_date < '2001-01-01'
```

演算子による差異を表3.1にまとめた。xとyは、テーブルの別々のカラムを表しており、それぞれインデックスを持つ。zは別のテーブルのカラムを示す（インデックスを持つ）。

表3.1：インデックスが利用可能な書き方

インデックス利用可能	不可能
x = 10	x <> 10
x IS NULL	x IS NOT NULL
x > 25	x = 4 OR y = 5
x = z	x = y
x IN (4,5,6)	x NOT IN (4,5,6)
x LIKE 'pat%'	x LIKE '%tern'
x = 20 - 2	x + 2 = 20

■ 複合インデックス

複数のカラムをまとめて、1つのインデックスを作ることができる。次のSQL文で

は、姓と名を組み合わせたインデックスを作成している。

```
CREATE INDEX lf_name
ON employee (emp_lname, emp_fname)
```

このようなインデックスは、同じ名を持つ従業員が存在し、名だけでは従業員を特定できない場合に有効だ。emp_lnameだけに単独のインデックスがある場合、重複するキーで検索して選択性が悪いと、インデックスを使わずにテーブルスキャンするほうが効率が良いことがある。例が悪いが、全体の30％が同じ名前「daigo」を持つ場合、WHERE emp_lname = 'daigo' で検索してもインデックスは利用されないだろう。しかし、複合インデックスにすると、1つ目のカラムemp_lnameで特定できなくても、さらにemp_fnameで絞ることが可能となる。WHERE emp_lname = 'daigo' and emp_fname = 'moriwaki'と検索した場合は、複合インデックスlf_nameが利用されるだろう。

もし、それぞれのカラムに別々のインデックスが作成されていた場合、WHERE emp_lname = 'daigo' and emp_fname = 'moriwaki' の検索では、どちらかのインデックスのみが利用可能だ。2つのインデックスが同時に利用されることはない。

複合インデックスは、1つのカラムに対するインデックスとしても利用できる。インデックスlf_nameは、emp_lnameカラムのインデックスとしても使えるので、別途emp_lnameにインデックスを作る必要はない。しかし、インデックスlf_nameは、emp_fnameのインデックスとしては利用できない。このように、複合インデックスは、作成するカラムの順序に意味を持つ。emp_fname単独での検索が必要な場合は、lf_nameの順序ではなく、次の順序でインデックスを作成すればよい。

```
CREATE INDEX fl_name
ON employee (emp_fname, emp_lname)
```

このように、2つのインデックスと1つの複合インデックスとでは意味が異なるので注意が必要だ。

3.2.3 インデックスの実験

実際に行を追加・削除しながら、インデックスの動きを確認しよう。まず、データベースを作成して起動する。

```
$ dbinit -p 2048 fragtest.db
$ dbeng9 fragtest.db
```

次に、テーブルを作成する。

```
CREATE TABLE fragtest (
pk INT PRIMARY KEY,
    s1 CHAR(100)
);
```

では、スクリプトを使って、このテーブルに10万件の行を追加してみよう。ただし、あとから行を追加しやすいように、pkには下一桁を0とする数値を順番に割り当てる。つまり、pkは0から100万番までの値を10おきに採番され、10万行が追加される。

```
BEGIN
  DECLARE @i int;
  DECLARE @factor int;
  SET @factor = 10;

  SET @i = 0;
  WHILE @i < 100000 * @factor LOOP
    INSERT INTO fragtest (pk, s1)
    VALUES (@i, 'x');
```

```
      SET @i = @i + @factor;
    END LOOP;
    COMMIT;
  END;
```

さらに、ちょっとおまじないをする（REORGANIZE TABLE文の詳細については第4章「パフォーマンスチューニング」で解説する）。

```
REORGANIZE TABLE fragtest PRIMARY KEY;
```

さて、準備が整ったので、インデックスページの様子を見てみよう。インデックスページ数を確認するには、sa_index_densityプロシージャを使用する。

```
call sa_index_density('fragtest')
実行結果：
TableName TableId IndexName IndexId IndexType LeafPages Density
-----------------------------------------------------------------
fragtest  436     fragtest  0       PKEY      646       0.995984
```

fragtestテーブルの主キーのインデックスは646リーフページ（LeafPages）から成り、エントリー密度（Density）は0.995984になっている。ほぼびっしり、ページにエントリーが詰まっていることになる。

リーフページあたりのエントリー数は、次の式で求めることができる。

100,000（行）÷ 646（ページ）= 154.799（エントリー／ページ）

この密度が0.995984にあたる。リーフページに記録できるエントリーの数は以下の式で求められる。

154.799 ÷ 0.995984 = 155.423

したがって、最大155個のエントリーを記録できることになる。

■ リーフページの分割

つまり、ページ内の空きスペースは、あと1行分のエントリーを記録できるかどうか程度しかない。それを超えるエントリーが追加されれば、リーフページが分割されるはずだ。余裕をみて、3行ほど追加してみる。

```
insert into fragtest values (1, 'y');
insert into fragtest values (2, 'y');
insert into fragtest values (3, 'y');
commit;
checkpoint;
call sa_index_density('fragtest')
実行結果：
TableName TableId IndexName IndexId IndexType LeafPages Density
-----------------------------------------------------------------
fragtest  436     fragtest  0       PKEY      647       0.994495
```

たしかに、リーフページ（LeafPages）が646から647へと1つ増え、密度も低くなった。

■ 激しいリーフページの分割

これをリーフページごとに繰り返していけば、すべてのリーフページを分割できるかもしれない。1リーフページあたりおよそ155のエントリーがあるので、155行ごとに数行追加していけばよいだろう（pkの番号で言えば、1550番ごとに数行となる）。スクリプトを書いて実行する[12]。

```
-- 先ほどの3行を削除して元に戻す
delete fragtest
```

12　余裕を持たせて、155行ごとに9行追加している。また、番号の若いほうから追加してしまうと、分割したページに追加することになり効果が見えにくいので、降順で追加している。

```
 where pk between 1 and 3;
commit;
-- 行の追加
BEGIN
  DECLARE @i int;
  DECLARE @j int;
  DECLARE @factor int;
  SET @factor = 10;

  SET @i = 100000 * @factor;
  WHILE @i > 0 * @factor LOOP
    SET @j = 1;
    WHILE @j <= 9 LOOP
      INSERT INTO fragtest (pk, s1)
      VALUES (@i + @j, 'x');
      SET @j = @j + 1
    END LOOP;
    SET @i = @i - 155 * @factor;
  END LOOP;
  COMMIT;
END;
CHECKPOINT;
-- 確認
call sa_index_density('fragtest')
実行結果：
TableName TableId IndexName IndexId IndexType LeafPages Density
-----------------------------------------------------------------
fragtest  436     fragtest  0       PKEY      1291      0.533786
```

リーフページは647から1291へとほぼ2倍になった。10万行に対しておよそ6,000行（6%）を追加しただけで、リーフページは倍になったことになる。

■ 行の削除

初期状態のデータベースに戻り、今度は、行を削除してみよう。1リーフページあたりおよそ155のエントリーがあるので、155行につき154行削除する。

```
BEGIN
```

```
  DECLARE @i int;
  DECLARE @j int;
  DECLARE @factor int;
  SET @factor = 10;

  SET @i = 0;
  WHILE @i < 100000 LOOP
    DELETE from fragtest
     WHERE pk between (@i + 1) * @factor and (@i + 154) * @factor;
    SET @i = @i + 155;
  END LOOP;
  COMMIT;
END;
CHECKPOINT;
-- 確認
call sa_index_density('fragtest')
実行結果
TableName TableId IndexName IndexId IndexType LeafPages Density
-----------------------------------------------------------------
fragtest  436     fragtest  0       PKEY      646       0.020017
```

ほぼすべての行を削除したにもかかわらず、リーフページ数に変化はない。SQL Anywhereでは、削除時にインデックスツリーの構造が変化することはない。削除されたスペースは、以後の行の追加などで再利用される。

■ インデックスの再構築

インデックスの保存効率が悪くなると、インデックスツリーの高さが必要以上に高くなる、エントリーがキャッシュされるメモリ効率が悪くなる、範囲検索で必要以上にインデックスページを読んでしまう、といった弊害がある。保存効率の悪くなったインデックスを改修するには、REORGANIZE TABLE文でインデックスの再構築を行う（詳細については第4章で解説する）。

3.3 トランザクションログ

トランザクションログとは、コミットされたデータをデータベースファイルに書き込む前に蓄えておく場所のことである。データの更新を、トランザクションログファイルとデータベースファイルとの2段階に分けることで、データ更新のパフォーマンスが上がり、システム障害時にコミット済みデータが消失するのを防ぐ。このように、トランザクションログはRDBMSの重要な機能の1つであるため、以下ではページの視点からその機構を説明する。前節まではページの内部がどのように使われているかについて見てきたが、本節ではページを最小要素として、それがどのようにファイルに記録されていくかを見ることにする。

トランザクションログの詳細を説明する前に、なぜトランザクションログが必要なのか説明しておこう。

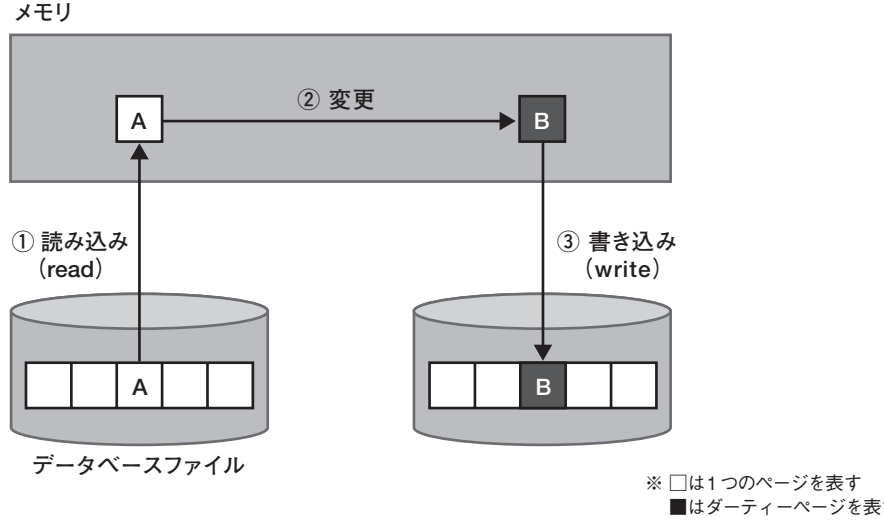


図3.8：トランザクションログがない場合の更新

まず、トランザクションログがないデータベースファイルだけの世界で、ある行のデータをAからBに更新する処理を考えてみる（図3.8）。まず、① データベースファイルから行Aを含むページをメモリ上に読み込み[13]、② データをBに更新する。これでデータは更新されたが、このままではデータベースサーバが停止したあとで更新されたデータを再び参照することはできない。データを永続化するために、③ データベースファイル上でそのページが存在した場所に、メモリ上のページ内容を上書きする。これで、AからBへの更新が完了する。

データベースファイル上のオリジナルのページからコピーされたページのうち、内容が書き換えられたページをダーティーページ（dirty page）と呼ぶ。この方式では、ダーティーページはすぐにデータベースファイルに書き出されるので、実質的にダーティーページは発生しない。

この方式の欠点は、ページ内のデータが1バイトでも変更されると、そのたびにデータベースファイルにページが書かれるので、更新の頻度が高くなる点だ。また、ファイルの途中にページを書き込むには、その位置を検索せねばならず、この検索は比較的コストが高い。よって、頻繁にデータを読み書きするには、このままではパフォーマンスが悪く、工夫がいる。そこで登場するのがトランザクションログだ。

■ トランザクションログを使った更新

更新されたデータを永続化する際、ダーティーページの変更差分をトランザクションログファイルに記録する（図3.9）。メモリ（キャッシュ）上のダーティーページはそのまま残り、このページのデータが参照されるときは、ダーティーページが使われる。

13　すでにメモリ上にあればそれを使う。

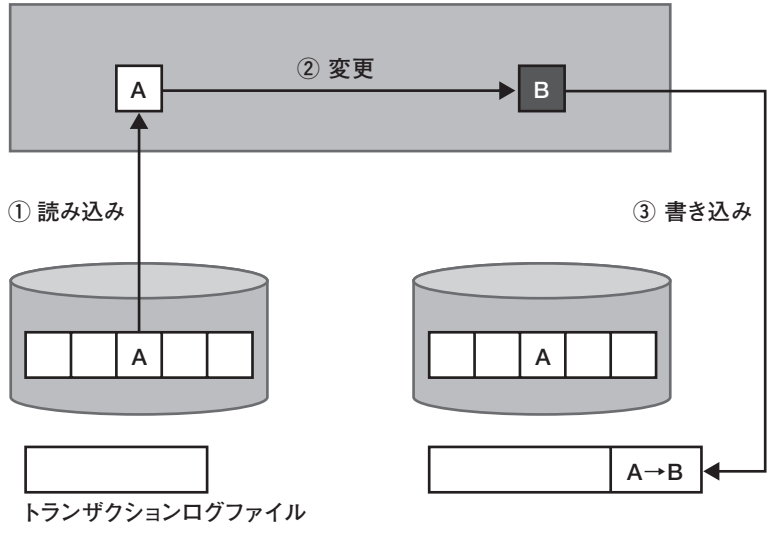


図3.9：トランザクションログを使った更新。トランザクションログに変更差分（A→B）が書き込まれ、ダーティーページはそのままメモリ上に残る

この方式の利点として、次の2つがある。

1. トランザクションログファイルへの書き出しは、常にその末尾に加えていくのでコストが低い
2. 元のページ内容全体ではなく、変更されたデータだけを記録できる[14]

こうしてトランザクションログを利用することで、効率的な永続化が可能となる。

しかし、データはデータベースファイルに保存されるべきという原則に変わりはない。データベースサーバを停止するときなど、トランザクションログの内容は、最終的にはデータベースファイルへ記録されなければならない。その仕組みがチェックポ

14　トランザクションログファイルへの書き込みもページ単位ではあるが、変更されたデータを複数まとめて1つのページとして記録することができる。テーブルに主キーがあれば変更部分だけ記録できるが、主キーがないときは行全体となる。この点からも、テーブルには主キーが定義されていることが望ましい。

イントと呼ばれる処理である。

3.3.1 チェックポイント

チェックポイントとは、キャッシュ上にあるすべてのダーティーページをデータベースファイルに書き込む処理のことである（図3.10）。チェックポイント前の最新のデータは、「キャッシュ上のダーティーページ」と「ダーティーページになっていないデータベースファイル上のページ」とを合わせたものだったが、チェックポイント後、データベースファイルは最新のデータ状態を反映したものになる。

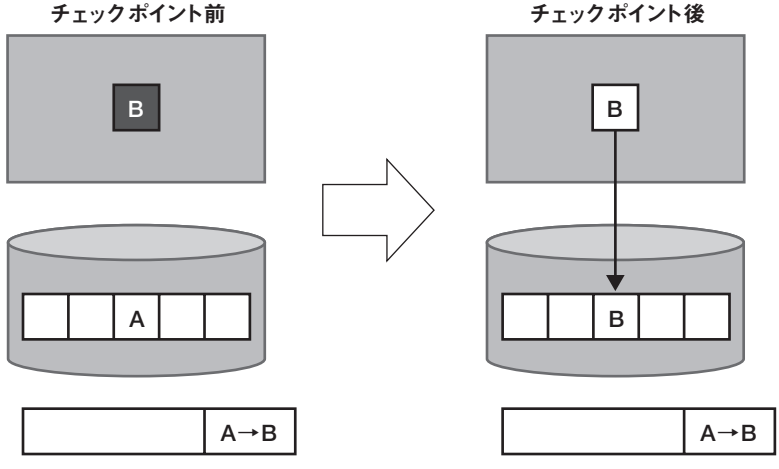


図3.10：チェックポイントでダーティーページをデータベースファイルに書き込む

書き込みのコストの小さいトランザクションログを使って頻繁に更新を記録し、コストの高いデータベースファイルへの書き込みはアイドルタイムなどを見計らってチェックポイントとしてバッチ的に行う。こうして、パフォーマンスを落とすことなく、データを記録し続けることができる。

■ トランザクションログの切り詰め

チェックポイントが発生してもトランザクションログファイルが削除されることはない。ここで説明したデータの永続化の文脈では、チェックポイント後のトランザクションログに使い道はないが、トランザクションログは変更データがすべて記録されている貴重なものなので、バックアップ時に切り詰める（削除する）のが基本だ。そのやり方については、第5章で説明する。

とはいえ、トランザクションログが大きくなりすぎないようにしたいこともあるだろう。データベースサーバの-mオプション（dbeng9/dbsrv9の引数）を指定すると、チェックポイントが発生するたびにトランザクションログが切り詰められるようになる。

3.3.2 まとめ

以上が、データベースファイルとトランザクションログファイルとの基本的な関係である。トランザクションログの主な役割は、次のとおり。

- 書き出しパフォーマンスの向上
- データの冗長化
  過去の変更記録がトランザクションログファイルとして記録されるので、データベースファイルが壊れたとしても、ある時点のデータベースファイルとその後のトランザクションログファイルとがあれば、データベースを復旧させることができる。
- レプリケーションで参照されるデータ
  Mobile Link同期クライアントなどでは、トランザクションログファイルのデータをもとにして、クライアント側で発生した更新を統合データベースに送る。

トランザクションログがファイルに書き出されるタイミングを以下にまとめておく。ダーティーページの更新差分は、メモリ上のバッファにまず蓄えられ、条件が満たされると、トランザクションログファイルに記録される。その条件とは、次の2つだ。

- コミット（COMMIT）が実行されたとき
- バッファが一杯になったとき

このことからわかるように、トランザクションログファイルにはコミット前のデータも記録されている可能性がある。この点は、次節で取り上げるチェックポイントログとロールバックログの存在に関係する。

次に、チェックポイントが実行されるタイミングをまとめておく。

- データベースのシャットダウンが実行されたとき
- 前回のチェックポイントから一定時間経過したとき（CHECKPOINT_TIMEオプションで指定可能）
- 予想されるリカバリ時間が閾値を超えたとき（RECOVERY_TIMEオプションで指定可能）
- データベースサーバが十分アイドルなとき
- CHECKPOINT文が実行されたとき
- トランザクションログファイルなしで運用されていて、コミット（COMMIT）されたとき

3.4 チェックポイントログとロールバックログ

トランザクションログにはコミット前のデータも記録されうるので、トランザクションの途中でデータベースが強制終了してしまうと、トランザクションログとデータベースファイルからだけではうまく復旧できない。それを補うのが、チェックポイントログとロールバックログである。

チェックポイントログとは、データベースファイルから読み出されるオリジナルのページをコピーしておくログで、リカバリ時に利用される。データベースファイルの末尾に記録されるが、チェックポイント時に削除される。

ロールバックログは、変更をロールバックするためのログで、データベースファイルに記録される。コネクションごとに用意され、コミットまたはロールバック時に削除される。

ページの読み出しからチェックポイントまでの一連の流れを、チェックポイントログとロールバックログとを含めて説明すると、以下のようになる。

1. データベースファイルからページを読み出し、同時に、そのコピーをチェックポイントログに記録する。
2. ページを変更する際、ロールバック可能なようにロールバックログにも記録する。
3. コミットすると、変更差分がトランザクションログに記録され、ロールバックログは削除される。
4. チェックポイントが実行されると、ダーティーページがデータベースファイルに記録され、チェックポイントログは削除される。

こうしておくことで、正常終了しなかった場合でもデータの整合性を保つことが

できる。システム障害からのリカバリ処理については、第5章で説明する。

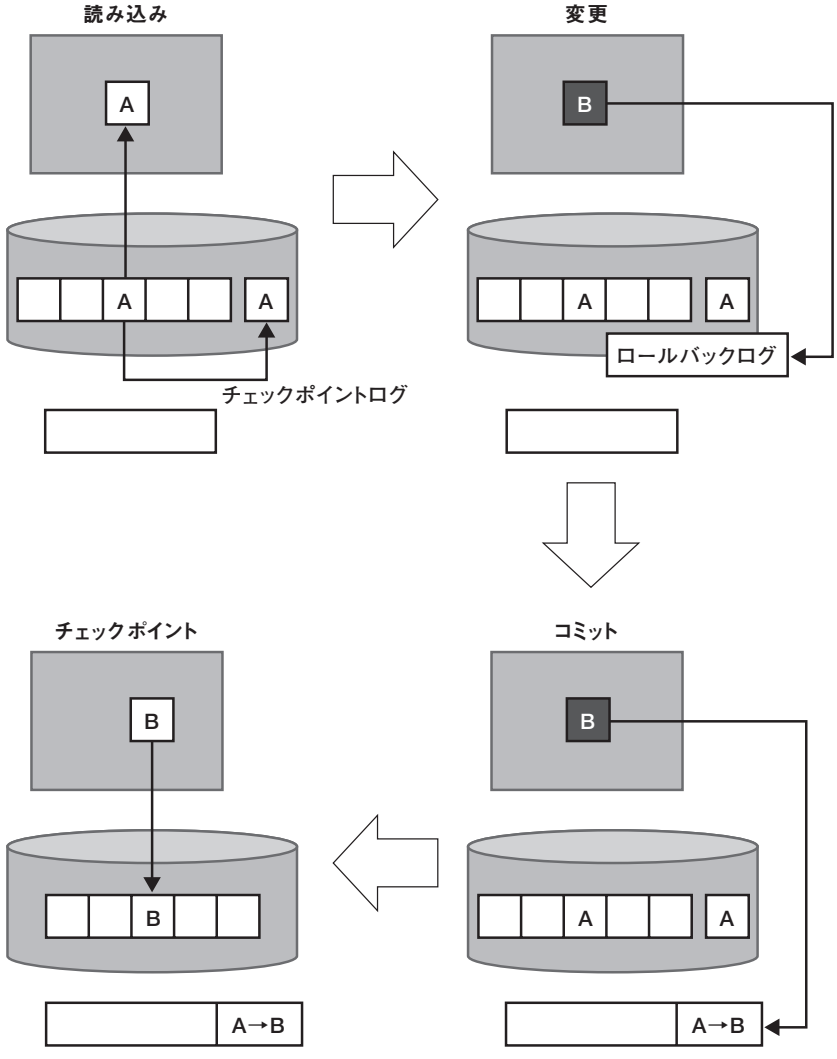


図3.10 : 読み込みからチェックポイントまでの流れ

第4章 パフォーマンスチューニング

パフォーマンスチューニングの鉄則は、むやみに対策せずに、きちんと原因究明することだ。最も悪い部分を見つけ出して改善するのが効果的である。全体に与える影響が小さな部分に対してチューニングしても効果は薄い。原因を突きとめて対策を考え、副作用や費用対効果をみながらチューニングするとよい。

パフォーマンスが悪くなったときは、発生状況を確認しよう。たとえば、次のようなことを確認してほしい。

- ある日突然遅くなったのか、もともと遅かったのか
- 常に遅いのか、ある時間帯だけ遅いのか
- システム全体が遅いのか、特定のマシンだけ遅いのか、特定の処理だけ遅いのか
- データ量に関係するのか
- アクセスしているユーザ数に関係するのか

パフォーマンスが悪くなる原因にはさまざまな要素が考えられる。ソフトウェア面では、アプリケーションやデータベース、一方、ハードウェア面では、CPUやメモリやHDDやネットワークなどが考えられる。これらを統合して原因の特定にあたる。その際、アプリケーションやハードウェアの知識が必要であり、OSなどに付属するツールを使うこともあるだろう。

すべての例を扱うことはできないので、本章ではデータベースの設定で解決可能ないくつかの問題を扱う。具体的には、パフォーマンスチューニングを助けるSQL Anywhereのツールを紹介し、効果的なインデックスの作成方法やフラグメンテーションの解消方法を取り上げる。最後に、物理設計のヒントなど、パフォーマンスが悪くなる前にあらかじめ気をつけておくべき点について説明する。

4.1 クエリの調査

SQL Anywhereには実行されたクエリのログをとる機能があり、それを分析して処理時間の長いクエリを抽出できる。アプリケーションのデータベース処理が遅くなった場合、遅いクエリを特定するのは一般に面倒な作業であるが、SQL Anywhereの機能を使えば容易に判別できる。それには、次のような手順を踏む（図4.1）。

1. ログを取得しながら、処理を実行する
2. そのログを解析し、問題となるクエリを発見する
3. そのクエリを分析し、アクセスプランを確認する

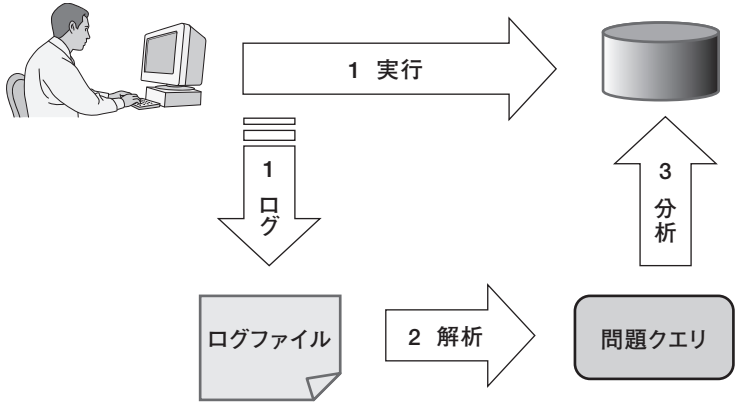


図4.1：クエリ調査の一連の流れ

クエリが遅くなる原因は不適切なアクセスプランによる実行とは限らないが、アクセスプランは最初に確認すべき項目である。アクセスプランの違いにより、処理時間に大きく差が生じる場合があるからだ。特にインデックスの役割は重要で、ど

のようにインデックスが使われているか（いないか）を確認し、必要ならばインデックスを作成する。次節では、インデックスの作成を支援するツールであるIndex Consultantを紹介する。

4.1.1 ログの採取

SQL Anywhereのロギング機能を使って、データベースが実行するクエリのログを取得する。このログには、どのユーザがどのようなクエリを実行したのかについて、その詳細が時系列に記録される。デフォルトの設定ではこのログ機能はオフなので、有効にしなければならない。それには、①request_level_loggingサーバオプションをオンにする方法と、②データベースサーバの起動オプションで指定する方法と、③Sybase CentralからGUIで設定する方法とがある。

① request_level_loggingサーバオプションをオンにする方法

サーバオプションを変更するには、sa_server_optionシステムプロシージャを使う。

```
call sa_server_option('request_level_logging', 'sql')
```

ログの冗長さは、sql、sql+hostvars、allの3段階で調整でき、順に冗長さが増す。

- sqlでは、実行されたクエリがすべて記録される[1]
- sql+hostvarsでは、さらに、Prepared Statementなどの内部変数の値も記録される

1　このログ機能は、データベースに送信されてきたクエリをすべて記録するので、パフォーマンスチューニング以外でも役立つ。たとえば、SQLをツールで自動生成しているような場合は、どのようなSQLが実際に実行されたのか、このログから確認できる。

- allでは、クエリだけでなく、接続イベントも含めたログが出力される

ログが冗長になるほど負荷がかかり、データベースのパフォーマンスに悪影響を与えやすいので注意する。

デフォルトではログはコンソールウィンドウに出力されるが、request_level_log_fileオプションでファイル名を指定すれば、ファイルに記録される[2]。後述するように、ログを解析するにはファイルでなければならないので、ファイルに出力されるようにする。

```
call sa_server_option('request_level_log_file', 'd:¥¥test.log')
```

② データベースサーバの起動オプションでオンにする方法

ログの冗長さは-zrオプションで指定し、出力先ファイルは-zoオプションで指定する。

```
$ dbeng9 test.db -zr sql -zo 'd:¥test.log'
```

③ Sybase Centralによる設定方法

Sybase Centralの左ペインで、データベースサーバのアイコンを右クリックし、プロパティを選択する。[サーバのプロパティ]ダイアログボックスが表示されたら、[オプション]タブを選択する。[要求レベルのロギング]グループボックスで、ログレベルと出力先ファイル名とを指定する。

2 このファイル名はデータベースサーバからのパスであり、リモート接続している場合、リモートマシン上のパスではない。

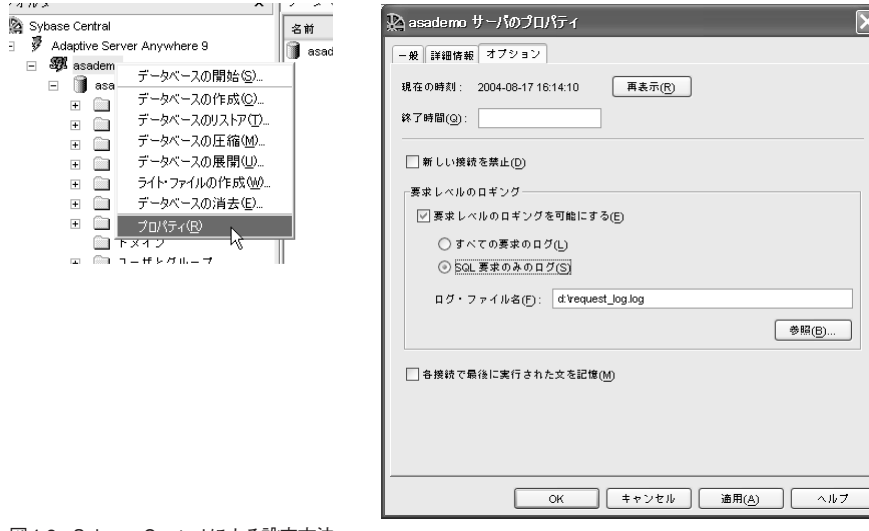


図4.2 : Sybase Centralによる設定方法

■ 処理の実行

ログの設定が済んだら、アプリケーションから懸案の処理を実行し、ログを採取する。

■ ログ取得の終了

ログの取得を終了するには、request_level_loggingプロパティをnoneに切り替える。

```
call sa_server_option('request_level_logging', 'none')
```

Sybase Centralから指示する場合は、[要求レベルのロギング] グループボックスで [要求レベルのロギングを可能にする] チェックボックスをオフに設定すればよい。

4.1.2 ログの解析：処理時間

「遅い」クエリには、大きく2種類ある。1つは、単独のクエリで処理時間の長いもの、もう1つは、繰り返し実行されているクエリで処理時間の合計が長いものだ。クエリ単独では短時間に処理されるとしても、繰り返し実行されれば全体に与える影響は大きいので、後者も考慮する必要がある。

さて、前節で採取したログを解析すれば、クエリごとの処理時間がわかる。幸い、SQL Anywhereには、ログ解析用のシステムプロシージャが2つ用意されている。

- sa_get_request_times ―― それぞれのクエリの処理時間を解析する
- sa_get_request_profile ―― クエリの種類ごとの処理時間を解析する

これらプロシージャの実行は、ログを出力する元になったデータベースでなくてもよいので、新たなデータベースを作成して、そこで実行することも可能だ。順番に使い方を見ていこう。

sa_get_request_timesプロシージャは、引数で指定されたログファイルを分析して、その結果を、satmp_request_timeというテンポラリテーブルに出力する。同じクエリは別々のものとして扱われる。

```
call sa_get_request_times('d¥¥test.log');
SELECT *
  FROM satmp_request_time
 ORDER BY millisecs DESC;
```

satmp_request_timeテーブルはテンポラリテーブルなので、接続を切ると情報が失われる。結果を保存するには別のテーブルに移すとよい。次のようなクエリを実行すると、request_timeというテーブルが新たに作られて、satmp_request_timeのスキーマやデータがそのままコピーされる。

```
SELECT * INTO request_time
  FROM satmp_request_time;
```

結果

	req_id	conn_id	conn_handle	stmt_num	millisecs	stmt_id	stmt
1	394	0	262642048	65603	1383411	61	SELECT C
2	306143	0	262642048	137044	16318	59	select HTt
3	490	0	262642048	131154	10480	37	INSERT IN
4	306149	0	262642048	202582	8154	59	select HTt
5	498	0	262642048	131155	7013	39	DELETE F
6	479	0	262642048	131152	5769	36	SELECT C
7	513	0	262642048	131158	1754	40	select MA
8	505	0	262642048	131156	1753	38	select CO
9	459	0	262642048	131149	1696	33	UPDATE S

図4.3 : sa_get_request_timesの解析結果。millisecsで処理時間、stmtでクエリがわかる

4.1.3 ログの解析：処理回数

次に、sa_get_request_profileプロシージャを使って、処理回数を考慮したクエリの処理時間を調べてみよう。sa_get_request_profileプロシージャは、引数で指定されたログファイルを解析して、その結果を、satmp_request_profileというテンポラリテーブルに出力する。同じクエリの実行は集計される。なお、sa_get_request_profileは内部でsa_get_request_timesを呼び出すので、sa_get_request_profileプロシージャを実行すると、satmp_request_timeテンポラリテーブルとsatmp_request_profileテンポラリテーブルとの両方に結果が残る。

```
call sa_get_request_profile('d¥¥test.log');
SELECT *
  FROM satmp_request_profile
 ORDER BY total_ms DESC;
```

結果

	stmt_id	uses	total_ms	avg_ms	max_ms	prefix
1	61	1	1383411	1383411	1383411	SELECT CO
2	50	754	589625	781	1470	update SumS
3	49	2765	81123	29	52	insert into #S
4	44	30991	69671	2	20	select GC
5	41	755	38644	51	60	select HTb
6	59	2	24472	12236	16318	select HTbl.D
7	37	1	10480	10480	10480	INSERT INTO
8	52	754	7593	10	19	update TaxS
9	39	1	7013	7013	7013	DELETE FRO
10	36	1	5769	5769	5769	SELECT CO
11	43	718	4634	6	22	select GC

図4.4 : sa_get_request_profileの解析結果。図4.3の表示とはかなり異なることがわかるだろう。usesが実行回数を示し、その合計処理時間はtotal_msとなっている

このように、処理回数や処理時間の合計・平均・最大値が集計されるので、sa_get_request_timesプロシージャでは見つけにくかった処理回数の多い問題クエリを発見できる。

satmp_request_profileテーブルもテンポラリテーブルなので、接続を切ると情報が失われる。結果を保存するには、前節と同様に、別のテーブルに移しておく。

```
SELECT * INTO request_profile
  FROM satmp_request_profile;
```

このようにして、問題となるクエリを特定する。次に、そのクエリがなぜ遅いのかを分析する。

Column　RememberLastStatementデータベースオプション

RememberLastStatementデータベースオプションをオンにすると、各データベース接続で最後に実行されたSQLを取得できる。

```
CALL sa_server_option( ' RememberLastStatement ', 'ON');
SELECT connection_property( 'LastStatement', conn_id )
```

4.1.4 クエリの分析：グラフィカルプラン

SQL Anywhereでは、クエリが実行されたアクセスプランをすぐに確認できる。クエリをInteractive SQLから実行してアクセスプランを確認し、結合（JOIN）処理やソート処理など内部処理に問題がないか調査する。

Interactive SQLで、クエリを実行した後［プラン］タブを選択すると、実行されたアクセスプランが表示される。表示されるプランの形式はいくつかあるので、設定で好みのものを選ぶ。［ツール］－［オプション］－［プラン］では、以下の4つから選択できる。

- グラフィカルなプラン
- 統計情報付きのグラフィカルなプラン
- 短いプラン
- 長いプラン

［短いプラン］と［長いプラン］では、簡略化されたテキスト形式でプランが表示される。ここでは、最も冗長な［統計情報付きのグラフィカルなプラン］に設定する（図4.5）。グラフィカルなプランでは、プランの構成要素がツリー状に図示され、把握しやすい。さらに、統計情報付きのグラフィカルなプランではオプティマイザの予測値と実際の値とを比較できるので、より詳細に分析できる。なお、［ファイル］－［名前を付けて保存］からXML形式で保存すると、プランのデータを保存できる。

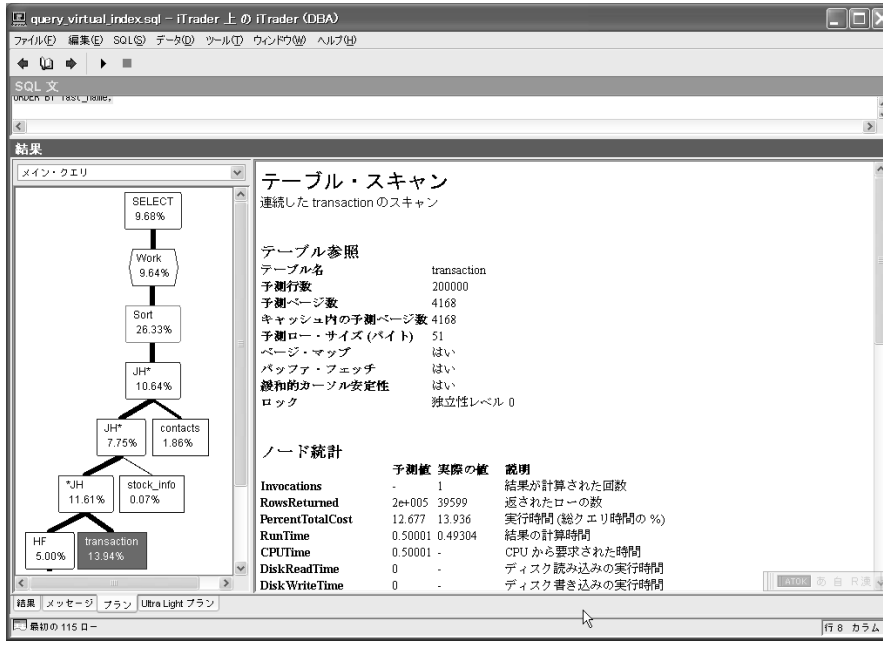


図4.5 : 統計情報付きのグラフィカルなプラン。左側の四角で囲まれたものをノードと呼び、個々の処理を表す。ノードを選択すると、右側に内部データが表示される

統計情報付きのグラフィカルなプランでは多くの情報が提示されるので、見るべきポイントをいくつか挙げておく。統計情報の予測値は実際の値と異なることが多いので、大きく違うときに注意する。

- インデックスの利用の確認

  ノードを個別に見ていき、意図したようにインデックスが利用されているか確認する。特に、赤い色のノードは最も処理時間の長いノードを示し、太線のツリーがメインの流れを示すので、その部分を重点的に調べる。

- 統計情報のRowsReturnedの予測値と実際の値の比較

- 統計情報のRunTimeの予測値と実際の値の比較
- 未確定述部の統計情報
  WHERE句などで行を絞り込む際に、カラムの統計情報から選択率（条件に該当する行数÷全行数）が予測される。

このようにしてアクセスプランを把握したのち、インデックスの作成が必要ならば、次節で説明するIndex Consultantや仮想インデックスの機能を使いながら、インデックスを新規作成することによりパフォーマンスが改善されるかどうか確かめる。

インデックスヒントの指定

どうしても意図したようにインデックスが使われない場合は、オプティマイザが使用すべきインデックスをWITH節で強制的に指定することができる。

```
SELECT *
  FROM "DBA"."customer" WITH (INDEX(ix_cust_name))
 WHERE fname LIKE 'M%'
```

インデックスヒントは多用せず、避けられない場合のみの最小限にしたほうがよい。データ量やキャッシュの状態によっては、別のインデックスの使用やテーブルスキャンをしたほうが速い場合もある。SQL Anywhereのオプティマイザは元来コストベースであり、データの統計やキャッシュ状態も考慮してプランを作成するので、原則的には、プランの選択はオプティマイザにまかせるのがよい。

4.2 Index Consultant

稼働中のデータベースに対して、効果的なインデックスを作成する作業は簡単ではない。実行される多数のクエリを1つずつ分析し、インデックスの候補を見つけ、インデックスの効果を確認し、ディスク容量の増加といった副作用を考慮しなければならない。

このようなDBAの作業を助けるSQL Anywhereのツールが、Index Consultantである。Index Consultantは、実行されているクエリを解析して、パフォーマンスの上昇が見込めるインデックスを推奨する[3]。その際、改善される程度やディスクスペースの増加容量が定量的に示される。アクセスプランの変化もその場で確認できるので、DBAはこれらの情報を頼りに効果的なインデックスを迅速に作成可能だ。

Index Consultantは、Sybase CentralとInteractive SQLの両方から利用できるが、利用目的が異なる。

- Sybase Centralからの利用 —— 複数のクエリに対して分析する
- Interactive SQLからの利用 —— 1つのクエリに対して分析する

順に説明する。

3 すでに作成されているインデックスに対して、削除候補も推奨する。

Column　インデックスの種類は何通りあるか

1つのテーブル内で作成可能なインデックスの組み合わせは多数ある。n個のカラムからなるテーブルにおいて、とりうるインデックスの種類は、以下の式で求められる。

$$\sum_{k} 2^{k} \frac{n!}{(n-k)!}$$

たとえば、10カラムからなるテーブルであれば、60億通りものインデックス定義がありうるのである。

4.2.1 インデックス推奨機能

Index Consultantは、ワークロード（workload）を解析して、推奨すべきインデックスを調査する。ワークロードとは、データベースが処理した一連のクエリの実行記録である。Index Consultantは、次のような手順でワークロードを利用する。

1. Index Consultantの開始
2. ワークロードの取得開始
3. クエリの実行
4. ワークロードの取得終了
5. 取得したワークロードを解析して、インデックスを推奨する
6. Index Consultantの終了

ワークロードの開始から終了までに発生したクエリが、1つのワークロードに保存される。ワークロードはデータベース内に保存されるので、あとから何度でもワーク

ロードの解析が可能だ。ワークロードの取得中、多少の負荷がデータベースサーバにかかるので、高負荷なデータベースに対して実行する際はあらかじめ影響を確認する。

Index Consultantは、ワークロード内で有効なインデックスを推奨する。誤解されそうな点を正しておく。

- Index Consultantは、静的なデータベーススキーマだけからインデックスを推奨するわけではない。ワークロードも考慮する。
- Index Consultantが考慮するのはワークロード内に記録されたクエリだけである。決して、データベースが過去に処理したすべてのクエリを調べるわけでも、発生しうるクエリをすべて予想するわけでもない。ワークロードの作成時に、考慮すべきクエリがなるべく含まれるようにする必要がある。
- ワークロードは、データベースが実際に行った処理に基づくものなので、マシンの性能やキャッシュの状況などデータベースの実行状態に依存する。あるワークロードで推奨されたインデックスが、ワークロードと異なる環境（マシンスペックやデータ量やキャッシュ量など）でも有効とは限らない。

Index Consultantは万能ではないが、DBAやアプリケーション開発者の作業をかなり軽減することは事実である。Index Consultantの特徴を理解して、上手に使ってほしい。

では、具体的な手順を見ていこう。Index Consultantは、Sybase Centralのウィザードとして利用する。データベースに接続し、［インデックス］を右クリックしてショートカットメニューの［インデックス・コンサルタント］を選択すると、ウィザードが開始される。

図4.6：［インデックス・コンサルタント］を選択

保存されているワークロードを分析するのか、新規にワークロードを作成するのか尋ねられるので、新規作成を選び、ワークロードの取得を開始する。

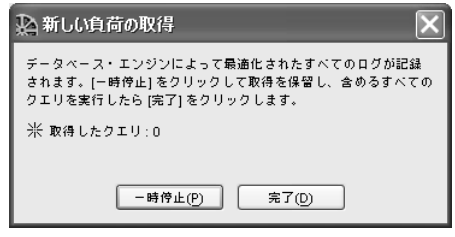


図4.7：［新しい負荷の取得］ダイアログボックス

このダイアログボックスが出ている間、データベースで処理されたクエリがワークロードとして蓄積される。［取得したクエリ］の数が増加するので、きちんと動作していることがわかる。クエリを実行し終えたら、［完了］ボタンをクリックしてワークロードの取得を終了し、ワークロードに名前を付けて保存する。

ワークロードの分析の前に、オプションを選択する2つのページがある。1つ目のページでは、インデックスの推奨に関して、クラスタードインデックスも含めて推奨するか否か、既存のセカンダリインデックスの削除を推奨するか否かを選択する。2つ目のページでは、作成するインデックスの上限サイズを指定する。

オプションを設定し終えるとワークロードの分析が始まり、しばらくすると結果が表示される（図4.8）。

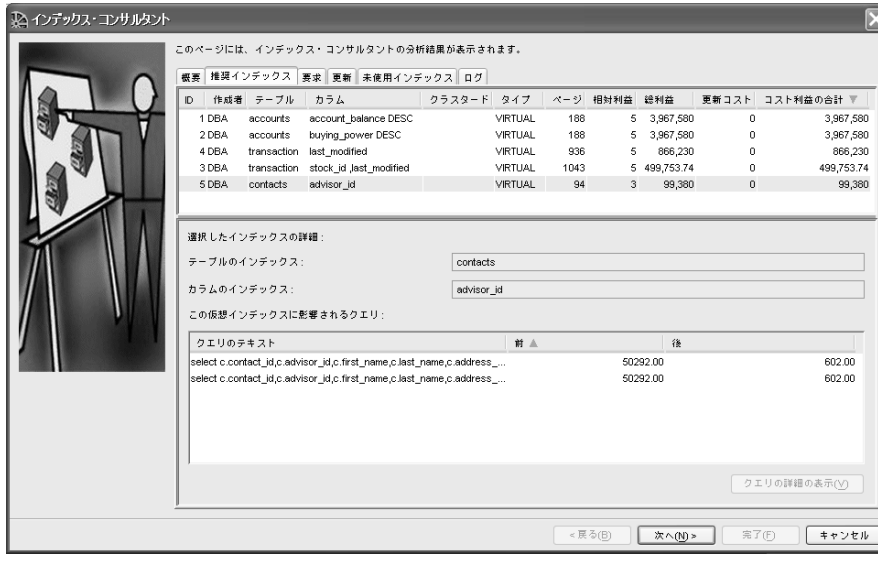


図4.8 : Index Consultantの分析結果画面。［ページ］で必要なページ量がわかる。［コスト利益の合計］はSQL Anywhere独自の単位で表した効果度で、インデックス同士を定量的に比較できる。［相対利益］は10点満点のスコアで表した推奨度（数字が大きいほど推奨度は高い）

Index Consultantの分析結果画面では、機能ごとにタブが用意されている。たとえば、［推奨インデックス］タブでは、推奨インデックスのある／なしによるアクセスプランの変化が確認できる。

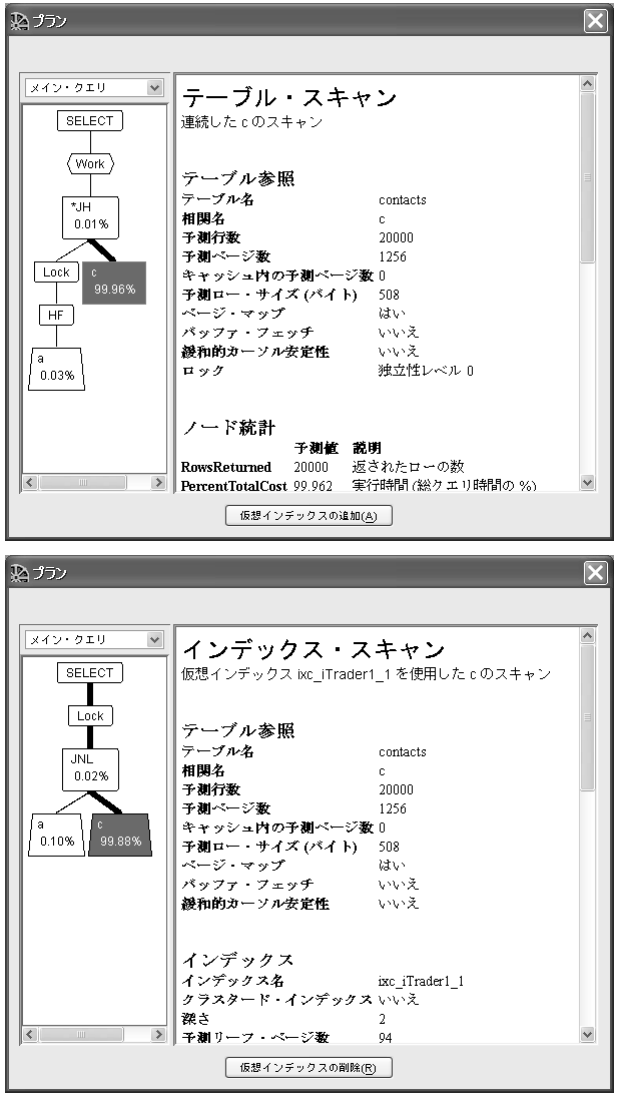


図4.9 : 上の画面は推奨前。テーブルスキャンによる検索で効率が悪い。下の画面は推奨後。インデックスによる検索が行われるようになった。仮想インデックスについては、次節で取り上げる

［次へ］ボタンをクリックすると、推奨されたインデックスを作成するスクリプトが示される。インデックス名を認識しやすい名前に変更して、必要なものを実行すればよい。

推奨インデックス：

```
/* Results of automatic index tuning for analysis iTrader1 */

DROP INDEX "DBA"."transaction"."idx_transaction_type";

CREATE  INDEX "ixc_iTrader1_1" ON "DBA"."accounts" ( account_balance DESC);

CREATE  INDEX "ixc_iTrader1_2" ON "DBA"."accounts" ( buying_power DESC);

CREATE  INDEX "ixc_iTrader1_3" ON "DBA"."transaction" ( stock_id ,last_modified );
```

図4.10：推奨インデックスが表示される

4.2.2 1つのクエリの分析

1つのクエリを分析するには、Interactive SQLから行えばよい。Interactive SQLの［SQL文］ボックスに分析したいクエリを書き、［ツール］－［インデックス・コンサルタント］を選択するとクエリの分析が始まり、推奨結果が表示される（図4.11）。

Results of the index tuning analysis

Summary | Recommended Indexes | Unused Indexes | Log

ID	Creator	Table	Columns	Clustered	Type	Pages	Relative Benefit	Tota
1	DBA	accounts	contact_id ,buying_power		VIRTUAL	209	5	18
2	DBA	transaction	account_id		VIRTUAL	936	7	18
3	DBA	contacts	last_name		VIRTUAL	94	5	18

図4.11：Interactive SQLで実施したIndex Consultantの結果

4.3 仮想インデックス

Index Consultantによるインデックスの推奨機能を見てきたが、仮想インデックスという機能を使って、ユーザ自身が試行錯誤しながらインデックスを作成し、その効果を調べることもできる[4]。

仮想インデックスとは、インデックスの効果を調べるための機能だ。物理的な実体をもたず、仮想インデックスを作成したその接続内だけに存在する。Interactive SQLなどでアクセスプランを表示する際、オプティマイザは仮想インデックスの存在を感知し、それを反映したプランを表示する。しかし、仮想インデックスは実際に利用可能なものではないので、クエリの実行には影響しない。また、仮想インデックスは、作成された接続内だけに存在する一時的なものなので、ほかの接続と干渉することもない。接続が切断されれば仮想インデックスも失われる。

インデックスの作成は一般にコストの大きな処理で、時間やディスク容量を必要とする。しかし、仮想インデックスは物理的実体を伴わないので短時間に作成でき、インデックスの効果を試行錯誤しやすい。

仮想インデックスの作成は、通常のインデックスを作成するCREATE INDEX文でVIRTUALキーワードを指定する。

```
CREATE VIRTUAL CLUSTERED INDEX "vix_accounts_contact_id_buyingpower"
ON accounts ( contact_id, buying_power );
```

4　Index Consultantでも、この仮想インデックスが内部で使われている。

4.4 フラグメンテーション

フラグメンテーションは断片化とも呼ばれ、ディスク内のデータの物理配置が細分化されて不連続になってしまうことだ。フラグメンテーションが起きても情報としてのデータが論理的に変化するわけではないが、データにアクセスするディスクI/Oの回数が増えるので、パフォーマンスには悪影響を及ぼす。

一方、フラグメンテーションが少ない場合、すなわち連続したデータ領域が多い場合は、第3章「SQL Anywhereの内部動作」で説明したブロックI/Oの効果によりパフォーマンスは向上する。論理的に隣り合うデータが物理的にも隣り合って連続的に配置されていれば、1回のディスクI/Oでアクセスできる機会が増えるからだ。

データベースにおけるフラグメンテーションは、次の2種類に分けられる。

1. データベースファイルのフラグメンテーション
2. テーブルのフラグメンテーション

さらに、物理的に不連続になるというフラグメンテーションとは多少意味合いが異なるが、

3. インデックスのフラグメンテーション

というものもある。これらの意味と解決方法とを順に説明する。

4.4.1 データベースファイルのフラグメンテーション

ファイルのデータをどのようにディスクに記録するかはOSが受け持つ役割であり、必ずしもデータが連続して配置されるとは限らない。特に、ファイルのサイズが

大きくなってディスクの空き容量が少なくなると、連続した領域を確保するのが難しくなり、ファイルのデータが細分化されて配置される。これがファイルのフラグメンテーションだ。

ファイルが断片化してしまうのは、SQL Anywhereのデータベースファイルも例外ではない。データベースファイルのフラグメント数は、データベースプロパティで確認できる。以下のSELECT文を実行すればよい。

```
SELECT DB_PROPERTY ('DBFileFragments');
```

また、フラグメンテーションが発生していれば、データベース起動直後のコンソールウィンドウで注意がうながされる。

図4.12：起動直後のサーバウィンドウ。「注意：」のあとに、「7個のディスク・フラグメントで構成されています」とある

■ 解消方法

ファイルのフラグメンテーションを解消するには、デフラグメンテーションを行えばよい。デフラグメンテーションとは、断片化されたデータを再配置して、フラグメ

ンテーションを解消する（少なくする）ことだ。デフラグメンテーションを実行するソフトウェアがあるので、それを利用する。Windowsであれば「デフラグ」が標準装備されているし、サードパーティ製品もある。

■ 防止策

ファイルのフラグメンテーションの発生はOSレベルの問題であるため、SQL Anywhereが制御することはできないが、その頻度を減らすことは可能だ。SQL Anywhereは、データを追加するときに空きスペースが足りなければデータベースのサイズを自動拡張する。拡張した部分をOSが連続して配置するとは限らないので、自動拡張の機会が増えればフラグメンテーションの機会も増えることになる。そこで、自動拡張する必要がないように、ある程度のデータベースサイズをあらかじめ確保して、データの増加に備えればよい。データベースサイズを増やすには、データベース作成後、ALTER DBSPACE文で行う。以下の例では、データベースサイズを400MBに増やしている。

```
ALTER DBSPACE system ADD 400 MB
```

なお、このとき追加した部分が断片化しているといけないので、いったんデータベースを停止し、デフラグメンテーションを実行しておく。

4.4.2 テーブルのフラグメンテーション

通常、1つの行は1つのページ内に書かれるが、既存の行をより大きなサイズで更新（UPDATE）したとき、ページ内に十分なスペースがなければ、その行のデータは複数のページにまたがって記録される。また、そもそもページサイズよりも大きな行を追加（INSERT）すれば、複数のページに記録するしかない。このように、1つの行のデータが複数のページに記録されることを行分割と呼び、このような現象をテ

ーブルのフラグメンテーションと呼ぶ。テーブルのフラグメンテーションはSQL Anywhere内部の問題であり、データベースファイルのフラグメンテーションの有無とは関係なく発生する。

SQL Anywhereには、テーブルのフラグメンテーションを確認するツールが2つある。

- sa_table_fragmentationプロシージャ
- dbinfoコマンド（-uオプション）

これらツールの使い方などの詳細は、第3章で説明したので、ここでは解消方法や防止策を挙げる。

■ 解消方法

テーブルのフラグメンテーションの解消は、REORGANIZE TABLE文（テーブルの再構築）で行う。REORGANIZE TABLE文は行を再配置する。その際、行が属する先頭ページをも移動させながら並べ替えるので、行のデータが1ページに収まるならば分割されていた行も1つになる。

REORGANIZE TABLE文の処理は、内部的には、テーブル全体を複数のグループに分け、グループごとに実行される。このとき、グループごとにテーブルロックがかかる。よって、REORGANIZE TABLE文の処理中、ずっとテーブルロックがかかるわけではないので、ほかの処理がすべて止まってしまうということはないが、負荷の高いデータベースに対して実行するときは注意してほしい。

なお、REORGANIZE TABLEの実行前に、以下のコマンドを実行して、処理が割り当てられる優先度を下げておくのも良いやり方だ。

```
SET TEMPORARY OPTION BACKGROUND_PRIORITY = 'ON'
```

■ 防止策1　PCTFREE

ページ内のUPDATE用の予備サイズはPCTFREEパラメータで設定できるので、多くの更新が予想されるならば大きめに確保するとよい。CREATE TABLE文などで、PCTFREEのキーワードのあとに、ページサイズに対する予備部分の百分率を指定する。

```
CREATE TABLE fragtest (
  pk INT PRIMARY KEY,
  s1 CHAR(100),
  s2 CHAR(100),
  PCTFREE 20
);
```

■ 防止策2　可変長データ

SQL Anywhereにおける可変長のデータ型には、次のものがある。

- CHAR
- VARCHAR
- LONG VARCHAR（TEXT）
- BINARY
- VARBINARY
- LONG BINARY
- NUMERIC（MONEY, SMALLMONEY）

これらに対して、あとから大きなサイズで更新すると行分割が起きやすい。

また、SQL Anywhereはデータ型によらず、NULL値にデータを割り当てないので、NULL値から実データに更新するときも行が分割されやすい。

いずれの場合も、デフォルト値を利用できるならばなるべくそれを使用して、デー

タの追加時に一定のサイズを埋めておく。

4.4.3 インデックスのフラグメンテーション

第3章で説明したように、行の追加や削除に伴いインデックス密度が低下し、インデックスの効率が悪くなることがある。その際、インデックスツリーの高さが必要以上に増すこともあるので注意が必要だ。このように、インデックス密度が低くなる現象をインデックスのフラグメンテーションと呼ぶ。SQL Anywhereでは、インデックスのフラグメンテーションが自動的に解消されることはないので、必要に応じてインデックスを再構築するとよい。再構築の作業はデータベース管理者が手動で実施してもよいし、SQL Anywhereのスケジュール機能を使って、定期的に実行してもよい。

■ 解消法

インデックスのフラグメンテーションの解消は、REORGANIZE TABLE文（インデックスの再構築）で行う。

```
REORGANIZE TABLE employee
PRIMARY KEY
```

PRIMARY KEYを指定すると主キーのインデックスのフラグメンテーションが解消される。外部キーやそれ以外のキーのインデックスのフラグメンテーションを解消するには、それぞれ、FOREIGN KEY *foreign_key_name*やINDEX *index_name*を指定すればよい。

インデックスを再構築する処理は新規にインデックスを作成して古いものと切り替える処理なので、ロックが発生する期間は短い。テーブルの再構築ほどには、データベース負荷を気にする必要はない。

4.5 その他の考慮事項

データベース設計時やアプリケーション開発時にパフォーマンスに関して考慮すべき点で、前節までに言及できなかったものを列挙する。

■ データベースファイルのバージョンを最新にする

SQL Anywhereのデータベースサーバはバージョン間の互換性が高く、過去のバージョンで作成されたデータベースファイルであっても実行できるが、最新の機能が利用できるわけではない[5]。最新の機能はデータベースファイルの形式に依存することもあるからだ。原則として、SQL Anywhereをメジャーまたはマイナーバージョンアップした際はunload/reload機能を用いてデータベースファイルを作り直し、最新にする。

なお、システムテーブルのSYSHISTORYには、データベースファイルが作成されたり起動されたときのバージョン情報の履歴が記録されている。

■ OPTIMIZATION_GOALオプション（all-rows, first-row）

データベースオプションのOPTIMIZATION_GOALは、first-rowとall-rowsのどちらかを指定でき、オプティマイザがクエリプランを選別するときの判断基準となる。first-rowでは、クエリ結果の最初の行が早く返ってくることを重視し、all-rowsでは、クエリ結果の全体が早く返ってくるようにする。言い換えれば、first-rowはレスポンス重視で、all-rowsはスループット重視と言える。

SQL Anywhere 8.0.2からはall-rowsがデフォルト値となっている。ただし、旧バージョンからunload/reloadしてバージョンアップした際、データベースオプション

5 ただし、SQL Anywhere 5とそれ以降とでは通信プロトコルが異なるので、クライアントアプリケーションに変更が必要なことがある。詳細は、マニュアルの「SQL Anywhere Studio 新機能ガイド：バージョン5アプリケーションのアップグレード」を参照。

の設定も引き継がれるので、新バージョンのデータベースで意図したオプションになっているか注意する。

カーソル利用時など個別のクエリで一時的にfirst-rowを選択したい場合は、次のようにSELECT文中のFASTFIRSTROWキーワードを使う。

```
SELECT * FROM customer WITH (FASTFIRSTROW)
```

■ ページサイズ

大きなページサイズも小さなページサイズもそれぞれ長所短所があり、最適なサイズはシステムに依存する。よくわからない場合は4096を指定する（デフォルトは2048）。dbinit文を使い、以下のように指定してデータベースを作成する。このサイズで問題があるとわかったら、2048や8192などの別のサイズを検討すればよい。

```
dbinit －p 4096 test.db
```

■ ファイルを別々のディスクに配置する

SQL Anywhereが使用するファイルには、データベースファイル、トランザクションログファイル、テンポラリファイルがあるが、これらを別々のディスクに分散すればパフォーマンスは向上する。

■ テーブル内の合計カラムサイズ（カラム数）を大きくしすぎない

テーブルの合計カラムサイズがページサイズよりも大きいと、行分割が発生しやすい。また、ページサイズの半分を超える合計カラムサイズ（たとえば60％）だと、行の追加時に1つのページに2つの行が記録されることはないので、無駄なスペースが多くなる。合計カラムサイズとページサイズとの関係に注意する。

■ 主キーのカラムサイズを大きくしすぎない

主キーのカラムサイズが大きすぎると、インデックスに必要なサイズも大きくなり、インデックススキャンの効率が落ちる。複数のカラムを主キーとして指定する場合は特に注意する。

■ カラム定義の順番

SQL Anywhereは、CREATE TABLE文で定義されたカラムの順番で行をページに記録し、行にアクセスする。頻繁に利用するカラムはCREATE TABLE文の先頭に近いほうがよい。

■ インデックススキャン可能なSQL文かどうか

第3章で説明したように、インデックスが定義されていたとしても、SQL文の書き方によってはインデックスが利用されない。たとえば、次の文ではorder_dateのインデックスは利用できないので注意してほしい。

インデックス利用不可の例：

```
SELECT * from sales_order
 WHERE year(order_date)='2000'
```

■ トランザクションに含まれないテンポラリテーブル

テンポラリテーブルを作成する際、NOT TRANSACTIONALというキーワードを付けると、そのテーブルへのデータ操作はトランザクションに含まれず、ロールバックログの作成を抑制できる。長いトランザクションで、テンポラリテーブルを頻繁に利用するときに検討するとよい。

■ トランザクションの長さを適切に

トランザクションが長くなると、ロールバックログやロックを維持する負担が増える。トランザクションの長さは必要最小限にするとよい。

第5章 バックアップとリカバリ

本章では、バックアップとリカバリに関するSQL Anywhereの機能を説明する。バックアップとリカバリに関する運用指針を立てるには、何がバックアップされているのかといったSQL Anywhereの動きを知っているとよい。それらについては第3章「SQL Anywhereの内部動作」で説明しているので、本章の前にそちらを読んでおくのが望ましい。

言うまでもないが、データベースのバックアップとリカバリは単独で計画できるようなものではない。システム全体、つまりほかのアプリケーションやOS、ストレージやネットワークなどのハードウェアも含めて総合的に運用する必要がある。本章の内容を踏まえて、バックアップとリカバリの戦略を立案してもらいたい。

5.1 障害の種類

障害にもいろいろなレベルがあるが、本書で扱うのはシステム障害とメディア障害である[1]。

5.1.1 システム障害

システム障害とは、アプリケーションやOSのエラーなどの原因でデータベースサーバのプロセスが異常終了し、正常に終了できない障害のことである。システム障害が起きると、トランザクションの途中のデータは不完全なまま残ることになる。SQL Anywhereは起動時にコミットされていないデータを以下のように自動的にリカバリするので、システム障害からのリカバリをユーザが意識する必要は特にない。

1　障害対策として、ディスクをRAID構成にするなどハードウェア的な対策も考慮すべきだが、本書では触れない。

1. データベースファイルを前回チェックポイント時に戻す
   チェックポイントログにあるページをデータベースファイルに上書きすれば、データベースファイルは前回チェックポイント時の状態に戻る。
2. チェックポイント以後の変更を適用する
   トランザクションログのデータから、チェックポイント以後の変更を適用する。
3. ロールバックする
   トランザクションログにはコミット前の変更も記録されるため、ロールバックログを使ってロールバックし、コミット済みのデータだけが残るようにする。

こうしてシステム障害が発生した時点におけるコミット済みのデータに確実に戻れる。

5.1.2 メディア障害

メディア障害とは、HDDのクラッシュやSQL AnywhereやOSのバグなどの原因で、データベースファイルやトランザクションログファイルが壊れて利用できなくなる障害のことである[2]。以下のようなケースが考えられる。

1. データベースファイルが利用できなくなった場合（トランザクションログファイルは正常）

この場合、トランザクションログファイルに障害時点までのコミット済みトランザクションが記録されているので、障害時点へのリカバリが可能だ。前回バックアップしたデータベースファイルに対して、トランザクションログを適用すればよい。

2　第2章「SQL Anywhereの使い方」で説明したように、SQL Anywhereが使用するファイルには、データベースファイル、トランザクションログファイル、テンポラリファイルの3種類がある。テンポラリファイルはSQL処理で利用される一時的なファイルなので、バックアップやリカバリに関係するのはデータベースファイルとトランザクションログファイルだけだ。

2. トランザクションログファイルが利用できなくなった場合（データベースファイルは正常）

この場合、障害時点へのリカバリはできない。データベースファイルはチェックポイント時のままなので、障害が起こる直前のチェックポイント時点には戻れるが、その時点から障害が起こるまでの間に発生したトランザクションは失われる。トランザクションログをミラーリングすれば、このような危険性を減らすことが可能だ。

3. データベースファイルとトランザクションログファイルの両方が利用できない場合

この場合は、バックアップされたデータベースを使用する。ただし、バックアップ時点から障害が起こるまでに発生したトランザクションは失われる。

■ トランザクションログのミラーリング

デフォルトではトランザクションログファイルは1つであるが、もう1つ増やすことが可能だ。2つのトランザクションログには同じデータが記録されるので、同一の内容をもつ。トランザクションログを別々のディスクに配置すれば、片方のディスクに障害が起きてもトランザクションログのデータが失われることはない。このように、トランザクションログのミラーリングによりデータが冗長となり、システムの耐障害性が高まる。

トランザクションログのミラーリングは、① データベース作成時に指定することもできるし、② ミラーログを持たない既存のデータベースに対して設定することもできる[3]。

① データベース作成時に指定する

dbinitの引数で、-tでデフォルトのトランザクションログファイルの位置を-mでミラーログの位置を指定する。

3　CREATE DATABASE文やSybase Central（GUI）でも可能。

```
$ dbinit -t c:¥primary.log -m d:¥mirror.log test.db
```

② ミラーログを持たない既存のデータベースに設定する

dblogコマンドを用いる。データベースが停止している状態で行う。

```
$ dblog -m d:¥mirror.log test.db
```

5.1.3 その他

火災や地震などによってマシンが壊れてしまう災害などがある。考慮すべき範囲がデータベースのバックアップだけにとどまらず本書の範囲を超えるので、次節でライブバックアップ機能を紹介するにとどめる。

5.2 バックアップの種類

メディア障害からリカバリするには、データベースファイルやトランザクションログファイルを随時バックアップしておかなければならない。バックアップ手段はさまざまなものがある。以下に挙げるものは直交する概念を含むので、どれか1つを選ぶというよりは適切に組み合わせて使用するとよい。

5.2.1 オンラインとオフライン

バックアップを実行する際に、データベースを止めるか否かで利用するコマンドは異なる。データベースを止めることができる場合は、データベースファイルやトランザクションログファイルをOSのコマンドから単にコピーすればバックアップでき

る。一方、データベースを起動したままバックアップするには、SQL Anywhereのコマンドを利用する。

5.2.2 DATABASE BACKUP文とdbbackupコマンド

オンラインバックアップするには、DATABASE BACKUP文かdbbackupコマンドを使う。DATABASE BACKUP文を使うと、データベースサーバと同じマシン上にバックアップファイルが作成される。一方、dbbackupコマンドを使うと、コマンドを実行したマシン上にバックアップファイルが作成される。

BACKUP DATABASE文の例

```
BACKUP DATABASE
DIRECTORY backup_directory;
```

dbbackupコマンドの例

```
$ dbbackup -c "dsn=ASA 9.0 Sample" -t backup_directory
```

5.2.3 フルバックアップと増分バックアップ

フルバックアップでは、データベースファイルとトランザクションログファイルをすべてバックアップする。データベース全体が対象になるため、データ転送などの時間はかかるが、フルバックアップされたデータベースはそのまま起動できる。

一方、増分バックアップでは、前回バックアップされた時点（フルバックアップまたは増分バックアップ）からの差分のみをバックアップする。SQL Anywhereでは、トランザクションログの差分のみをバックアップすることを意味している。差分データだけが対象となるので転送時間などは短い反面、バックアップ時点のデータベー

スを復元するには一手間かかり、前回フルバックアップされたデータベースに対して差分を適用することでバックアップ時点のデータベースを復元できる。

このように長所短所があるので、「ある時点の完全な状態を保存し、そのあとの変更分だけを保存していく」ことを繰り返すのがバックアップの基本手順となる。たとえば、毎日深夜0時にバックアップを実行するとして、日曜日はフルバックアップし、その他の曜日は前日からの増分バックアップとする。このとき、火曜日の正午にメディア障害が起こりデータベースファイルが失われたとすると（トランザクションログのミラーリングのお陰でトランザクションログファイルは利用可能とする）、日曜日のフルバックアップに対してバックアップされていた月曜日と火曜日との変更分を適用することで、火曜日深夜0時のデータに復旧できる。さらに、障害が起こったマシンに残されたトランザクションログファイルを適用して障害時点のデータベースにリカバリする。

このように、フルバックアップの間隔が長くなったり増分バックアップの回数が増えたりすると、復旧にかかる手間や時間も大きくなる。また、フルバックアップから次のフルバックアップまでが1世代となるので、これらを残しておくことで世代管理も可能となる。

ここまでは厳密に用語を使い分けていなかったが、変更分をバックアップする際は、① 前回からの変更分をバックアップする「増分バックアップ」と ② フルバックアップからの変更をバックアップする「差分バックアップ」が考えられる（図5.1）。SQL Anywhereでは、どちらの場合もトランザクションログのみがバックアップ対象となる。

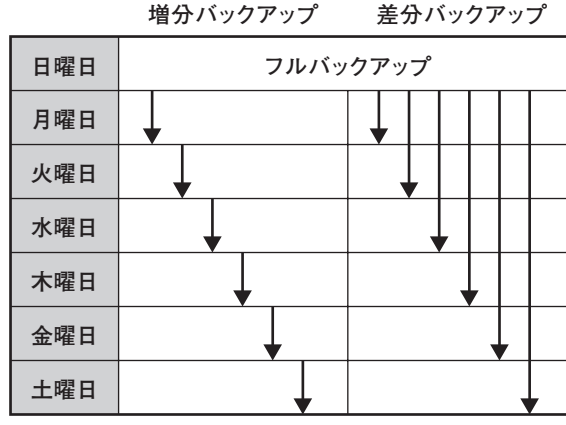


図5.1：増分バックアップと差分バックアップの違い。矢印はその曜日に発生したトランザクションを示す

差分バックアップは、以下のように実行する。

BACKUP DATABASE文の場合

```
BACKUP DATABASE
DIRECTORY backup_directory
TRANSACTION LOG ONLY;
```

dbbackupコマンドの場合

```
$ dbbackup -c "dsn=ASA 9.0 Sample" -t backup_directory
```

増分バックアップするには、バックアップ時に既存のトランザクションを切り捨てる（または切り替える）。トランザクションログの削除については、本章の「5.4　トランザクションログの切り捨てとリネーム」で後述する。

BACKUP DATABASE文の場合

```
BACKUP DATABASE
DIRECTORY backup_directory
TRANSACTION LOG ONLY
TRANSACTION LOG TRUNCATE;
```

dbbackupコマンドの場合

```
$ dbbackup -c "dsn=ASA 9.0 Sample" -x -t backup_directory
```

5.2.4 アーカイブバックアップ

バックアップされるファイルは、通常、データベースファイルとトランザクションログファイルとに分かれている。これはイメージバックアップと呼ばれる。また、テープドライブなどに保存する場合の利便性から、1つのファイルとしてバックアップすることも可能だ。この方式をアーカイブバックアップと呼ぶ。

アーカイブバックアップするには、BACKUP DATABASE文において、保存先をTOオプションで指定する。

```
BACKUP DATABASE
TO '¥¥¥¥.¥¥tape0'
ATTENDED OFF
WITH COMMENT 'May 6 backup'
```

ATTENDEDオプションがOFFの場合、テープの容量がなくなった場合はバックアップに失敗するが、ONの場合はテープの交換などのアクションが促される。

5.2.5 ライブバックアップ

ライブバックアップとは、データベースサーバと異なるマシン上（「マシンR」と書く）にもトランザクションログを置き、トランザクションログを冗長化する機能だ。マシンRでライブバックアップコマンドを実行すると、データベースサーバに接続し、その接続が維持される。そして、トランザクションログファイルの更新がリアルタイムにマシンRにも伝わる。

トランザクションログを冗長化するという点でライブバックアップはトランザクションログのミラーリングと似ているが、目的が異なる（図5.2）。ミラーリングでは、データベースサーバと同じマシン上でトランザクションログを複製され（普通は別々のディスク上に配置される）、メディア障害に備えるための機能である。一方、ライブバックアップは異なるマシン上でトランザクションログを複製し、ディスクだけでなく、データベースサーバマシンが使えなくなるような障害に備えるための機能である。その場合、障害が起きて使用できなくなったマシンの代わりにマシンR上でデータベースサーバを動作させることになる。

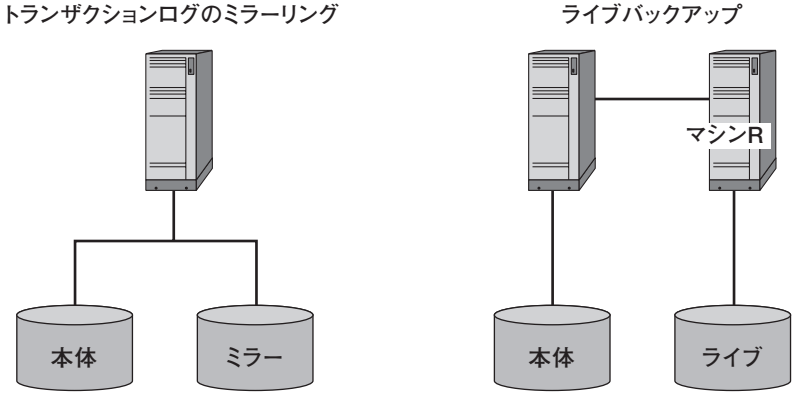


図5.2：トランザクションログのミラーリングとライブバックアップとの違い

なお、ライブバックアップでは、サーバの負荷が高い場合などトランザクションログの転送に遅延が発生する可能性があるため、サーバがダウンしたときにコミット済みのトランザクションがすべてマシンR上に記録されているとは限らない。

ライブバックアップは、dbbackupコマンドの-lオプションで行う。マシンRでのコマンド実行例を次に示す。

```
$ dbbackup -l path¥filename.log -c "connection_string"
```

5.3 監査

正常でない状態のデータベースをバックアップしても意味がないので、バックアップする前に、現在のデータベースの状態が正しいことを確認するとよい。監査(verify)でエラーが発見された場合は、そのデータベースに壊れた部分があることを意味するので、使うべきではない。バックアップしておいたデータベースからリカバリすることになる。監査を実行する方法として、コマンドとSQL文の両方を使える。

- コマンドの場合

  dbvalidコマンドで行う。

  ```
  $ dbvalid -c "接続先"
  ```

- SQL文の場合

  sa_validateストアドプロシージャを呼ぶ。

  ```
  call sa_validate
  ```

デフォルトの監査(WITH EXPRESS CHECK)は、テーブルページとインデッ

クスページとをスキャンし、インデックスエントリーの示す行IDがテーブルページに存在するかチェックする。行IDレベルで一致するものを探すので、キー値の一致まではチェックしない。インデックスエントリーの数が行の数を超えないことも確認する。また、拡張ページはスキャンしない。このため、完全ではないが、高速な監査が可能となっている。

もう少し詳しく監査したい場合は次に挙げるオプションを指定する（VALIDATE TABLE構文のオプション。括弧内はdbvalidコマンドのオプション）。

- WITH DATA CHECK (-fd)
  拡張ページを含めて行のカラムをすべて追う。
- WITH INDEX CHECK (-fi)
  キー値レベルでインデックスのエントリーと行とが一致するか調べる。これにより、インデックスの示す行が存在する確実性が高まる。
- WITH FULL CHECK (-f)
  上記2つを束ねたもの。

Column　チェックサムによる監査

本文に挙げた監査方法ではなく、ページのチェックサムによる監査を選択できる。データベース作成時にページごとにチェックサムを作り(dbinitの-sオプション)、VALIDATE CHECKSUM構文（dbvalid -s）でチェックする。各ページのチェックサムを計算して比較するので、確実性の高い監査が可能となる。

5.4 トランザクションログの切り捨てとリネーム

バックアップのデフォルト動作ではオリジナルのトランザクションログファイルはそのまま利用されるので、トランザクションログが永久に増え続けてしまう。トランザクションログを削除するには、バックアップ時のオプションでトランザクションログの切り捨てや切り替えを指定するとよい。

ただし、データベースが同期に関わっている場合（Mobile Link同期クライアントやSQL Remote）は注意が必要だ。トランザクションログをもとにデータが同期されるので、不用意にログを切り捨ててしまうと正常な同期ができなくなる。このような場合には、トランザクションログの切り捨てではなくリネームを使う。

■ トランザクションログの切り捨て

バックアップを実行後、既存のログの内容を削除する（図5.3）。

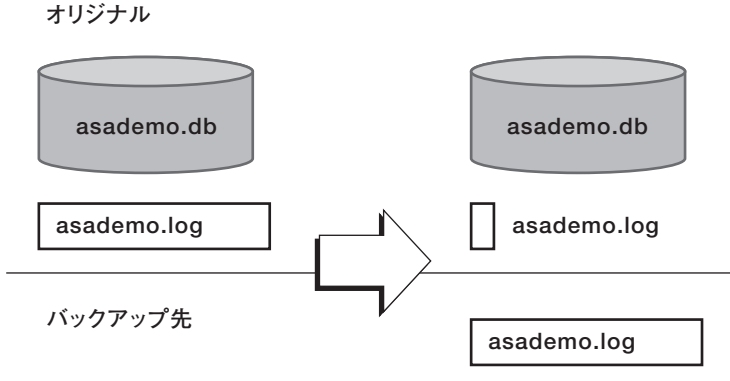


図5.3：トランザクションログの切り捨て

- コマンドの場合

```
$ dbbackup -c "connection_string" -x [ -t ] backup_directory
```

- SQL文の場合

```
BACKUP DATABASE
DIRECTORY backup_directory
[ TRANSACTION LOG ONLY ]
TRANSACTION LOG TRUNCATE
```

角括弧内は、トランザクションログのみバックアップする際に利用する（前述した増分バックアップに対応する）。

■ トランザクションログの切り替え

トランザクションログの切り替えの場合、バックアップを実行すると既存のログは切り捨てられるが、その内容はファイル名を変えて保存される（図5.4）。

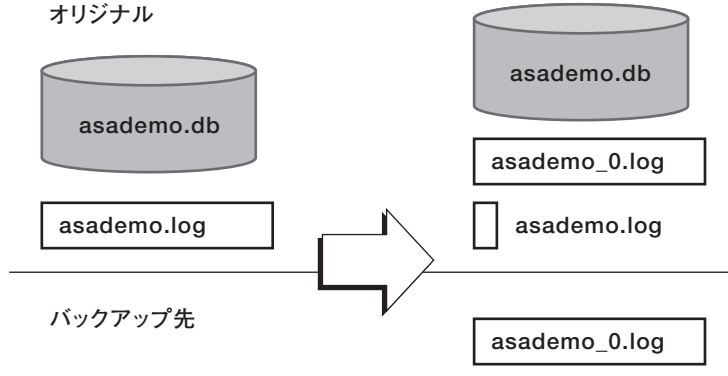


図5.4：トランザクションログのリネーム

コマンドは基本的に変わらず、ログを切り替える（リネームする）オプションを付加する。

- コマンドの場合

```
$ dbbackup -c "connection_string" -r [ -t ] backup_directory
```

- SQL文の場合

```
BACKUP DATABASE
DIRECTORY backup_directory
[ TRANSACTION LOG ONLY ]
TRANSACTION LOG RENAME
```

5.5 バックアップの内部動作

たくさんのトランザクションをデータベースが次々に処理している状態であってもバックアップは実行できる。このとき、バックアップされたデータベースは、バックアップを実行した時点（つまり、コマンドを実行した時点）とは若干ずれるので注意が必要だ。以下に、バックアップのSQL Anywhereの内部動作を示す。

① バックアップの開始

② チェックポイントの発生
バックアップが終了するまで、ほかのチェックポイントは禁止される。

③ データベースファイルのコピー
バックアップ中に②以外のチェックポイントは発生しないので、バックアップされるデータベースファイルは必ず②時点のものになる。なお、①～③の処理と並行してトランザクションは処理され、トランザクションログは増え続けている。

④ トランザクションログのコピー
　トランザクションログの先頭ページから最終ページまでをコピーする。よって、バックアップされるトランザクションログは、この時点のものとなる。

⑤ トランザクションログの削除が指定されていれば、削除する
　未コミットのログがあれば、新しいトランザクションログファイルに繰り越される。

このように、バックアップされたデータベースファイルは②の時点のもので、トランザクションログファイルは④の時点のものだ。バックアップされたデータベースを起動するとき、必ずリカバリが発生し、④のトランザクションがデータベースファイルに適用され、④の時点のデータを持つデータベースが起動する。

■ パフォーマンスについて

トランザクションログにトランザクション途中のログが記録されていた場合、新しいログにそれらが繰り越される。よって、ログの切り捨てや切り替え処理は、実行中のトランザクションの終了を待つ必要はなく、トランザクション処理中でもログの切り替えや切り捨てが可能だ。

5.6 リカバリの手順

以下では、リカバリで使用するコマンドを紹介する。読みやすくするために、本番環境にあるファイルを小文字で、バックアップされていたファイルを大文字で記す。なお、実際にリカバリするときは、万一の失敗に備えてオリジナルのファイルをコピーして実行するとよいのだが、それらのSQL Anywhereと関連しない作業はここで

は省略した。

5.6.1 データベースファイルが失われたとき

本番環境：asademo.db（破壊）、 asademo.log（正常）
バックアップ：ASADEMO.DB、ASADEMO.LOG

データベースファイルが失われてもトランザクションログが残っているので、障害時点にリカバリ可能だ。

1. バックアップされているトランザクションログの適用
   dbeng9の-aオプションを使う。トランザクション適用後、データベースサーバは自動的に終了する。バックアップされているトランザクションログが複数ある場合は、古いものから順にすべて適用する。

   ```
   $ dbeng9 ASADEMO.DB -a ASADEMO.LOG
   ```

   これで、バックアップ時のデータベースができる。
2. 本番環境のトランザクションログの適用

   ```
   $ dbeng9 ASADEMO.DB -a asademo.log
   ```

   これで、障害発生時点のデータベースにリカバリできた。
3. データベースの監査（verify）
4. ASADEMO.DB を本番環境で利用する

Column　dbtran コマンド

dbtran コマンドは、トランザクションログに書かれているトランザクションをSQL 文に変換するコマンドだ。次のようにすると、asademo.log にあるトランザクションの内容をasademo.sql ファイルに出力する。

```
$ dbtran asademo.log asademo.sql
```

トランザクションログからdbeng9 でリカバリする代わりに、dbtran で出力されたSQL 文ファイルをデータベースファイルに適用してもリカバリ可能だ。
dbtran によるリカバリの利点として、複数のトランザクションログファイルからのリカバリが挙げられる。複数のトランザクションログを 1 つのSQL 文ファイルに一度に変換できるので、一度の適用で済むからだ。たとえば、c:¥backup ディレクトリ内に複数のトランザクションログを置き、次のようにすると、recoverylog.sql にすべてのトランザクションがSQL で書かれる。

```
$ dbtran -m "c:¥backup" -n recoverylog.sql
```

なお、dbtran でログファイルの順番も知ることができる。dbtran を実行する際にオフセット値が標準出力されるので、この値が若いほど古いトランザクションログファイルを意味する。

5.6.2　トランザクションログが失われたとき（ミラーなし）

本番環境：asademo.db（正常）、asademo.log（破壊）
バックアップ：ASADEMO.DB、ASADEMO.LOG

トランザクションログが壊れてしまったので障害時点まではリカバリできないが、データベースファイルが残っているので、そのデータベースファイルが更新されたチ

ェックポイント（障害発生前のチェックポイント）時点のデータベースは得られる。

1. チェックポイントリカバリ
   トランザクションログがない状態なので、dbeng9の-fオプションを利用して、チェックポイントリカバリを行う。実行後、新しいトランザクションログが作成され、データベースは自動的に終了する。なお、本番環境がdbsrv9で実行されていたらdbeng9ではなくdbsrv9を利用する。
   壊れてしまったトランザクションログがないディレクトリで、次のようにする。

```
$ dbeng9 asademo.db -f
```

   このとき、新しいトランザクションログファイルが生成される。

5.6.3 トランザクションログファイルの片方が失われた

本番環境：asademo.db（正常）、asademo.log／asademo_mirror.log（どちらか破壊）
バックアップ：ASADEMO.DB、ASADEMO.LOG

トランザクションログが残っているので、障害時点までリカバリ可能だ。

1. どちらのトランザクションログファイルが正常なのか
   この判定はdbtranコマンドで確認する。dbtranコマンドが、トランザクションログをSQLに変換する際、ログファイルが壊れていればエラーを出力する。正常に変換できたほうが、利用可能なログファイルだ。

```
$ dbtran asademo.log
```

2. 正常なログファイルで起動

asademo.dbと正常なログファイルとで、データベースを起動する。自動的にリカバリされて、データベースが起動する。

5.6.4 ライブバックアップからのリカバリ

本番環境：asademo.db（破壊）、asademo.log（破壊）
バックアップ：ASADEMO.DB、ASADEMO.LOG、
ASADEMO_LIVEBACKUP.LOG（正常）

ライブバックアップされたトランザクションがあるので、障害時点にリカバリ可能だ。ただし、サーバの負荷状態などにより、障害時点で完了しているトランザクションすべてが必ずしも伝達されているわけではない。

1. ライブバックアップされたトランザクションログの適用

```
$ dbeng9 ASADEMO.db -a ASADEMO_LIVEBACKUP.LOG
```

第6章 Programming API

本章では、Java、.NET、Perl、PHPの各言語からSQL Anywhereに接続する方法を説明する。なお、各言語の仕様については専門書などを参照していただきたい。

6.1 Java

データベースに接続するAPIであるJDBCには4つの種類がある[1]。

- Type 1 —— JDBC-ODBC bridge
- Type 2 —— Native-API partly-Java driver
- Type 3 —— Net-protocol all-Java driver
- Type 4 —— Native-protocol all-Java driver

SQL Anywhereには2種類のJDBC 2.0ドライバがあり、Type 2/3/4に対応する。各ドライバの対応関係を示す。

ドライバ名称	ライブラリの場所	JDBCの種類
iAnywhere JDBC driver	%ASANY9%¥java¥jodbc.jar	Type 2
jConnect	%ASANY9%¥Shared¥jConnect-5_5¥classes¥jconn2.jar	Type 3 Type 4

以下では、利用されることの多いType 2/4における接続方法を説明する。

1 参考URL：http://java.sun.com/products/jdbc/overview.html

6.1.1 jConnect（Type 4）

jConnectは、Sybase社のAdaptive Server Enterpriseと共通のJDBCドライバで、インストールされる場所は%ASANYSH9%ディレクトリ（デフォルトではC:¥Program Files¥Sybase¥Shared）となっている。また、バグフィックスであるEBF（Express Bug Fix）も、SQL Anywhere本体のものとは別にSybase社のサイトで配布されている[2]。jConnectはPure Javaモジュールなので、アプリケーションを実行する際はjConnectライブラリが参照可能であればよい。

jConnectを使う際のドライバマネージャと接続URLは、以下のとおり。

- ドライバマネージャ —— com.sybase.jdbc2.jdbc.SybDriver
- 接続URL —— jdbc:sybase:Tds:127.0.0.1:2638

サンプルコードを以下に示す[3]。

```
Properties props = new Properties();
props.put("USER",             "dba");
props.put("PASSWORD",         "sql");
props.put("DYNAMIC_PREPARE", "true");
String driver = "com.sybase.jdbc2.jdbc.SybDriver";
String url    = "jdbc:sybase:Tds:127.0.0.1:2638";
Class.forName(driver);
Connection conn =  DriverManager.getConnection(url, props);
```

PreparedStatementを利用する場合は、上記のコード例のように、DYNAMIC_PREPAREオプションを設定する。

2 EBFダウンロードURL：http://www.sybase.com/products/informationmanagement/softwaredeveloperkit/jconnect
ドキュメントは、http://sybooks.sybase.com/jc.htmlからダウンロードできる。

3 ただし、ほかのサンプルコードも含め、SQL Anywhereと関わらない例外処理などは省略した。詳細は、マニュアルや%ASANY9%¥Samples¥ASAにあるサンプルプログラム集で確認できる。

URLによるプロパティの指定

jConnectのプロパティは、Propertyオブジェクトで指定するだけでなく、URLのオプションとしても指定できる。たとえば、ユーザとパスワードの指定は次のように記述できる[4]。

```
jdbc:sybase:Tds:127.0.0.1:2638/?USER=dba&PASSWORD=sql
```

データベース名の指定

データベースサーバに複数のデータベースがある場合は、接続するデータベース名も指定する。それには、URLでServiceNameプロパティを用いる。

```
jdbc:sybase:Tds:127.0.0.1:2638/?ServiceName=database_name
```

Adaptive Server Enterpriseとの互換モード

jConnect経由でSQL Anywhereに接続するとき、以下の2つのストアドプロシージャが実行され、データベースプロパティがAdaptive Server Enterprise互換モードに設定される。この互換モードが都合悪い場合はストアドプロシージャを書き換えるとよい。

- dbo.sp_tsql_environment
- dbo.sp_mda

DataSourceを使った接続

JNDI経由でDataSourceを使用する場合は、以下のいずれかのクラスを利用する。

4 このほかにもさまざまな接続プロパティがあるので、詳細はjConnectのドキュメントを参照していただきたい。

- com.sybase.jdbc2.jdbc.SybDataSource
- com.sybase.jdbc2.jdbcSybConnectionPoolDataSource

コネクションプーリング機能がアプリケーションサーバから提供されている場合は後者を用いる。

6.1.2 iAnywhere JDBC（Type 2）

iAnywhere JDBCは、OSネイティブなライブラリを使用するType 2のJDBCドライバだ。一般に、Type 4のものよりもパフォーマンスに優れている。iAnywhere JDBCは、データベースの接続にCommand Sequence Client/Server Protocolを使用する。これは、ODBCでSQL Anywhereに接続するときに使われているものと同じだ[5]。

ネイティブライブラリを使用するので、JDBCドライバであるjodbc.jarのほかに以下のライブラリを使用する。このライブラリは、システムパスになければならない。

- Windows —— %ASANY9%¥win32¥dbjodbc9.dll
- Linux —— $ASANY9/lib/libdbjodbc9.so

ドライバマネージャと接続URLは、以下のようになる。サンプルコードはType 4のものと同様だ。

- ドライバマネージャ —— ianywhere.ml.jdbcodbc.IDriver
- 接続URL —— jdbc:odbc:dsn=ASA 9.0 Sample

5 jConnectではTDSプロトコルが用いられる。

6.2 .NET

.NET環境からSQL Anywhereに接続する方法は3つある。

- iAnywhere.Data.AsaClient
  SQL Anywhereにダイレクトに接続するためのデータプロバイダ。高速であり、この使用をお勧めする。なお、Windows CE上の.NETデータプロバイダは、このiAnywhere.Data.AsaClientのみがサポートされている。
- System.Data.Oledb
  SQL AnywhereのOLE DBドライバ経由で接続するためのデータプロバイダ。
- System.Data.Odbc
  SQL AnywhereのODBCドライバ経由で接続するためのデータプロバイダ。

本書では、iAnywhere.Data.AsaClientを使った接続方法を取り上げる。iAnywhere.Data.AsaClientをVisual Studio .NETで使用するには、iAnywhere.Data.AsaClient.dll を参照できるようにしなければならない。

1. ［ソリューションエクスプローラ］ウィンドウの［参照設定］を右クリックし、ショートカットメニューの［参照の追加］を選択する。これにより、［参照の追加］ダイアログボックスが表示される。
2. ［参照の追加］ダイアログボックスの［.NET］タブで［参照］ボタンをクリックし、［%ASANY9%¥win32¥ iAnywhere.Data.AsaClient.dll］を選択する。

サンプルコードを以下に示す。

```
using iAnywhere.Data.AsaClient

AsaConnection conn = new AsaConnection( "Data Source=ASA 9.0 Sample"
);
conn.Open();
AsaCommand cmd = new AsaCommand( "select emp_lname from employee",
conn );
AsaDataReader reader = cmd.ExecuteReader();
listEmployees.BeginUpdate();
while( reader.Read() ) {
     listEmployees.Items.Add( reader.GetString( 0 ) );
}
listEmployees.EndUpdate();
reader.Close();
conn.Close();
```

■ コネクションプーリング

SQL Anywhere .NET Providerはコネクションプーリングをサポートしている。この機能はデフォルトでオンになっていて、デフォルト値は「`Max Pool Size=100; Min Pool Size=0`」だ。

```
"Data Source=ASA 9.0 Sample;UID=DBA;PWD=SQL;POOLING=TRUE;Max Pool
Size=50;Min Pool Size=5"
```

■ DataSetを使う例

AsaDataAdapterオブジェクトと.NET FrameworkのDataSetオブジェクトを用いると、DataGridなどのGUIコンポーネントに結果セットを表示できる。以下に、サンプルコードを示す。

```
using iAnywhere.Data.AsaClient;

AsaConnection conn = new AsaConnection("Data Source=ASA 9.0 Sample");
conn.Open();
dataGridResults.DataSource = null;
```

```
AsaCommand cmd     = new AsaCommand("SELECT * FROM employee", conn );
AsaDataReader   dr = cmd.ExecuteReader();
dataGridResults.DataSource = dr;
dr.Close();
conn.close;
```

その他のオブジェクト

トランザクションを制御するAsaTransactionオブジェクトなどがある。これらについては、マニュアルを参照していただきたい。

6.2.1 配備

AsaClientを使うクライアントアプリケーションを配備するには、次のDLLも一緒にインストールする。

Windowsの場合

- %ASANY9%¥win32¥iAnywhere.Data.AsaClient.dll
- %ASANY9%¥win32¥dbdata9.dll

なお、iAnywhere.Data.AsaClient.dllについては、Global Assembly Cache（GAC）にアセンブリを登録する必要がある。

Windows CEの場合

- %ASANY9%ce¥iAnywhere.Data.AsaClient.dll
- %ASANY9%ce¥arm.30¥dbdata9.dll
- %ASANY9%ce¥emulator.30¥dbdata9.dll
- %ASANY9%ce¥mips.30¥dbdata9.dll
- %ASANY9%ce¥x86¥dbdata9.dll

dbdata9.dllについてはプラットフォームに合うものを選び、デバイスのWindowsディレクトリにインストールする。

6.3 Perl

Perlからデータベース接続するには、DBI/DBDと呼ばれるモジュールを利用するのが一般的だ。DBIモジュールはデータベース製品に依存しないインターフェイスである。一方、DBDモジュールはデータベース固有のドライバで、データベースに対応するDBDモジュールを接続時に指定することで、そのデータベースに接続できる。このようにして、製品に依存しないコーディングが可能となっている。

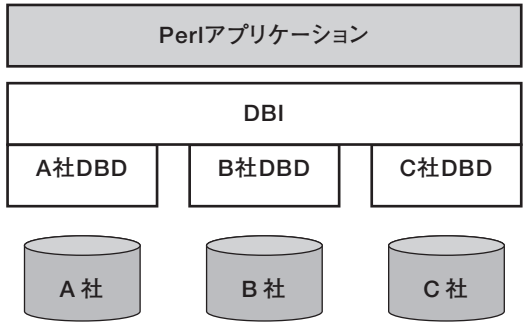


図6.1：DBI/DBDモジュールの関係

SQL AnywhereにはDBD::ASAnyというDBDモジュールがある。このDBDモジュールとDBIを使って、PerlからSQL Anywhereに接続することができる。

DBD::ASAnyをコンパイルして利用するには、以下の環境が必要だ。

- Cコンパイラ（Windowsでは、Microsoft Visual Cコンパイラのみサポート）

- Perl 5.6.0以降（Windowsでは、ActivePerl 5.6.0 build 616以降）
- DBI 1.34以降

6.3.1 インストール

PerlおよびDBIをインストールしておく。次の手順で、DBD::ASAnyをコンパイルし、インストールする。

```
$ cd $ASANY9/src/perl
$ perl Makefile.PL
$ make
# make install
```

Windows上では、makeではなくnmakeを用いる。

6.3.2 コーディング

サンプルコードを以下に示す。

```
#!/usr/bin/env perl

use DBI;

$user     = 'dba';
$passwd   = 'sql';
$database = 'asademo';
$db = DBI->connect("DBI:ASAny:ENG=$database;DBN=$database", $user,
$passwd);
$sth = $db->prepare("SELECT fname, lname FROM customer");
$sth->execute;
while($row = $sth->fetchrow_arrayref) {
    print join("¥t¥t", @$row), "¥n";
}
$sth->finish;
```

```
$db->disconnect;
```

6.4 PHP

PHPからSQL Anywhereに接続する方法は3つある。設定が容易でコーディングも簡単なので、SQL Anywhere PHPモジュールがお勧めだ。

- SQL Anywhere PHPモジュール
- PHPのODBCモジュール
- PHPのSybase-CTモジュール[6]

6.4.1 インストール

まず、インストールすべきモジュールを選択する。

Windowsの場合

PHP4用とPHP5用のモジュールがある。

表6.1：Windows用のSQL Anywhere PHPモジュール

モジュール	説明
php4_sqlanywhere9.dll	PHP version 4 library for Windows
php5_sqlanywhere9.dll	PHP version 5 library for Windows

6 TDSプロトコルを利用した接続で、Adaptive Server EnterpriseやMicrosoft SQL Serverと互換性が高いAPI。

Linux

PHP4用とPHP5用のモジュールがある。また、マルチスレッドの扱いが異なるものがある。Apache 2.xなどのようにマルチスレッド環境で利用する場合は、*_r.soを利用する。

表6.2 : Linux用のSQL Anywhere PHPモジュール

モジュール	説明
php4_sqlanywhere9.so	Non-threaded PHP version 4 library for UNIX
php4_sqlanywhere9_r.so	Threaded PHP version 4 library for UNIX
php5_sqlanywhere9.so	Non-threaded PHP version 5 library for UNIX
php5_sqlanywhere9_r.so	Threaded PHP version 5 library for UNIX

選択したモジュールを、php.iniで指定されているextension_dirにコピーする。

次に、php.iniを編集し、そのモジュールがロードされるようにする（もしくはスクリプト上で、dl()関数を用いて動的にロードしてもよい）。たとえば、php5_sqlanywhere9_r.soを選んだ場合は、次のようになる[7]。

```
extension=php5_sqlanywhere9_r.so
```

6.4.2 コーディング

サンプルコードを以下に示す。

7 このほか、LD_LIBRARY_PATHなどの環境変数が設定されていることを確認する。Apacheであれば、設定ファイルhttpd.conf中のSetEnvやPassEnvで設定するとよいだろう。

```
<?
  $conn = sqlanywhere_connect( "uid=dba;pwd=sql" );
  if( ! $conn ) {
    echo "sqlanywhere_connect failed¥n";
  } else {
    $result = sqlanywhere_query( $conn, "select * from customer" );
      if( ! $result ) {
        echo "sqlanywhere_query failed!";
      } else {
        echo "query completed successfully¥n";
        sqlanywhere_result_all( $result );
        sqlanywhere_free_result( $result );
      }
    echo "Connected successfully¥n";
    sqlanywhere_disconnect( $conn );
  }
?>
```

%ASANY9%¥src¥php¥examplesにサンプルコードもあるので、それらを見ればその他のAPIについてもすぐに理解できるはずだ。

6.5 その他の言語

SQL Anywhere固有のドライバは、本章で紹介したものだけである。そのほかの言語であってもODBC接続のAPIがあれば接続可能なので、詳細については各言語のドキュメントを参照していただきたい。

第7章 Mobile Link

7.1 Mobile Linkとは

Mobile Linkとは、データベース同期のためのミドルウェアで、複数のSQL Anywhereと1つのセンターデータベースとを双方向に同期させる仕組みだ。Mobile Linkを使えば、センターデータベースを中心として多数のSQL Anywhere同士でデータを共有することができる。このような同期テクノロジは、データベースが分散してしまうモバイルシステムには欠かすことができない[1]。

Mobile Linkの用語では、センターデータベースの役割を「統合データベース」と呼び、分散しているSQL Anywhereの役割を「リモートデータベース」と呼ぶ。「統合データベース」はリモートデータベースのデータを「統合」し、一方、統合データベースからみて「離れた」場所に「リモートデータベース」は位置するので、この役割名で呼ばれる[2]。

Mobile Linkの適用例を2つ挙げてみよう。1つ目の例は、営業スタッフが携帯する顧客情報管理システムだ。ノートパソコンやPDAなどのモバイルデバイスを用いて、営業スタッフが外出先で顧客データを活用する状況を想定しよう（図7.1）。ネットワーク環境が常に利用できるとは限らず、レスポンスも重視されるため、デバイス単独で業務処理ができるとよい。そこで、モバイルデバイスにアプリケーションとSQL Anywhereとをインストールして顧客情報をデータベース化すれば、デバイス上で検索可能となる。また、更新された情報があれば、逐次デバイス上のSQL Anywhereに記録していく。一方、本社には統合データベースがあり、顧客や営業結果に関する全情報を把握する。営業スタッフ全員分となればデータサイズもかな

1 「データベースの同期」と言うとクラスタリングの用途を思い浮かべる読者もあるかと思うが、Mobile Linkの同期はそのような目的ではなく、分散したデータを整合性を保ちながら共有するためのものである。

2 統合データベースとリモートデータベースとが同一マシンにあったり、近接したマシンにあったりすることもあるので、名前に惑わされないようにしてほしい。

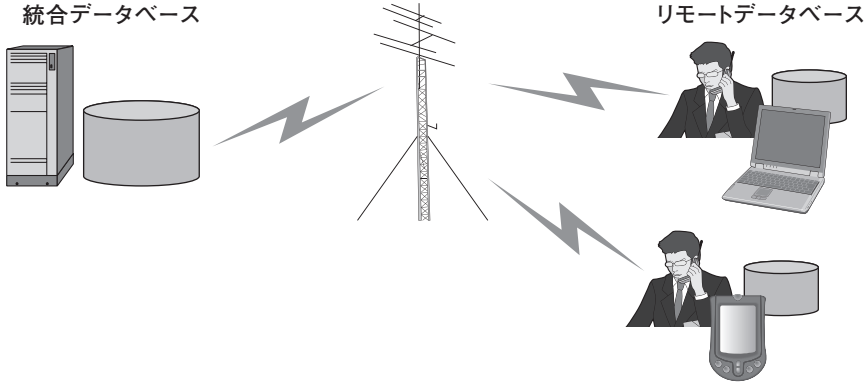


図7.1：営業スタッフのためのモバイルシステムの例

りの量になるだろう。

ここで技術的課題となるのが、本社の統合データベースと営業スタッフが携帯するデバイスのリモートデータベースとの情報をどのようにして共有するか、すなわちデータベースの同期方法である[3]。統合データベースのコピーを持ち歩ければよいが、物理的にそれは不可能であるし、そうする必要もない。営業スタッフは担当する顧客情報さえ手元にあれば済むので、必要なデータが同期できればよい。このような場面で要求されるデータベースの同期機能には次のようなものがある（図7.2）。

1. 統合データベースの一部分を切り出して同期できること
   データを切り出す際には、行やカラム単位、またはその両方で、任意の部分を指定できるとよい。
2. データを双方向に同期できること
   外出先ではデータの検索だけでなく更新もできないと不便なので、リモートデ

3　ここで「デバイス」とは、リモートデータベースのあるマシンを総称したものである。PDAやノートパソコンを念頭におくが、技術的にはデスクトップPCでもよい。

ータベースで発生した更新を統合データベースに伝える必要もある。また、同期完了時点で統合データベースのデータとデバイス上のデータとが一致するのが好ましい[4]。

3. 競合したデータを同期時に解決できること
   営業スタッフ同士が情報を共有した場合、同一であるはずのデータに対して別々に更新してしまうことが起こりうる。単純にデータをやり取りしたのではデータの不整合が起きてしまうので、同期時にデータの競合を検出し、解決できなければならない。
4. 低速で不安定な回線であっても確実に同期できること
   外出先で同期する場合、無線通信（無線LANや携帯電話・PHSによるデータ通信）を利用するだろうから、回線が不安定であっても確実な同期が行われなければならない。

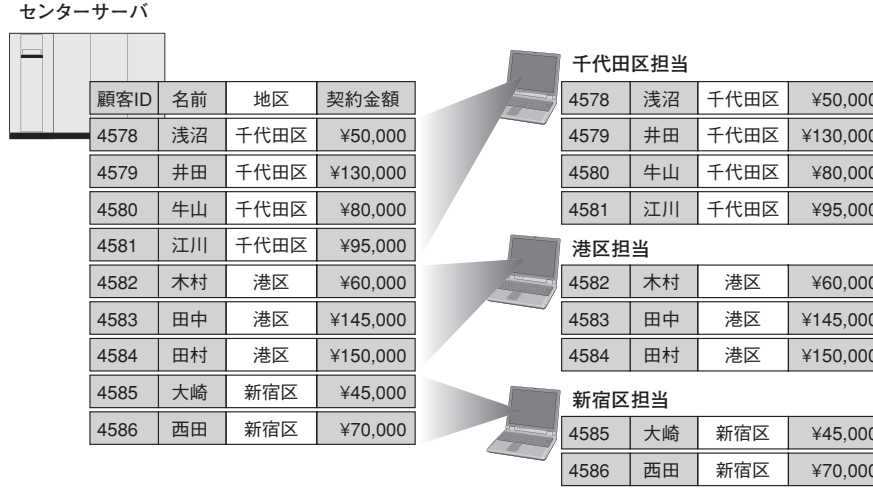


図7.2：同期の概念図。センターサーバのデータを地区担当の3つのリモートデータベースで同期している

4　たとえば、SQL Remoteは「ゆるやかな同期」であるため、データの伝達にタイムラグが発生する。

2つ目の例は、本社と多数の支店を持つ流通業界などで利用される業務システムの例だ（図7.3）。各支店にリモートデータベースを配備し、支店のトランザクションは自身のリモートデータベースに対して行う。本社には統合データベースを置いて全支店のデータを管理し、データウェアハウスとして業務分析する。リモートデータベースの情報は夜間などにバッチ処理として、定期的に本社に伝える。

このような同期を用いたアーキテクチャは、支店のアプリケーションが直接統合データベースにアクセスする場合に比べ、個々のトランザクションがWAN経由でなされることがなく処理が分散するので、統合データベース側と支店側の双方にとってパフォーマンスに優れている。また、本社・支店間で通信できなくとも支店でのトランザクションが完了するので、障害にも強い。同期に対する課題は1つ目の例と共通する。

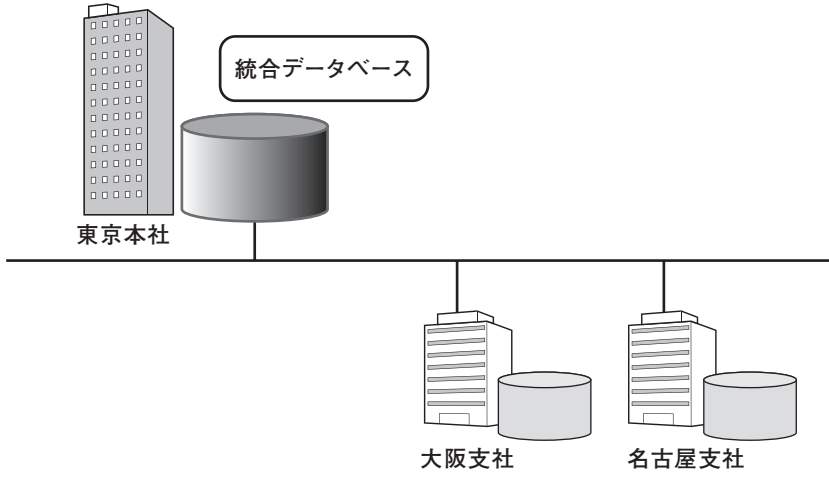


図7.3：本社と支店からなる分散データベースの例

このように、データベースの同期にはさまざまな技術的課題があるが、それに応えるのがMobile Linkという同期技術だ。Mobile Linkは、統合データベースと複数

のリモートデータベースとが同期するプラットフォームを提供する。テーブル内の行と列の両方を指定して同期の対象とすることができ、トランザクションは双方向に伝わり、競合解決ロジックも柔軟に実装できる。同期途中で不意に回線が切断されてもデータの整合性が失われることはない。また、更新されたデータだけ同期するなど、冗長なデータの転送を抑制できる。本章では、Mobile Linkのさまざまな機能を紹介し、実践的な同期システムの構築方法を見ていく。

7.1.1 Mobile Linkのアーキテクチャ

Mobile Linkのアーキテクチャは図7.4のようになる。統合データベースサーバとリモートデータベースサーバとが直接に接続することはなく、Mobile Linkサーバが両者の通信を仲介して同期を制御する。

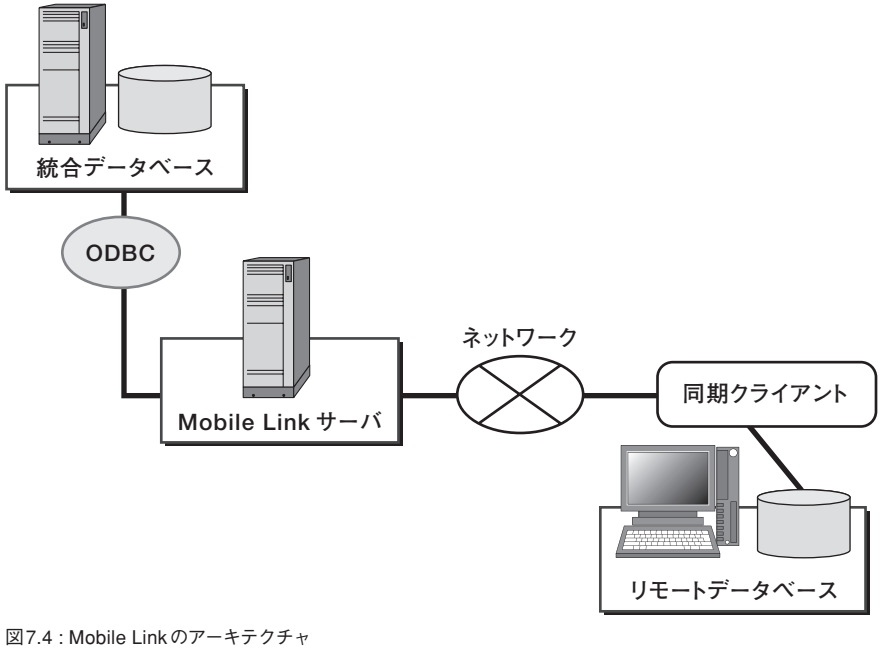


図7.4 : Mobile Linkのアーキテクチャ

まず、この3つのサーバの役割について説明する（以後、読みやすさを優先して、誤解の恐れのない限り、データベースサーバのこともデータベースと書く）。

■ 統合データベース

統合データベースは、リモートデータベース上の同期対象となるデータすべてを保持する。同期されないデータが統合データベースやリモートデータベースにあってもかまわない。

Mobile Linkが対応する統合データベースとして、以下のものがある[5]。このようにiAnywhere社以外のデータベースにも対応している。

- SQL Anywhere
- Sybase Adaptive Server Enterprise
- Oracle
- Microsoft SQL Server
- IBM DB2

統合データベースは、同期されるデータだけでなく、Mobile Link同期を実行するために必要なシステム情報（同期ユーザ情報や同期スクリプト、競合解決のスクリプト、同期ステータスなど）も保持する。Mobile Linkサーバはこれらを参照して、同期を制御する。

■ Mobile Linkサーバ

Mobile Linkサーバは、統合データベースとリモートデータベース間の通信を仲介

5　Mobile Link同期を行うには、Mobile Link同期用のシステムテーブルやストアドプロシージャが必要である。SQL Anywhereのデータベースはそれらがデフォルトで備わっている。そのほかのデータベースについては、各データベース用の設定スクリプトがSQL Anywhere Studioに用意されているので、それを実行すればよい。

し、同期プロセスを制御する。統合データベースとMobile Link間のやり取りは、ODBCで行われる。Mobile Linkサーバは同期開始時にMobile Link同期に関するシステム情報を統合データベースから読み出し、その情報に従って同期を処理する。

一方、リモートデータベースとMobile Linkサーバ間は、以下の2つの接続方法がある。

- TCP/IP

  TCP/IP上で、Mobile Link固有のプロトコルで通信する。SSLによる暗号化も可能である。

- HTTP/S

  HTTPに従ってMobile Linkのデータをやり取りする。HTTP/Sによる暗号化も可能である。

Column　Active Syncを使った同期

Windows CE上のSQL Anywhereから同期する場合、Active Sync経由でも同期できる。この場合、クレードルのActive Syncを通してホストPCとやり取りし、ホストPCとMobile Linkサーバ間はTCP/IPやHTTP/Sによる接続になる。もちろん、Windows CEデバイスにネットワークカードを装着し、TCP/IPやHTTP/Sで直接Mobile Linkサーバに接続することも可能だ。

Mobile Linkサーバは、複数の同期クライアントを同時に処理できる。デフォルトでは5つの同期処理用スレッドが待機していて、同期クライアントからのリクエストを受信すると同期を開始する。同期終了後、スレッドは待機状態に戻る。1つのスレッドは一度に1つの同期セッションのみを担当する。

同期スレッド数以上のリクエストがMobile Linkサーバに一度に到達すると、い

ったんMobile Linkサーバでキューイングされ、リクエストは待たされる。同期スレッドが同期セッションを終えるとキュー内からリクエストが選ばれ、新たな同期セッションが開始される（図7.5）。

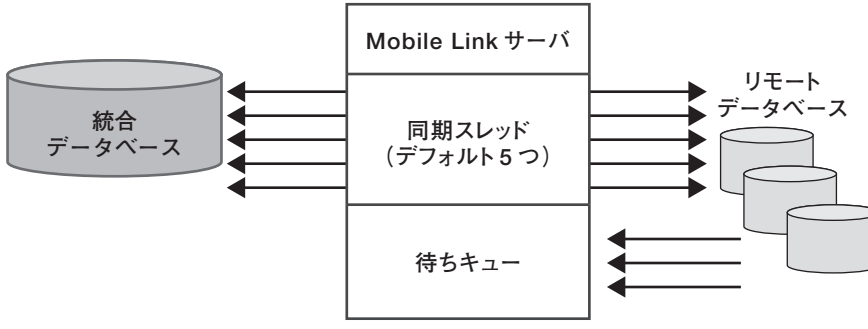


図7.5：同期スレッドと待ちキュー。なお、Mobile Linkサーバが制御用として統合データベースに接続するセッションもあるので、統合データベース上のODBCセッション数は同期スレッド数以上ある

Column　Webサーバ経由の同期

同期クライアントからのHTTPリクエストをMobile Linkサーバが直接受け取るのではなく、別のWebサーバを経由させることもできる。Mobile Link Redirectorというモジュールがあるので、これをWebサーバに組み込めばよい。同期クライアントからのHTTPリクエストを受け取ったWebサーバ（Mobile Link Redirector）は、指定されたMobile Linkサーバにリクエストを転送する。このため、Webサーバはリバースプロキシの役割を担っている（図7.6）。

Webサーバをファイアウォールとして機能させたいときや、複数のMobile Linkサーバに処理を分散させるためのロードバランサとして機能させたいときにMobile Link Redirectorを利用するとよい。Mobile Link Redirectorには、Mobile Linkサーバの状態をチェックして転送を制御する機能もある。

Mobile Link Redirectorは次の4つのWebサーバをサポートする（角括弧内はMobile Link Redirector名）。

- Sun One（Netscape iPlanet Enterprise Edition）[NSAPI Redirector]

- Microsoft IIS［ISAPI Redirector］
- Apache TomcatなどJava Servlet API 2.3をサポートするもの［Servlet Redirector］
- Apache Webサーバ（1.3.x / 2.0.x）［Native Apache Redirector］
- M-Business Anywhere Webサーバ［M-Business Anywhere Redirector］

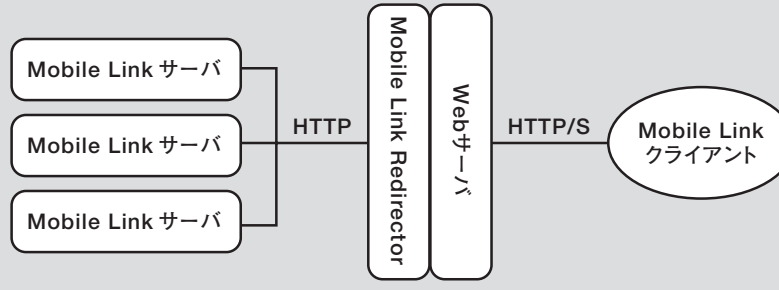


図7.6 : Mobile Link Redirector

■ リモートデータベース

リモートデータベースは、iAnywhere社のデータベースだけに対応し、以下の2つがある。

- SQL Anywhere
- Ultra Light

リモートデータベースとMobile Linkサーバとが通信すると今まで書いてきたが、より詳細に記述すると、同期クライアントが存在し、同期クライアントがMobile Linkサーバとの通信を担っている。詳しくは次のセクションで説明する。

7.1.2 同期プロセス

これまでのところで、統合データベース、Mobile Linkサーバ、リモートデータベースの役割を説明した。次に、どのように同期処理が行われるのか見ていくことにする。

同期プロセスは、アップロードとダウンロードの2つのプロセスからなる（図7.7）。同期プロセスの起点は同期クライアントだ[6]。同期クライアントとは、リモートデータベースと同じデバイス上で動くdbmlsyncというプログラムで、Mobile Linkサー

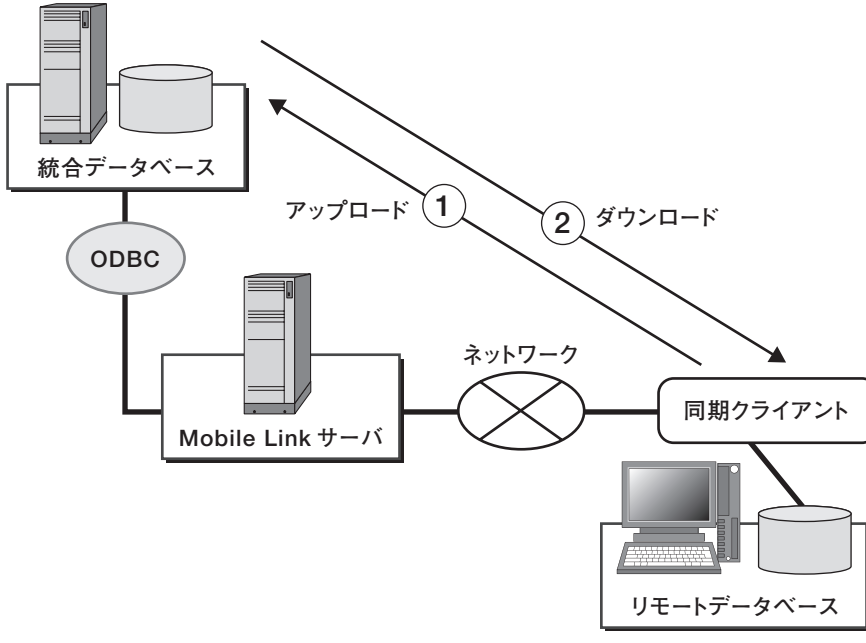


図7.7 : 同期プロセス

6　では、誰が同期クライアントを実行するのか。基本はデバイス上のユーザ（やアプリケーション）である。定期的に同期をスケジュールすることもできる。また、サーバから指示することもできるが、本書ではこのサーバ起動同期（Server Initiated Synchronization）に触れないので、詳細についてはマニュアルを参照していただきたい。

バに対するデータの送受信を担当する。起動された同期クライアントは、リモートデータベースのトランザクションログを調べて、前回同期成功時以降に変更のあった行のみを抽出し、Mobile Linkサーバに送信する（アップロードストリーム）[7]。送信されるデータの形式はMobile Link独自のものである。

アップロードストリームを受け取ったMobile Linkサーバは、それをSQL文に翻訳して統合データベースにODBC経由で実行する。こうして、リモートデータベースのデータが統合データベースに反映される。翻訳の方法に関しては、あらかじめアップロード用の同期スクリプトとして作成し、統合データベースに保存しておく。Mobile Linkサーバは同期前に同期スクリプトを調べ、同期処理に利用する。

続いて、Mobile Linkサーバはダウンロード用の同期スクリプトに従って統合データベースからODBCでデータを取得し、そのデータをMobile Link固有のデータ形式にして元の同期クライアントに返信する（ダウンロードストリーム）。この同期スクリプトもSQL文としてあらかじめ用意しておく。

ダウンロードストリームを受け取った同期クライアントは、そのデータをリモートデータベースに反映させる。

以上がMobile Linkによる同期プロセスの概略だ。「アップロードとダウンロードの2つの過程で同期が処理され、処理の仕方は同期スクリプトとして作成しておく」ことが現段階ではわかればよいが、ここで重要なポイントは以下の2つだ。

- Mobile Linkサーバと同期クライアントがトランザクションを管理しているので、同期途中でエラーが起こったり通信回線が切断されてしまったりしても更新データはロールバックされて同期前の状態に戻り、統合データベースもリモートデータベースも不整合を起こさない。
- 統合データベースから見れば、Mobile LinkサーバはODBC経由でSQLをやり

7　初めての同期であれば、同期設定以降に発生したトランザクションとなる。

取りする通常のデータベースクライアントなので、同期スクリプトに記述可能なロジックは統合データベースが提供する環境そのものである。よって、同期処理に関する統合データベースの設定やチューニングも、通常のデータベースクライアントと同じように行えばよい。

■ SQL Anywhere Studio 8からの変更

SQL Anywhere Studio 8.0.2からは同期プロセスが改良されている。8.0.1までは、ダウンロードストリームのあとでリモートデータベースからMobile Linkサーバへの確認プロセスというものがさらに必要だった。SQL Anywhere Studio 8.0.2以降では、その確認プロセスは省略可能となり、上述したアップロードストリームとダウンロードストリームによる同期プロセスがデフォルトとなっている。確認プロセスがなくなったことで、同期パフォーマンスが向上した。

■ アップロード時にMobile Linkへ送られるトランザクション

同期クライアントがリモートデータベースのトランザクションログからアップロードストリームを作成するとき、1つの行（主キー）に対する複数のトランザクションは1つにまとめられ、転送の無駄が省かれる。たとえば、次のようなテーブルがリモートデータベースにあったとしよう。

```
CREATE TABLE Sample (
    ID       INT PRIMARY KEY,
    VALUE    INT
)
```

このテーブルに、3つのトランザクションを流す。

```
INSERT INTO Sample VALUES (1, 10);
UPDATE Sample SET VALUE = 20 WHERE ID = 1;
UPDATE Sample SET VALUE = 30 WHERE ID = 1;
```

ここで同期すると、3つのトランザクションではなく、1つのトランザクションとして送られる。アップロードストリームのイメージを仮にSQL文で表すとすれば、次のようになる。

```
INSERT INTO Sample VALUES (1, 30)
```

このように、1つの行に対してデータ操作を行うSQL文（INSERT、UPDATE、DELETE）を繰り返しても、アップロードストリームでは最終結果となる1つのトランザクションとして送られる[8]。したがって、追加された行が同期されることなく削除されれば、次回同期時にはその行は送信されない。

以上がデフォルトの動作であるが、同期クライアントの -tuオプションで、この機能をオフにできる（transaction-level upload）。統合データベース側でトリガを利用してロジックを実装しているなど、トランザクションをまとめたくない場合に効果的だ。

■ アップロードストリームの「前回同期成功時」とは

アップロードストリーム生成の開始点となる「前回同期成功時」とは、技術的にはトランザクションログのオフセット番号として管理されている[9]。オフセット番号とは、トランザクションのID番号のことで、コミットごとに採番されてトランザクションログに記録される。オフセット番号の大小はコミットされたトランザクションの時系列に一致し、オフセット番号が小さいほど過去のものとなる。「同期クライアントがアップロードストリームを生成するとき、リモートデータベースのトランザクションログを調べる」と書いたが、これは指定されたオフセット番号以降（から最新まで）

8 実は最新のデータだけでなく、競合解決用に前回同期成功時の値も送られる。詳細については、本章の「7.4.4 競合解決」を参照。

9 progress番号やstate番号と呼ばれることもある。符号なし64ビット整数値なので、値が足りなくなる事態は現実的にはあり得ない。

のトランザクションについて調べることを意味する。

前回成功時のオフセット番号は、リモートデータベースのSYSSYNCテーブルと統合データベースのML_SUBSCRIPTIONテーブルが保持し、アップロードストリームの終了時に、新しいオフセット番号がそれぞれ記録される。このためダウンロードストリームの途中で回線が切断して同期が失敗し、再度同期したとしても、すでにオフセット番号は更新されているので（ダウンロードストリームが開始されたということは、アップロードが正常に行われたことを意味する）、先ほどアップロードされたデータが再送される無駄はない。

アップロードストリーム開始時に統合データベースとリモートデータベースそれぞれのオフセット番号を比較し、データベースの整合性を保とうとする。もし両者のオフセット番号が異なれば、デフォルトでは統合データベースのオフセット番号を使ってアップロードストリームをやり直す。統合データベースのオフセットが小さい場合（統合データベースがバックアップからリストアされた場合など）、アップロード済みのトランザクションが再送される。一方、リモートデータベースのオフセット値が小さい場合（リモートデータベースがバックアップからリストアされた場合など）、統合データベースのオフセット値以降のトランザクションがアップロードされた後データがダウンロードされる。いずれの場合も、障害が起きたほうのデータベースが障害時点のデータに近づくので、通常はこのデフォルト動作で問題ない。

デフォルト動作で都合が悪い場合は、dbmlsyncの-raや-rbオプションで制御できる。たとえば、バックアップされていたリモートデータベースをリストアした場合、統合データベースのオフセット番号はリストアされたリモートデータベースには存在しない可能性もあり、デフォルト動作では同期できないこともある。このような場合には、-rbオプションを使えばリモートデータベースのオフセット番号が強制的に使われるようになり、同期が成功する。

■ リモートデータベースのロック

同期処理の整合性を保つため、同期処理の間（アップロードストリームの作成からダウンロードストリームの適用まで）、同期クライアントはリモートデータベースの同期対象テーブルに共有ロックをかける[10]。同期処理中に同期とは別のトランザクションが発生してしまうと、前回同期以降の変更をアップロードする際などで整合性が保てないからだ。よって、同期処理中のテーブルに対して、ほかのトランザクションは読み取りできるが書き込みできない。

■ Mobile Linkサーバのアイソレーションレベル

Mobile Linkサーバが統合データベースにODBC接続する際、そのアイソレーション（isolation：独立性）レベルはデフォルトでSQL_TXN_READ_COMMITTEDとなっている。競合解決時に反復可能読み取り（repeatable read）が必要となるからだ。

7.1.3 まとめ

Mobile Linkの同期プロセスはアップロードとダウンロードからなり、統合データベースに保存されているそれぞれの同期スクリプトに従って、Mobile Linkサーバが同期を行う。同期スクリプトの書き方を、次節以降で具体的に見ていく。

10 Windows CE上の場合は排他ロックとなる。

7.2 Getting Started

7.2.1 データベースの作成

実際に同期スクリプトを書いてMobile Linkを体験してみよう。例題として、商品テーブルの同期を取り上げる（図7.8）。商品テーブルは、商品IDと在庫数からなる単純なものだ。サーバは、統合データベース、リモートデータベースサーバ、Mobile Linkサーバの3つがある。運用環境では別々のマシンで稼働させるのが普通だが、本書では同じマシン上で稼働しているとする。

統合データベースとリモートデータベースの作成から始める。

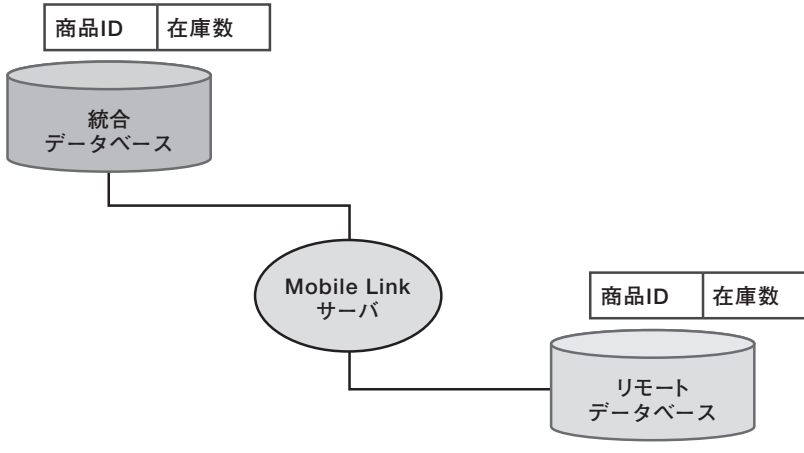


図7.8 : 例題のアーキテクチャ

```
$ cd d:¥asa9book¥mobilelink
$ dbinit consol.db
$ dbinit remote.db
```

両データベースに対して、接続に必要なODBCデータソース名を設定する。デー

タソースに対する接続要求があれば自動的にデータベースが起動し、接続がなくなれば自動的にデータベースが終了する。

```
$ dbdsn -w consol -y -c "uid=dba;pwd=sql;eng=consol;dbf=
d:¥asa9book¥mobilelink¥consol.db"
$ dbdsn -w remote -y -c "uid=dba;pwd=sql;eng=remote;dbf=
d:¥asa9book¥mobilelink¥remote.db"
```

両データベースに対してスキーマを作成する[11]。

```
$ type schema.sql
CREATE TABLE product (
    product_id    VARCHAR(16),
    quantity      INTEGER,
    PRIMARY KEY   (product_id)
);
$ dbisql -c "dsn=consol" schema.sql
$ dbisql -c "dsn=remote" schema.sql
```

統合データベースに対して初期データを3つ挿入する。リモートデータベースは空のままにしておく。

```
$ type data.sql
insert into product (product_id, quantity) values ('apple',  10);
insert into product (product_id, quantity) values ('banana', 20);
insert into product (product_id, quantity) values ('carrot', 30);
$ dbisql -c "dsn=consol" data.sql
```

ここまではMobile Linkとは関係のないデータベースの設定である。次のセクションから、リモートデータベース、統合データベースの順で、Mobile Link同期の設定について見ていく。

11 ここでは、統合データベースとリモートデータベースとで同じスキーマを用いたが、Mobile Linkはスキーマが異なっても同期可能である。

7.2.2 リモートデータベースの同期設定

リモートデータベースに対して、Mobile Link同期の設定を行う。① 同期ユーザ、② パブリケーション、③ サブスクリプションを作成する。Mobile Link用語が一度に出てしまったが、リモートデータベースには3つの項目を設定することに注意する。

① 同期ユーザ

Mobile Link同期の認証としてのユーザ名[12]。Mobile Link同期に使われるもので、データベース接続に用いるユーザとは別のものである。

② パブリケーション

同期対象となるテーブルを指定するもの。リモートデータベース上のテーブルのうち同期対象にしたいものをパブリケーションとして定義する[13]。

③ サブスクリプション

同期ユーザとパブリケーションとを結び付けるもので、Mobile Linkサーバの IPアドレスなどもここで指定する。

```
$ type ml_remote.sql
CREATE SYNCHRONIZATION USER moriwaki;
CREATE PUBLICATION     pub  (TABLE product );
CREATE SYNCHRONIZATION SUBSCRIPTION
     TO pub
    FOR moriwaki
   TYPE tcpip
ADDRESS 'host=127.0.0.1';
$ dbisql -c "dsn=remote" ml_remote.sql
```

同期ユーザの作成

パブリケーションの作成

サブスクリプションユーザの作成

12 ここでは設定しないが、パスワードも設定可能。

13 リモートデータベースと統合データベースとでテーブル名が異なってもMobile Linkは同期可能である。パブリケーションでは、リモートデータベースのテーブル名を指定する。また、テーブル内の一部のカラムを同期対象としたり、WHERE句で同期対象となる行を絞り込むこともできる。

一見、3種類も設定があることは冗長なようだが、これらが柔軟な同期設定を可能にする。複数のパブリケーションと複数のサブスクリプションを作っておけば、同期対象を目的別に使い分けることができるからだ。たとえば、外出時の無線回線では頻繁に更新されるテーブルのみを同期し、社内LANではすべてのテーブルを同期するなど、複数の同期方法が利用可能になる[14]。

7.2.3 統合データベースの同期設定

統合デーベースに対してMobile Linkの設定を行う。同期ロジックとなる同期スクリプトを作成する。スクリプトの説明は後述するので、4つの同期スクリプトを作成したことだけをおさえておこう。

```
$ type ml_consol.sql
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product', 'download_cursor',
   'SELECT product_id, quantity
      FROM product'
);
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product', 'upload_insert',
   'INSERT INTO product (product_id, quantity)
    VALUES (?, ?)'
);
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product', 'upload_delete',
   'DELETE FROM product
     WHERE product_id = ?'
);
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product', 'upload_update',
   'UPDATE product
       SET quantity = ?
     WHERE product_id = ?'
);
$ dbisql -c "dsn=consol" ml_soncol.sql
```

14 このような場合、同期ユーザとパブリケーションとを指定して同期する。残念ながら、サブスクリプションで指定することはできない。

Mobile Link用語がたくさん登場したので、ここまで行った設定項目を整理しておく。

リモートデータベース		統合データベース
・同期ユーザ	同期ユーザ名	4つの同期スクリプト
・パブリケーション	同期対象テーブル	・download_cursor
・サブスクリプション	Mobile Linkサーバ IP など	・upload_insert
		・upload_delete
		・upload_update

7.2.4 Mobile Linkサーバの起動

統合データベースのODBCデータソース名を接続先として、Mobile Linkサーバを起動する。

```
$ dbmlsrv9 -c "dsn=consol" -o mlserver.mls -v+ -dl -za -zu+
```

図7.9 : Mobile Linkサーバの起動

以上でMobile Link同期の環境は整った。いよいよ同期を実行する。

7.2.5 同期の実行

リモートデータベースを指定して同期クライアント（dbmlsyncコマンド）を起動し、同期を開始する。

```
$ dbmlsync -o dbmlsync.mlc -v -c "dsn=remote"
```

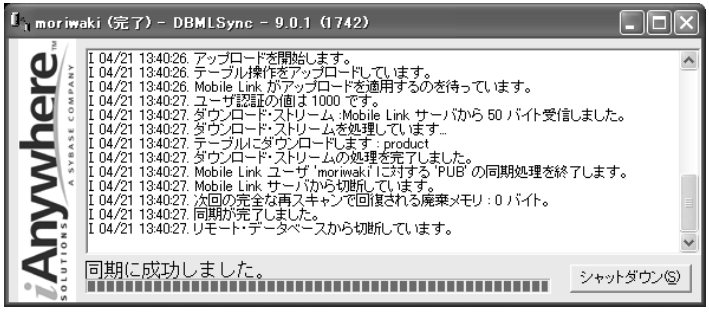


図7.10 : 同期の実行

同期が完了したら、リモートデータベースの状態を確認しよう。統合データベースにあった3件のデータがリモートデータベースにダウンロードされているはずだ。

```
$ dbisql -c "dsn=remote" "SELECT * FROM product"
実行時間 :0.03 秒
product_id quantity
-------------------
apple      10
banana     20
carrot     30
 （最初の 3 ロー）
```

今度は反対に、リモートデータベースを更新して、統合データベースにアップロードしてみよう。リモートデータベースに1件のデータを追加（INSERT）し、1件のデータを更新（UPDATE）する。

```
$ dbisql -c "dsn=remote" "INSERT INTO product values ('durian', 40)"
実行時間 :0 秒
1 レコードが挿入されました。
$ dbisql -c "dsn=remote" "UPDATE product SET quantity=11 WHER
E product_id='apple'"
実行時間 :0 秒
1 レコードが更新されました。
```

再び同期したのち、統合データベースのデータを確認する。1件増えて、1件更新されているはずだ。

```
$ dbmlsync -k -o dbmlsync.mlc -v -c "dsn=remote"
$ dbisql -c "dsn=consol" "SELECT * FROM product"
実行時間 :0.03 秒
product_id quantity
--------------------
apple      11  <-- 更新された
banana     20
carrot     30
durian     40  <-- 増えた
(最初の 4 ロー)
```

以上が同期実行の例である。一連の流れを整理すると、

1. リモートデータベースには、同期ユーザ・パブリケーション・サブスクリプションを設定する
2. 統合データベースには、同期スクリプトを登録する
3. Mobile Linkサーバを起動する
4. 同期クライアントを実行する

次節では、後回しにしていた同期スクリプトの説明をする。

7.3 同期スクリプトと同期イベント

前節では4つの同期スクリプトが登場した。後述するようにほかにもいろいろな種類があるが、この4つがMobile Link同期の基本となる。

- upload_insert
- upload_delete
- upload_update
- download_cursor

名前が「upload_」から始まる3つがアップロード用のスクリプトで、download_cursorがダウンロード用のスクリプトだ。それぞれの意味を順に解説するが、その前に、スクリプトを登録する方法について説明する。

7.3.1 ml_add_table_script ストアドプロシージャ

同期スクリプトの種類にかかわらず、同期スクリプトの登録はml_add_table_scriptストアドプロシージャを利用する。

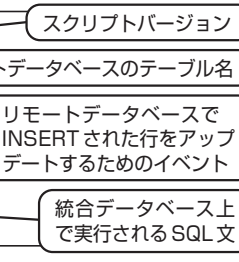
ml_add_table_scriptの引数は4つの項目がある。

表7.1：ml_add_table_scriptの引数

引数名	説明
スクリプトバージョン	複数のスクリプトをグループ化するグループ名[15]
テーブル名	パブリケーションで指定した同期対象のテーブル名。リモートデータベースの各テーブルに対して一致したものが実行される
同期イベント名	SQLスクリプトが実行されるための同期イベント
SQLスクリプト	SQLで書かれた同期ロジック

upload_insertスクリプト（再掲）

```
CALL ML_ADD_TABLE_SCRIPT(
'default',
'product',
'upload_insert',
   'INSERT INTO product (product_id, quantity)
    VALUES (?, ?)'
);
```



Mobile Linkでは同期対象のテーブル名を統合データベースとリモートデータベースとで同一にする必要はないので、統合データベースではなくリモートデータベースのテーブル名を指定する。このテーブル名がパブリケーションに含まれていれば、同期時のイベントに従ってSQLスクリプトが実行される（このSQL内で統合データベースのテーブル名を指定する）。

では、4つの同期イベント名に対応するSQLスクリプトの解説に入ろう。

15　「バージョン」という言葉が入っているので誤解されやすいが、グループ化する文字列である。数字にする必要もないし、どれかが古くてどれかが新しいという意味もシステム上はない（ユーザがそのように意味付けするのは自由）。同期クライアントでスクリプトバージョンを指定して同期することで、同期ロジックを同期ごとに切り替えることができる。

Column Sybase Centralによるスクリプトの登録

Sybase Centralを用いてGUIでスクリプトを登録することができる。Mobile Linkの統合データベースに対する設定は、Mobile Linkプラグイン経由で行うので注意してほしい。

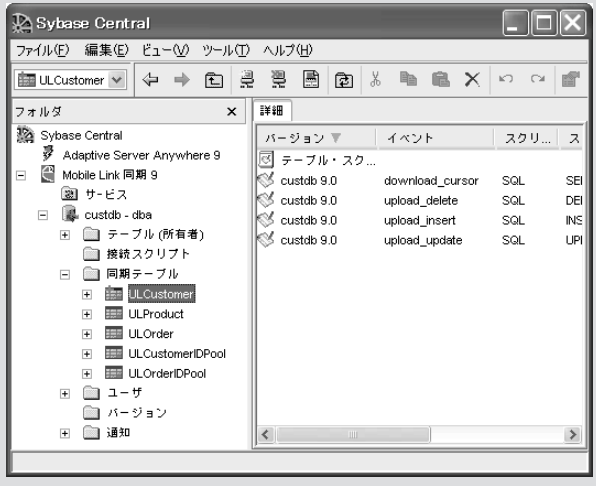


図7.11 : Sybase Centralによるスクリプトの登録

7.3.2 アップロードストリーム

名前が「upload_」から始まる3つのスクリプトは、アップロードストリームに対する同期ロジックである。同期クライアントからMobile Linkサーバに送信されたアップロードストリームは、追加された行・更新された行・削除された行に仕分けされ、3つのスクリプトを雛形としたSQL文として統合データベースに送信される（図7.12）。

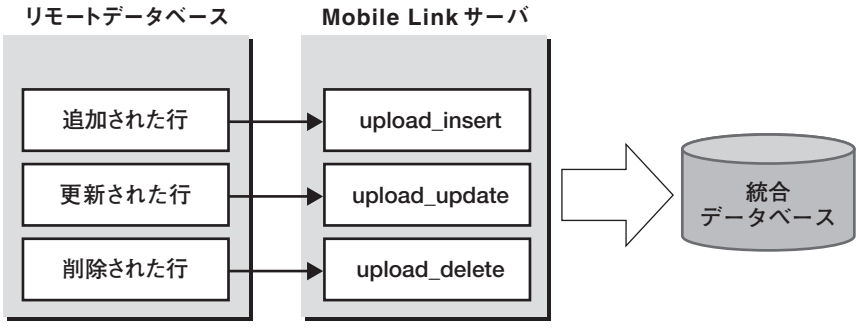


図7.12 : upload_insert・upload_update・upload_deleteイベント

■ upload_insertイベント

upload_insertのSQL文（再掲）

```
INSERT INTO product (product_id, quantity)
VALUES (?, ?)
```

スクリプト内の「?」はプレースホルダで、Mobile Linkサーバの機能により、アップロードされたデータに置き換わる。プレースホルダの順序は、リモートデータベースのテーブルにおいてCREATE文で定義された順序だ。

プレースホルダの置換を先の例で見ると、追加された「durian」に対し、upload_insertスクリプトを雛形として、次のSQL文が統合データベースへ送信される。

```
INSERT INTO product (product_id, quantity)
VALUES ('durian', 40)
```

■ upload_updateイベント

upload_updateのSQL文（再掲）

```
UPDATE product
   SET quantity = ?
 WHERE product_id = ?
```

WHERE句には主キーを含め、それ以外のカラムをSET句で更新する。プレースホルダの置換は、主キーの選別も含めてMobile Linkサーバが自動的に行う。先の例では、「apple」のデータを更新したが、upload_updateスクリプトに従って、次のSQL文が統合データベースで実行される。

```
UPDATE product
   SET quantity = 11
 WHERE product_id = 'apple'
```

■ upload_deleteイベント

upload_deleteのSQL文（再掲）

```
DELETE FROM product
 WHERE product_id = ?
```

WHERE句には主キーを含める。削除された行の主キーが自動的に置換される。

7.3.3 ダウンロードストリーム

■ download_cursor

download_cursorは、ダウンロードストリームを作り出すスクリプトである。ここに書かれたSQL文が統合データベース上で実行され、その結果セットがリモート

データベースに返信される（図7.10）。

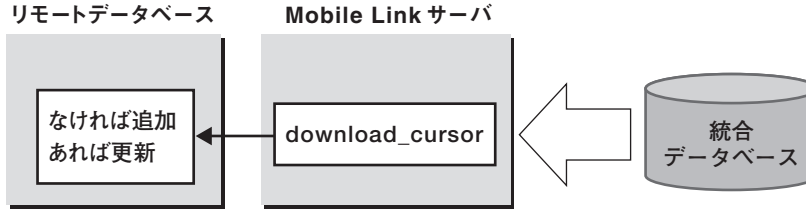


図7.10 : download_cursorイベント

先の例では、次のようにproductテーブルの全行を取得するSQL文になっている。よって、同期時には、productテーブル全体がリモートデータベースに返信される。

download_cursorのSQL文（再掲）

```
SELECT product_id, quantity
  FROM product
```

同期クライアントは、受信した行とリモートデータベース上の行とを（主キーで）比較し、既存の行があれば受信した行で更新し、なければ受信した行を追加する。

■ カラムによる絞り込み

download_cursorの結果セットにあるカラムだけがダウンロードされる。例ではもともと2つしかカラムがなかったので判別しづらかったが、たとえば、統合データベースのproductテーブルが4つのカラムを持っていて、リモートデータベースのものよりも多いとすると、次のようになる。

- 統合データベース　—— product_id, quantity, product_name, price
- リモートデータベース —— product_id, quantity

これらを同期するためのdownload_cursorスクリプトのSQLは、次のものでよい。

```
SELECT product_id, quantity
  FROM product
```

■ 行による絞り込み

SELECT文でWHERE条件を追加して、特定のデータだけをダウンロード対象としてもよい。たとえば、在庫数の小さなものだけダウンロードしたいならば、以下のようにする。

```
SELECT product_id, quantity
  FROM product
 WHERE quantity < 100
```

7.3.4 ダウンロード時のプレースホルダ

アップロード時のスクリプトでは、アップロードされるデータをプレースホルダとして指定できた。一方、ダウンロード時のスクリプトにもプレースホルダがあり、Mobile Linkの機能によって定められた値に置換される。

ダウンロード時のスクリプトでは、3つのプレースホルダが利用可能だ（表7.2）。

表7.2 : download_cursorスクリプトのプレースホルダ

プレースホルダの順番	内容
1番目	前回同期成功時刻（ダウンロード時刻）
2番目	Mobile Linkユーザ名
3番目	同期テーブル名

注：同期テーブル名を示す3番目のプレースホルダは、download_cursorイベントを含む一部のイベントでは使用できない。

Mobile Linkユーザ名を利用する例を見ていこう。ここで各リモートデータベースは、自身のMobile Linkユーザに関連するデータのみを持つとする[16]。Mobile Linkユーザ名で条件付けすることで、テーブルすべてのデータではなく、リモートデータベース（＝Mobile Linkユーザ）ごとに必要なデータを絞り込める。たとえば、統合データベース上に顧客テーブルがあり、Mobile Linkユーザごとに顧客が分かれているとする（図7.11）。

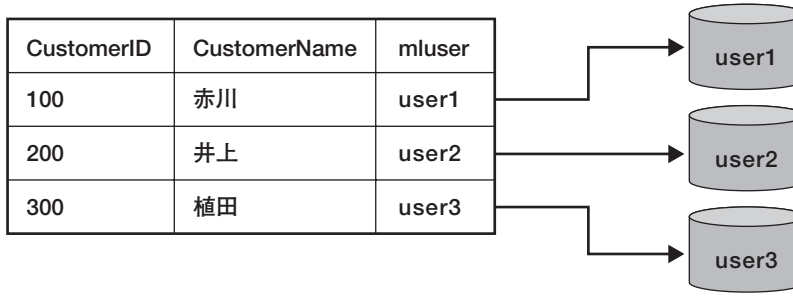


図7.14：Mobile Linkユーザごとにダウンロードする例

download_cursorスクリプトは、次のようになる。

```
SELECT CustomerID, CustomerName
  FROM Customer
 WHERE ? IS NOT NULL
   AND mluser = ?
```

1つ目のプレースホルダ：最終同期時刻

2つ目のプレースホルダ：Mobile Linkユーザ名

この例では最終同期時刻の値は利用しないのだが、プレースホルダは表の順番通りに置換されるため、2番目の同期ユーザ名を利用するには、1番目のプレースホルダも書いておく必要がある。そこで、NULLと比較するダミーの条件を置いた。

16　このような過程はMobile Linkの利用では一般的だ。本章の最初で適用例を挙げたように、営業スタッフは自身の顧客データのみに関心がある。

7.3.5 JavaやNETによる同期ロジック

本書では同期ロジックとしてSQL文やストアドプロシージャを使用するが、JavaやNETでも同期ロジックを記述できる。JavaやNETで書く利点としては、言語が持つライブラリを利用して複雑なロジックを構築できる点だ。本書ではコーディングに触れないが、「7.4.6 同期ユーザのカスタマイズ認証」で、外部システムと連携する設定例を紹介する。

7.3.6 同期イベント

同期スクリプトはイベントハンドラでもある。Mobile Linkには同期処理の過程におけるさまざまなイベントが存在し、それに対するイベントハンドラ（＝同期スクリプト）が定義されていれば、それが実行される。

たとえば、download_cursorイベントは、ダウンロードストリームを作成しようとするときに発生するイベントである。本書の例の場合、download_cursorスクリプトが存在したのでそれが実行され、productテーブルのデータがリモートデータベースに伝わった。同様に、upload_insertイベントは、アップロードストリームのデータのうち追加されたデータを処理する際に発生するイベントである。本書の例ではupload_insertスクリプトが存在したのでそのスクリプトが実行され、追加された行が統合データベースに伝わった。

ここで取り上げた4つのイベント以外にもたくさんのイベントがある。それらに応じたスクリプトを書けば、複雑な同期ロジックを構築可能だ。イベントと同期処理の概略を擬似コードで以下に示す。イベント名はボールド体で示した。なお、この擬似コードは全体の一部であり、省略されている部分があるので、詳細はマニュアルを参照していただきたい。

まず、Mobile Linkサーバが起動してから終了するまでの全体像を示す。

```
CONNECT to consolidated database
begin_connection_autocommit
begin_connection
COMMIT
<同期クライアントからの同期処理を繰り返す>
end_connection
COMMIT
DISCONNECT from consolidated database
```

Mobile Linkサーバ起動時に実行したい処理があれば、この接続系のイベントを使うことになる。<同期クライアントからの同期処理を繰り返す>の部分で、同期クライアントからのリクエストを待ち受け実行する。

1つの同期セッションは、次のように処理される。

```
# 接続開始
begin_synchronization
# アップロードストリームの開始
begin_upload
  for each table being synchronized {
    begin_upload_rows
    <INSERTされた行を統合データベースへupload>
    <UPDATEされた行を統合データベースへupload>
    end_upload_rows
  }
  for each table being synchronized IN REVERSE ORDER {
    begin_upload_deletes
    <DELETEされた行をupload>
    end_upload_deletes
  }
  end_upload
  upload_statistics
  COMMIT
# ダウンロードストリームの開始
begin_download
for each table being synchronized {
  begin_download_deletes
  end_download_deletes
  begin_download_rows
```

```
  <リモートデータベースへdownload>
  end_download_rows
}
modify_next_download_timestamp
end_download
COMMIT
# 接続終了
end_synchronization
```

Column　クライアント側の同期イベント

Mobile Linkサーバの同期イベントを説明してきたが、実は、同期クライアント（リモートデータベース）側にも同期イベントがあり、アップロードやダウンロード時のイベントフックを用いて、エラー処理などを制御できる。こちらも擬似コードで概略を示す（省略あり）。

```
sp_hook_dbmlsync_abort
# 同期の開始
sp_hook_dbmlsync_begin
# アップロードストリームの開始
sp_hook_dbmlsync_logscan_begin
sp_hook_dbmlsync_logscan_end
sp_hook_dbmlsync_upload_begin
sp_hook_dbmlsync_upload_end
# ダウンロードストリームの開始
sp_hook_dbmlsync_download_begin
for each table
   sp_hook_dbmlsync_download_table_begin
   sp_hook_dbmlsync_download_table_end
next table
sp_hook_dbmlsync_download_end
# 同期の終了
sp_hook_dbmlsync_end
sp_hook_dbmlsync_process_exit_code
```

7.3.7 同期スクリプトの自動生成

Mobile Linkサーバには同期スクリプトを自動生成する機能がある。-zaオプションを付けてMobile Linkサーバを起動しておくと、初回同期時に以下のイベントスクリプトが自動生成されて統合データベースに保存され、そのスクリプトに従って同期が完了する[17]。システム開発時には、自動生成された結果を出発点として、改造していけばよい。

- upload_insert
- upload_update
- upload_delete
- download_cursor

自動生成されるにはいくつかの前提条件がある。

- 既存の同期スクリプトがないこと
  既存の同期スクリプトが1つでも存在すると、自動生成されない（当然上書きもされない）。自動生成したいならば、既存の同期スクリプトをすべて削除しておく。
- 同期クライアントで、カラム名を付けて同期すること
  同期クライアントがアップロードするのはデータだけであるが、オプションスイッチによってカラム名も送ることができる。自動生成にはカラム名が必要なため、このオプションスイッチを付けて同期クライアントを実行する。オプション名はSendColumnNamesであり、以下のように実行する。

17　というわけで、先ほどのスクリプトをすべて手入力する必要はなく、やってしまった方、説明が遅れてすみません。

```
dbmlsync -e "SendColumnNames=on"
```

7.3.8 Mobile Link モニタ

Mobile Link モニタとは、Mobile Link サーバの状態をリアルタイムにモニタリングするGUIツールだ（図7.15）。同期の所要時間などの情報を細かく把握できるので、同期処理のチューニングにも役立つ。Mobile Link モニタを起動し、Mobile Link サーバに接続すれば、その時点からずっとモニタリングしてくれる。

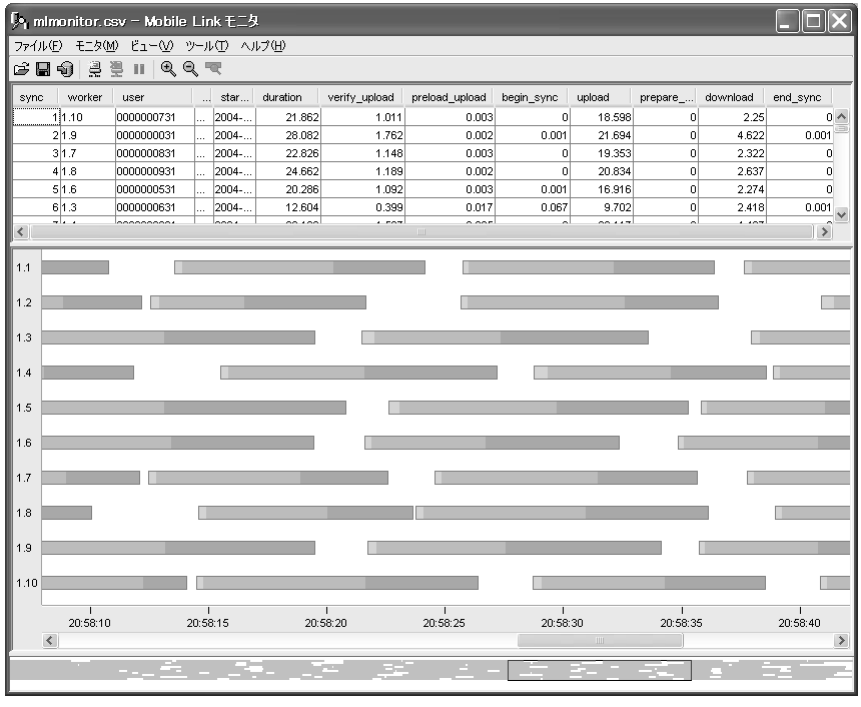


図7.15 : Mobile Link モニタ

図7.15の上部のグリッドでは、1行が1同期セッションを意味し、各過程の処理時

間が示される。下部のグラフは、1行がMobile Linkサーバの1スレッドを意味し（合計10スレッド動作している）、1つのバーが同期セッションを表す。各過程はバー内で色分けされている。このグラフは、リアルタイムに右から左へアニメーションする。カラーや動画でお見せできないのが残念だ。

7.4 さまざまな同期ロジックのレシピ

Mobile Linkの仕組みと簡単な同期ロジックを見てきた。しかし、このままでは統合データベースのデータが同期済みのものも含めて毎回ダウンロードされてしまい、効率が悪い。競合解決もしていない。本節では、応用的な同期ロジックの実装例を見ていく。

これから挙げる方法は定石ではあるがあくまでも例であり、ほかにもさまざまな実装方法がある。Mobile Linkの同期ロジックは柔軟であり、同期イベントと統合データベース側のストアドプロシージャなどとを組み合わせて複雑な同期ロジックを構築可能だ。どのようなスクリプトにするかは、開発コスト、運用管理コスト、通信コストなどとの兼ね合いになるので、システムごとに判断する必要がある。

7.4.1 アップロードのみ：UploadOnly

目的：リモートデータベースのデータを統合データベースにアップロードはしたいが、ダウンロードはしたくない

同期クライアントのオプションで設定可能だ。アップロード用のスクリプトがあったとしても、同期クライアントでオプションを設定すれば、アップロードストリームをスキップすることができる。

```
$ dbmlsync -e "UploadOnly=on"
```

7.4.2 ダウンロードのみ：DownloadOnly

目的：統合データベースのデータをリモートデータベースにダウンロードはしたいが、アップロードはしたくない

同期クライアントのオプションで設定可能だ。ダウンロード用のスクリプトがあったとしても、同期クライアントでオプションを設定すれば、ダウンロードストリームをスキップすることができる。

```
$ dbmlsync -e "DownloadOnly=on"
```

7.4.3 差分同期

目的：統合データベースとリモートデータベースのそれぞれで、変更のあったデータだけを同期したい

差分同期とは、前回同期成功時以降に更新されたデータ（以後、更新データと呼ぶ）だけ同期することである。

アップロードストリームでは、自動的に更新データのみが送られる。同期クライアントがリモートデータベースのトランザクションログを調べて、更新データのみを送信するからだ。ただし、競合解決用に更新前のデータも送られる。詳細は次節で説明する。

一方、ダウンロードストリームでは自動的な仕組みはなく、ユーザがロジックやスキーマを構築することになる。Mobile Linkサーバは ODBC経由で統合データベースに接続するので、トランザクションログのようなベンダー固有の特別なシステム情

報に頼らない。では、どのように構築すべきなのか順を追って見ていこう。

1. 最終更新時刻カラムの追加

統合データベースの同期対象テーブルに最終更新時刻カラムを加え、行を変更するたびにこの時刻を更新するようにする。そのためには、統合データベースを使用するアプリケーション側で実装してもよいし、トリガなどを使って統合データベース上で自動化してもよい。なお、統合データベースがSQL Anywhereであれば、カラムのデフォルト値としてDEFAULT TIMESTAMPを指定すれば自動的に時刻が更新される。

具体的に見ていこう。「7.2 Getting Started」での例をベースとして、差分更新に変えていく。まず、統合データベースのスキーマを変えて、最終更新時刻カラムlast_modifiedを追加する。このカラムにはデフォルト値が指定されているので、SQL Anywhereの機能により、行を修正するたびに自動的に最終更新日時が更新されていく。

```
CREATE TABLE product (
    product_id     VARCHAR(16),
    quantity       INTEGER,
    last_modified  TIMESTAMP DEFAULT TIMESTAMP,
    PRIMARY KEY (product_id)
);
```

2. download_cursorスクリプトの修正

次に、Mobile Linkのdownload_cursorスクリプトを修正する。前回同期成功時刻が自動的に記録されるので、この時刻とカラムの最終更新時刻とを比較すれば、更新されたデータのみを抽出できる。

前回同期成功時刻はプレースホルダとして指定する。WHERE条件にある「?」がプレースホルダだ。

```
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product', 'download_cursor',
   'SELECT product_id, quantity
      FROM product
     WHERE last_modified >= ?'
);
```

■ 同期成功時刻はどのマシンの時刻か

同期成功時刻は統合データベース上で計測され、統合データベース上のML_SUBSCRIPTIONテーブルやリモートデータベース上のSYSSYNCテーブルに記録される。統合データベース上で計測された時刻が統合データベース上で比較されるので、統合データベースとリモートデータベースとで時計が一致していなくても問題ない。

7.4.4 競合解決

目的：競合解決をしたい

データが分散する環境では、データの競合が発生する場合がある。まず、競合とはどういうものかを説明し、次にMobile Linkでの解決方法を示す[18]。

■ 競合とは何か

統合データベースに対して、営業のAさんとBさんの2人が同期する状況を想定する（図7.16）。

最初、製品XYZの在庫は10個あり、Aさんと Bさんがそれぞれ同期を行った。このとき、統合データベース、Aさんのデータベース、Bさんのデータベースのいずれも、製品XYZの在庫は10個である。これが初期状態だ。

18　言うまでもなく、競合しないシステム設計をまず心がけるべきである。競合解決は複雑になりがちであり、同期パフォーマンスも低下しやすい。

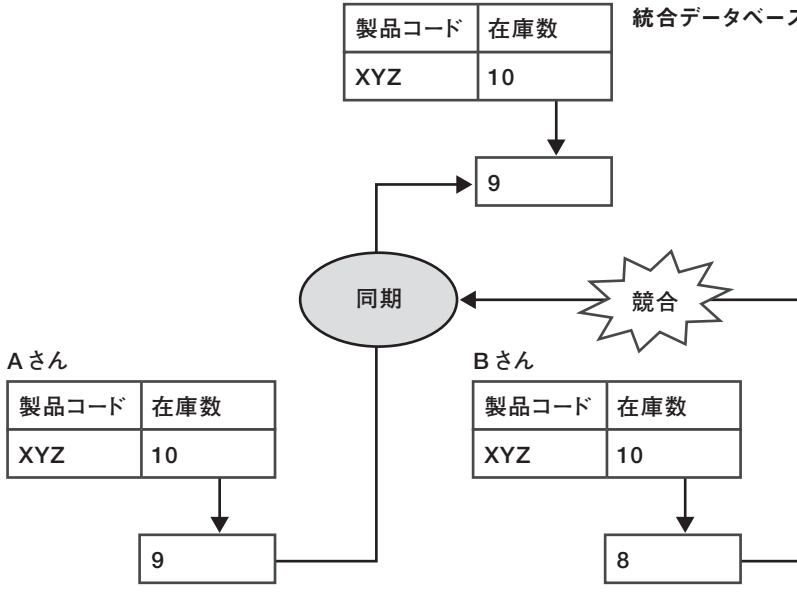


図7.16：競合発生の例

Aさんは製品を1つ売り、自身のデータベースで在庫を1つ減らして9にした。時を同じくして、優秀なBさんは製品を2つ売り、在庫数を2つ減らして8にした。

しばらくして、Aさんは同期を行った。この同期完了時点で、統合データベースとAさんのデータベースとの両方で、製品XYZの在庫は9個となる。ここまでは問題ない。

さて、ここでBさんが同期をしたらどうなるであろうか。Bさんのデータベースのデータで統合データベースをそのまま上書きしたのでは製品の在庫数は8になってしまい、現実と合わない。

これがデータの競合である。データの競合は、本来同じ実体であるはずの行に対して、複数のコピー先（AさんとBさんのデータベース）で更新してしまうときに発生する。競合したデータを解決するルールをあらかじめ定めておかないと、システムは全体として整合性のないものになってしまう。

このように競合解決では、競合を検出する過程と検出された競合を解決する過程とに分かれる。ただし、それぞれの過程におけるルールは定型のものがあるわけではなく、個々のシステムで定義する問題である。何をもって競合とするか、その競合をどう措置するかはシステム要件による。このルールを競合検出ロジックと競合解決ロジックと呼ぶことにしよう。Mobile Linkではこれらのロジックを柔軟に記述できるので、さまざまなビジネス要件に適合する同期ソリューションとなる。

■ 競合検出方法

実は、アップロード時に送られるデータはリモートデータベース内の最新のデータだけでなく、前回同期成功時点のデータも含む。前者を新データ、後者を旧データと呼ぼう。Mobile Linkサーバは、リモートデータベースから受け取った旧データと統合データベース上のデータとを比べることで競合を検出する。比較した結果に差異があれば競合発生とし、競合イベントを発生させる。

■ Mobile Linkでの競合解決方法

競合イベントに応じた競合スクリプトを定義しておき、それが実行されることで競合を解決する。Mobile Linkサーバは、スクリプトのロジックにかかわらず、そのスクリプトがエラーを出さずに正常終了したことをもって競合解決が行われたとみなす。

いろいろな解決方法があるが、代表的なものを3つ挙げる。

- クライアント優先の解決

  リモートデータベースのデータで統合データベースのデータを強制的に上書きする。この動作は、Mobile Linkのデフォルトの動作であるので、クライアントのデータを常に優先するのであれば、競合検出ロジックも競合解決ロジックも必要ない。

- サーバ優先の解決

  統合データベースのデータを残し、リモートデータベースのデータは破棄する。競合検出のロジックを定義し、競合解決として統合データベースへは何もしないようにすればよい。

- 3つのデータを比較するカスタム解決

  統合データベースのデータと新データと旧データの3つのデータが参照可能なので、これらから独自のロジックを構築する。先の例であれば、Bさんがアップロードしたときには、8－(10－9)＝7として統合データベースを更新することも可能だ。この実装例を次に紹介する。

競合解決した結果は統合データベースに更新し、ダウンロード時にその値がリモートデータベースに伝わる。こうして、同期後統合データベースとリモートデータベースとで整合性のとれた状態となる。

Column　追加時の競合

競合は本来更新処理において発生し、行の追加時では起こり得ない。行の追加時には主キーを必要とするが、主キーはその性質上、実体固有の識別子であり、別々の場所で1つの実体が生み出されるということはない。逆に言うと、重複して生み出されるような属性は主キーとしてふさわしくない。

とはいえ、そのようなシステム設計にせざるを得ない状況もあるだろう。そのような場合、主キーの重複を検出して解決するロジックを同期スクリプトに書けばよい。後述する強制的競合解決の仕組みも参考になるだろう。

なお、SQL Anywhereには、INSERT ON EXISTING UPDATEという便利な構文がある。この構文を使うと、同じ主キーの既存の行に追加しようとしたとき、自動的に更新処理となる。

■ カスタム解決の具体例

それでは、カスタム解決を実装してみよう。在庫数が競合した場合は、アップロードされた新旧データと統合データベースのデータから適切な在庫数を計算する（つまり、8 − (10 − 9) = 7とする）。

競合検出はupload_fetchスクリプトで行う。このスクリプトは、新データが統合データベースへ更新される前に実行される。Mobile Linkサーバは、upload_fetchを実行して統合データベースの行を取得し、旧データと比較する。両者が異なっていれば競合イベントを発生させ、同じならば通常どおりの処理を続ける（新データによって統合データベースが更新される）。

```
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product', 'upload_fetch',
   'SELECT product_id, quantity
      FROM product
     WHERE product_id = ?'
);
```

いよいよ競合解決であるが、そのためには、統合データベース上で新旧データが参照可能でなければならない。競合解決のスクリプトは、Mobile Linkサーバ上ではなく統合データベース上で実行されるからだ[19]。ここでは、統合データベースに対してテンポラリテーブルを1つ作成し、新旧のフラグ（OldとNew）を付けて挿入することにする[20]。

```
CREATE GLOBAL TEMPORARY TABLE product_conflict (
    product_id    VARCHAR(16),
    row_type      CHAR(1),
    quantity      INTEGER,
    PRIMARY KEY (product_id, row_type)
);
```

19 Mobile Linkサーバの役割は、あくまでも、SQLスクリプトを統合データベースに送信することだ。

20 SQL Anywhereのデフォルト動作では、テンポラリテーブルのデータはコミット時に失われる。Mobile Linkでは、アップロードストリームの終わりに一度コミットされるので、ここでデータが削除される。

新旧のデータを統合データベースに送信するためのイベントは、upload_new_row_insertとupload_old_row_insertだ。これらは、競合が検出されたあとで実行される。

```
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product',
   'upload_old_row_insert',
   'INSERT INTO product_conflict (product_id, row_type, quantity)
    VALUES (?, ''O'', ?)'
);
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product',
   'upload_new_row_insert',
   'INSERT INTO product_conflict (product_id, row_type, quantity)
    VALUES (?, ''N'', ?)'
);
```

残るは競合解決のロジックだ。競合解決イベントであるresolve_conflictスクリプトにロジックを書く[21]。Mobile Linkサーバは、このスクリプトの正常終了をもって競合が解決されたとみなす。

```
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product', 'resolve_conflict',
   'UPDATE product
       SET p.quantity = new_row.quantity -
                        (old_row.quantity - p.quantity)
      FROM product p,
           product_conflict old_row,
           product_conflict new_row
     WHERE old_row.product_id = p.product_id
       AND old_row.product_id = new_row.product_id
       AND old_row.row_type = ''O''
       AND new_row.row_type = ''N'' '
);
```

イベントがたくさん出てきたので、ここでまとめておこう。

21　ここではSQL AnywhereのSQL方言で記述した。ほかのデータベースを利用していて一文で書けなければ、ストアドプロシージャで記述し、それを呼ぶ。

データを更新しようとすると、Mobile Linkサーバはupload_fetchを実行する。競合検出されなければ、upload_updateを実行する。競合が検出されれば、upload_new_row_insertとupload_old_row_insertとを実行し、resolve_conflictで競合解決する（図7.17）。

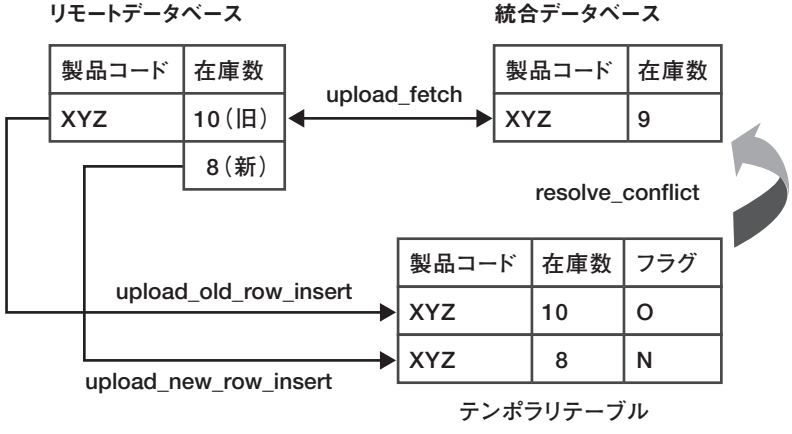


図7.17：競合解決のプロセス

では、実際に同期してみよう。初期状態では、統合データベースとリモートデータベースそれぞれに3件のデータがある。

```
$ dbisql -c "dsn=consol" "SELECT * FROM product"
実行時間 :0.03 秒
product_id quantity last_modified
-----------------------------------------------
apple      10       2004-04-08 16:43:02.485000
banana     20       2004-04-08 16:43:02.485001
carrot     30       2004-04-08 16:43:02.485002
(最初の 3 ロー)
$ dbisql -c "dsn=remote" "SELECT * FROM product"
実行時間 :0.02 秒
product_id quantity
-------------------
```

```
apple      10
banana     20
carrot     30
(最初の 3 ロー)
```

統合データベースのappleの在庫数を1つ減らし、一方、リモートデータベースのappleの在庫数を2つ減らす。これでappleの在庫数は競合状態になった。

```
$ dbisql -c "dsn=consol" "UPDATE product SET quantity = quant
ity - 1 WHERE product_id = 'apple'"
実行時間 :0 秒
1 レコードが更新されました。
$ dbisql -c "dsn=remote" "UPDATE product SET quantity = quant
ity - 2 WHERE product_id = 'apple'"
実行時間 :0.01 秒
1 レコードが更新されました。
```

同期して、競合解決の結果を見る。

```
$ dbmlsync -k -o dbmlsync.mlc -v -c "dsn=remote"
$ dbisql -c "dsn=consol" "SELECT * FROM product"
実行時間 :0.02 秒
product_id quantity last_modified
-----------------------------------------------
apple      7        2004-04-08 16:45:30.398000
banana     20       2004-04-08 16:43:02.485001
carrot     30       2004-04-08 16:43:02.485002
(最初の 3 ロー)
$ dbisql -c "dsn=remote" "SELECT * FROM product"
実行時間 :0.031 秒
product_id quantity
-------------------
apple      7
banana     20
carrot     30
(最初の 3 ロー)
```

たしかに、appleの在庫数は7個になっている。

■ upload_fetchによる競合検出の注意点

upload_fetchによる競合検出は、実装は簡単であるが同期パフォーマンスの低下を招く恐れがある。upload_fetchが定義されたテーブルのデータを同期する際は、アップロードされる行ごとにupload_fetchスクリプトが実行される。たとえば、1,000件のデータをアップロードしたとすると、1,000回upload_fetchが実行される。さらに、そのうち100件が実際に競合したとすると、100回ずつupload_new_row_insertとupload_old_row_insertが実行され、100回resolve_conflictが実行される。これは、Mobile Linkサーバと統合データベース間で、大量のクエリが発生することを意味する。

このような事態を避けるには、upload_fetchやresolve_conflictを使わない「強制的競合解決（Forced Conflict Resolution）」という方法がある。それを次に説明する。

7.4.5 強制的競合解決

強制的競合解決とは、アップロードされたすべての行をいったん統合データベースに格納し、同期の終わりにスクリプトを実行して、いっぺんに競合解決してしまう方法である。upload_fetchをしないため、一般に同期パフォーマンスの向上が見込める。

なお、「強制的」というネーミングが紛らわしいので誤解しないでいただきたい。この方法では、Mobile Linkのシステム上、競合をしていない行も含めてすべての行が「競合を起こした」と認識されるのでこのような名称が付いている。前述したupload_fetchのときと同様に正しく行われる。決して無理矢理という意味ではない。

強制的競合解決に切り替えるには条件がある。それは、upload_insert・upload_

update・upload_deleteの3つのスクリプトが存在しないことだ。これらが定義されていなければ、競合の有無にかかわらずアップロードされたすべての行に対して、upload_new_row_insertとupload_old_row_insertが実行される[22]。こうして、統合データベース上に新旧データが置かれることになるので、競合解決の有無にかかわらずバッチ処理すればよい。通常、新旧データはテンポラリテーブルに置き、end_uploadというアップロード終了イベントで、テンポラリテーブルにたまったデータをベーステーブルに反映させる。

具体的に見ていこう。競合解決の例の続きとして進めていく。同期スクリプトは、download_cursor・upload_new_row_insert・upload_old_row_insertを残して、ほかのスクリプトは削除する。product_conflictテンポラリテーブルはここでも使用するのでそのままでよい。

次に、テンポラリテーブルにたまったデータを一度に処理するストアドプロシージャresolve_productを作成する。

```
CREATE PROCEDURE "DBA"."resolve_product" ()
BEGIN
  -- Inserted data
  INSERT INTO product (product_id, quantity)
  SELECT product_id, quantity
    FROM product_conflict n
   WHERE n.row_type = 'N'
     AND NOT EXISTS (
               SELECT product_id
                 FROM product_conflict o
                WHERE o.row_type = 'O'
                  AND o.product_id = n.product_id);

  -- Deleted data
  DELETE
    FROM product
```

22　定義されていると、前述したようにupload_fetchで競合検出されたときに限り、upload_new_row_insertとupload_old_row_insertとが実行される。

```
    WHERE product_id IN (
            SELECT product_id
              FROM product_conflict
             WHERE row_type = 'O'
            EXCEPT
            SELECT product_id
              FROM product_conflict
             WHERE row_type = 'N');

  -- Conflicting data
  UPDATE product
     SET p.quantity = new_row.quantity - (old_row.quantity -
p.quantity)
    FROM product p,
         product_conflict old_row,
         product_conflict new_row
   WHERE old_row.product_id = p.product_id
     AND old_row.product_id = new_row.product_id
     AND old_row.row_type = 'O'
     AND new_row.row_type = 'N'
END;
```

この処理は、追加・削除・更新の3つの処理に分かれている。product_conflictテーブルにおいて、以下のような処理がそれぞれ行われる。

- 新データのみの商品は競合しておらず、挿入されたものであるので、統合データベースへ挿入する。
- 旧データのみの商品は競合しておらず、削除されたものであるので、統合データベースで削除する。
- 新旧両方のデータがある商品は、競合にかかわりなく、在庫数の差を統合データベースへ更新する（この例では、競合している場合もそうでない場合も同じロジックで計算可能）。

end_uploadイベントでこのストアドプロシージャを実行すれば、競合解決も含めてリモートデータベースのデータが統合データベースに反映される。

```
CALL ML_ADD_TABLE_SCRIPT( 'default', 'product', 'end_upload',
   'CALL resolve_product'
);
```

7.4.6 同期ユーザのカスタマイズ認証

Mobile Linkは、同期ユーザ名とパスワードによる認証に対応している。しかし、既存の認証手段がすでに存在する場合、Mobile Link用のパスワードが新たに増えることを避けたいこともあるだろう。Mobile Linkの認証方法をカスタマイズすれば解決できる。

カスタマイズ認証を行うには、authenticate_userイベントを利用する。プレースホルダとして、アップロードされてきたユーザ名やパスワードなどが渡されるので、それらを既存システムのものと比較し、成功・失敗などの認証ステータスを返せばよい。この実行結果がMobile Linkの認証結果となる。

カスタマイズ認証の代表的な例をいくつか紹介しよう。

■ 統合データベースに既存のID・パスワード情報がある場合

既存のID・パスワードがmy_userテーブルにあるとしよう。このテーブルに対してMobile Linkの認証を行う。

```
CREATE TABLE my_user (
    userid       VARCHAR(16),
    password     VARCHAR(16),
    PRIMARY KEY  (userid)
);
```

認証のためのストアドプロシージャmy_authを作成する。

```
CREATE PROCEDURE my_auth(inout @auth_status  integer,
                         in    @user_name    varchar(128),
                         in    @password     varchar(128),
                         in    @new_password varchar(128))
begin
  if exists
  ( SELECT * FROM my_user
    WHERE userid   = @user_name
      AND password = @password)
  then
    -- OK
    set @auth_status = 1000;
  else
    -- NG
    set @auth_status = 4000;
  end if
end;
```

認証ステータスコードはMobile Linkで定められている値を使用する。なお、ストアドプロシージャの引数として新パスワードの情報があるが、これはパスワードを変更するためのものであり、この例では扱いを省略した。

authenticate_userイベントでは、プレースホルダとして4つのパラメータ（認証ステータス・ユーザ名・パスワード・新パスワード）が渡されるので、このmy_authストアドプロシージャを実行する。

```
CALL ml_add_connection_script(
   'default', 'authenticate_user', 'call my_auth (?, ?, ?, ?)');
```

Column　ID・パスワード以外の情報

同期クライアントはID・パスワード以外にも任意の文字列を認証用に渡すことができるので、このパラメータをカスタム認証に組み込むことができる。このパラメータはdbmlsyncの -apオプションで指定し、authenticate_parametersイベントで参照する。

■ 外部システムの認証システムを利用する場合

Mobile Linkの同期ロジックはJavaや.NETでも記述可能なので、外部システムとの連携も可能だ。Mobile Linkでは、POP3、IMAP、LDAPの認証を利用するモジュールがあらかじめ提供されている。これら以外の外部システムを利用する場合は、必要なコードを実装すればよい。

POP3認証を例にして設定方法を示す。まず、authenticate_userイベントに対して、Javaのモジュールを指定する。

```
call ml_add_java_connection_script(
  'MyVersion'
  'authenticate_user'
  'ianywhere.ml.authentication.POP3.authenticate' )
```

POP3サーバのプロパティは、Mobile Linkのシステムテーブルに設定する。

```
call ml_add_property('ScriptVersion', 'default', 'mail.pop3.host',
'192.168.0.1' );
call ml_add_property('ScriptVersion', 'default', 'mail.pop3.port',
'110' );
```

7.4.7 統合データベースでの削除処理の扱い

リモートデータベースでデータが削除されたとしても、トランザクションログを調べる同期クライアントは、そのデータをMobile Linkサーバに伝えることができる。しかし、統合データベース上でデータが削除されてしまうと都合が悪い。Mobile Linkサーバは、リモートデータベースで削除すべき行をSELECT文で抽出して伝えるのだが、統合データベースで行が削除されてしまうと、その行を感知できないからだ。削除が起こったことをリモートデータベースに伝えるにはどうすればよいのだろうか。

そのためには、統合データベース上では、物理削除ではなく論理的に仮削除を行えばよい。2つの実装方法がある。

1. 削除フラグによる論理削除

   テーブルに削除フラグを立てるカラムを別途用意して、削除フラグの有無で削除を判断する方法である。このとき、upload_delete・download_cursor・download_delete_cursorスクリプトに注意する。

   - upload_deleteスクリプトでは、DELETE文で削除するのではなくUPDATE文で論理削除する。
   - download_cursorでは、DELETEフラグのない行を選択（SELECT）する。
   - download_delete_cursorでは、DELETEフラグのある行を選択（SELECT）する。ここで選択された行は、リモートデータベースでは物理削除される。

2. シャドーテーブル（shadow table）

   1つのテーブルに対して、その主キーを記録するテーブルを用意する（「シャドーテーブル」と呼ぼう）。元のテーブルでは物理削除するが、トリガなどを利用して削除された行の主キーを必ずシャドーテーブルに記録する。download_

delete_cursorでは、シャドーテーブルにある行を選択（SELECT）する。

■ 仮削除の物理削除

仮削除の注意点は、統合データベース上では物理的な行が残ってしまう点だ。これらが自動で消されることはないので、バッチ処理的に消す必要がある。仮削除した行の更新時刻と同期ユーザの最終同期時刻を調べ、仮削除のイベントがすべてのリモートデータベースに行き渡ったことを確認すれば、その行を物理削除できる[23]。このような処理を夜間バッチなどで実行する。

7.4.8 主キーの生成

データベースが分散してしまうと、一意の主キーを生成するのが難しくなるが、そのテクニックをいくつか紹介する。

■ UUIDを使う

UUID（Universally Unique Identifier）とは、16バイトのランダムな値で、別々のマシン上で生成されたとしても重複しないことが保証されているIDだ[24]。GUID（Globally Unique Identifier）とも呼ばれる。SQL AnywhereにはUUIDを生成する関数NEWIDがあり、カラムのデフォルト値とすることもできるので、手軽にUUIDを利用できる。

■ Autoincrementを複合キーとして使う

Autoincrementによる値の生成だけでは、複数のデータベース間で値が重複してしまう。しかし、データベース固有の値（たとえばユーザ名など）のカラムとAuto-

23　統合データベース上のMobile Linkのシステムテーブル ML_SUBSCRIPTIONで、同期ユーザごとの最終アップロード時刻や最終ダウンロード時刻が参照できる。

24　文字列表現も可能で、UUIDTOSTR関数やSTRTOUUID関数がある。

incrementのカラムとを複合キーとすれば、一意の主キーとなる。

■ Global Autoincrementを使う

SQL Anywhereには、Autoincrementだけでも値が重複しないような仕組みがある。それがGlobal Autoincrementで、整数型の範囲をあらかじめ区切って、その範囲内で値をインクリメントしていく。たとえば、1万ごとに範囲を区切ったとすると、リモートデータベースAは1から10000を使い、リモートデータベースBは10001から20000を使うというように設定できる。

Global Autoincrementの宣言で分割する範囲を指定し、個々のデータベースがどの範囲を使うかは、GLOBAL_DATABASE_IDパラメータとして数字を割り振る。

7.4.9 スキーマ変更

同期環境が稼働したあとで、テーブルスキーマを変更したい場合もあるだろう。次のような手順で実現できる。たとえば、ある同期対象テーブルにカラムを1つ追加したいとする。

1. 統合データベースでカラムを追加
2. 同期スクリプトとスクリプトバージョンの追加
   スキーマ変更後の同期スクリプトを追加する。このとき、新しいスクリプトバージョンを用いて登録する。

この時点で、古いスクリプトバージョンを指定して同期していれば、依然としてリモートデータベースは同期可能だ。

3. リモートデータベースでカラムを追加

リモートデータベースでは、未同期のトランザクションが存在しなければスキーマを変更してよい。よって、古いスクリプトバージョンで同期後すぐにカラムを追加する。これに合わせて、パブリケーションも変更する必要がある。

これらを実行するイベントフックsp_hook_dbmlsync_schema_upgradeがあるので、利用すると便利だ。イベントフックを設定して、カラムを追加するSQL文を書いておく。古いスクリプトバージョンで同期を行えば、カラムが自動追加され、パブリケーションが更新される。この時点でスキーマの変更は完了する。

4. 以後は、新しいスクリプトバージョンで同期を継続する

7.4.10 まとめ

以上、代表的な設定例を紹介した。細かなテクニックも多かったが、すべてを一度に理解する必要はないので、必要なものから順に取り組んでいただきたい。

このほかにも、以下のようなものがあるので状況に応じて利用していただきたい。

- 回線切断などによりダウンロードストリームが失敗したときに途中から再開する「レジューム」機能
- ダウンロードストリームをファイルで行う「File-based download」
- 統合データベースの更新を検知してリモート側に伝える「サーバ駆動同期」

7.5 Mobile Linkのチューニングポイント

Mobile Linkを運用していく上でのチューニングポイントをいくつか紹介する。

7.5.1 統合データベース

Mobile LinkはODBCで統合データベースに接続するので、Mobile Link固有の懸念は特にない。Mobile Linkサーバが実行するSQLやストアドプロシージャに対して、通常のデータベースクライアントと同様のチューニングを統合データベースに施せばよい。

1つ挙げるとすれば、Mobile Linkユーザや最終更新日付のカラムはインデックスの候補だ。ダウンロードストリームを生成する際に、これらのカラムを検索条件とすることが多いからだ。この検索は頻繁に発生するものであり、テーブルスキャンになるとパフォーマンスが悪化しやすいので、適切なインデックスが使われるように配慮する。

7.5.2 Mobile Linkサーバ

スレッド数

適切なスレッド数でMobile Linkサーバを動作させることがチューニングの基本だ。スレッド数の設定は、dbmlsrv9の-wオプションで行う。デフォルトは5スレッドだが、たとえば8スレッドでMobile Linkサーバを起動するには次のようにする。

```
$ dbmlsrv9 -w 8 -c "dsn=asademo9"
```

この数が、同期クライアントの同時実行数を決定する。なるべく多くのスレッド

数を設定すべきだが、多すぎると逆に統合データベースの負荷が高くなり、パフォーマンスは悪化する。

最適なスレッド数は、統合データベースの性能やシステム環境や処理内容などに依存するので一概には言えない。スレッド数を変えながらパフォーマンスを実際に計測して、最適な数を決定するほかないが、経験上、5から10スレッド程度がよい。

dbmlsrv9のオプションには、スレッド数に関するオプションとして、-wuもある。-wuオプションでは、アップロード時の同時スレッド数を指定する。アップロードとダウンロードでは最適なスレッド数が異なることもあるので、それらを別々に指定しようという考えだ。

-wオプションで指定したスレッド数は、アップロードストリームとダウンロードストリームの両方に使われる。しかし、アップロードストリームのほうが一般に統合データベースに対する負荷が大きい。アップロード時はデータの更新が主であり、一方、ダウンロード時はデータを参照するのが主だからだ。このようなとき、アップロード時の同時接続数を小さめにし、ダウンロード時の同時接続数を大きめにするとパフォーマンスが向上することがある。そこで、-wオプションで指定したスレッド数の範囲内で、アップロードの同時実行数を-wuオプションで指定できる。たとえば、10スレッドのうちアップロードには最大3つ同時に利用したければ、以下のように指定する。

```
$ dbmlsrv9 -w 10 -wu 3
```

■ キャッシュ

Mobile Linkサーバがデータをやり取りする際に利用するメモリキャッシュサイズを適切に設定する。キャッシュにはアップロード用のキャッシュとダウンロード用のキャッシュとがある。

アップロード用のキャッシュサイズのデフォルトは500KBである。これを超える

データをMobile Linkサーバが同期クライアントから受け取ると、残りはテンポラリファイルとしてディスクに記録されてしまい、ディスクI/Oが発生する。このキャッシュサイズはMobile Linkサーバ全体に対する設定であるので、同期クライアントあたりのアップロードデータ量とMobile Linkサーバのスレッド数を考慮して、適切なサイズを指定する。たとえば、同期クライアントあたり100KBのデータをアップロードし、10スレッドでMobile Linkサーバを動かすのであれば、1MB程度のアップロードキャッシュがほしいので、以下のように指定する。

```
$ dbmlsrv9 -w 10 -u 1MB
```

ダウンロード時も同様だ。ダウンロード用のキャッシュサイズのデフォルトは500KBである。これを超えるデータをMobile Linkサーバが統合データベースから受け取ると、残りはディスクに記録されてしまう。たとえば、同期クライアントあたり1MBのデータをダウンロードし、10スレッドでMobile Linkサーバを動かすのであれば、10MB程度のダウンロードキャッシュがほしいので以下のように指定する。

```
$ dbmlsrv9 -w 10 -d 10MB
```

■ 同期スクリプトの確認の抑制

Mobile Linkサーバは、同期のたびに、統合データベース上の同期スクリプトに変更がないか確認する（デフォルト動作）。開発時には便利な機能だが、本番環境では同期スクリプトが頻繁に変更されることは少ないので、無駄な動作と言える。-fオプションを付けると、同期スクリプトの確認はサーバ起動時の1回だけとなる。

```
$ dbmlsrv9 -f
```

7.5.3 同期クライアント

■ ダウンロードバッファサイズ

同期クライアントがダウンロードデータを受信する際、メモリ上のバッファを利用する。このバッファサイズはデフォルトで1MBであるが、これを超えるデータを受信すると、テンポラリファイルとして記録されるのでディスクI/Oが発生する[25]。ダウンロードデータサイズを考慮して、適切な値に設定する。設定は、dbmlsyncの引数として起動時に設定する方法と、サブスクリプション内でデフォルト値として設定しておく方法の2通りがある。たとえば、2MBに設定するには、以下のように指定する。

- dbmlsyncの引数で指定

```
$ dbmlsync -e "DownloadBufferSize=2M"
```

- サブスクリプションで指定

```
CREATE SYNCHRONIZATION SUBSCRIPTION
TO sales_publication
FOR ml_user1
OPTION DownloadBufferSize='2M';
```

■ HTTP/S通信でのアップロードバッファサイズ

HTTP/S通信で同期をする際は、アップロードバッファサイズも指定できる。アップロードストリームは、このバッファサイズ単位で送信される。デフォルトは64KBだ。たとえば、640KBのデータをアップロードする際、10回のHTTP通信として送信されるので無駄が多い。通信環境に合わせて適切な値を設定する。1MBに設定するには、以下のように指定する。

25 デフォルトのバッファサイズは、Windows CEでは32KBである。

- dbmlsyncの引数で指定

```
$ dbmlsync -e "ctp=HTTP;adr='host=localhost;port=80;
buffer_size=1024000'"
```

- サブスクリプションで指定

```
CREATE SYNCHRONIZATION SUBSCRIPTION
TO sales_publication
FOR ml_user1
TYPE 'http'
ADDRESS host='localhost;port=80; buffer_size=1024000'
```

トランザクションログの自動削除

パフォーマンスの話題から多少外れるが、関連する話としてトランザクションログの扱いについて触れておく。リモートデータベースのトランザクションログは、アップロードストリームの生成に必要なので無闇に削除してはならない。同期済みの部分を自動的に削除する機能があるので、それを利用するとよい。そのためには2つの仕掛けが必要だ。

1. dbmlsyncの–xオプションの指定

同期終了後、アップロード済み部分をトランザクションログから切り離して別ファイルに保存する。なお、-xオプションでは切り離すべき上限ファイルサイズも指定できる。たとえば、「-x 1M」と指定すれば、トランザクションログのサイズが1MBを超えたときに切り離しが行われる。

2. DELETE_OLD_LOGSデータベースオプション

DELETE_OLD_LOGSデータベースオプションはデフォルトではオフだが、オンに設定するとdbmlsyncの-xオプションで切り離されたログが自動的に削除される。

```
SET OPTION PUBLIC.DELETE_OLD_LOGS = 'ON'
```

また、「ON」ではなく「DELAY」を指定してもよい。DELAYでは、ログが切り離された当日は自動削除せず、翌日以降に削除対象となり、削除期間に余裕を持たせられる。

7.5.4 リモートデータベース

リモートデータベースに関するMobile Link固有の設定は特にない。

第8章 Ultra Light

8.1 Ultra Lightとは

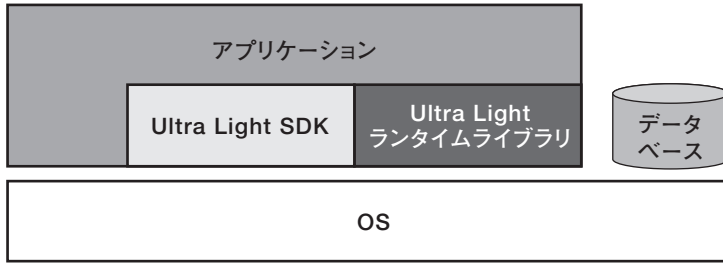
Ultra Lightは、Windows CEやPalm OSに対応するモバイルデバイス向けデータベースだ[1]。極めて省リソースで動作し、RDBMSとしての特徴も備える。トランザクション、参照整合性、インデックス、暗号化などの機能を持ち、JOIN文を含めたさまざまなSQL文を実行できる。さらに、Mobile Link同期クライアントとしての機能もある。このように、Ultra Lightを使えばWindows CEやPalm OSが動作するPDAや業務用端末・デジタル機器などのモバイルデバイス上でもデータベースを基盤としたデータ保存や検索機能を持つアプリケーションを開発でき、統合データベースとの同期も可能となる。

SQL Anywhere StudioにはSQL AnywhereとUltra Lightという2つのデータベースがあるが、そのアーキテクチャは大きく異なる（図8.1）。Ultra Lightを利用するアプリケーション開発はUltra Light SDK（Software Development Kit）を利用し、ライブラリのAPIを介してUltra Lightのデータベースにアクセスする。SQL Anywhereのようにアプリケーションとは別プロセスのデータベースサーバが起動して、アプリケーションがTCP/IP接続しながらデータベースを利用するというわけではない[2]。

1 Windows XP/XP Embedded/Tablet PCにも対応する。

2 Ultra Light EngineというUltra Lightデータベースのための「データベースサーバ」があるので本文の説明は適切ではないが、アプリケーションに組み込んでUltra Lightを利用するのが一般なので本書ではUltra Light Engineについては言及しない。

Ultra Lightのアーキテクチャ例



SQL Anywhereのアーキテクチャ例

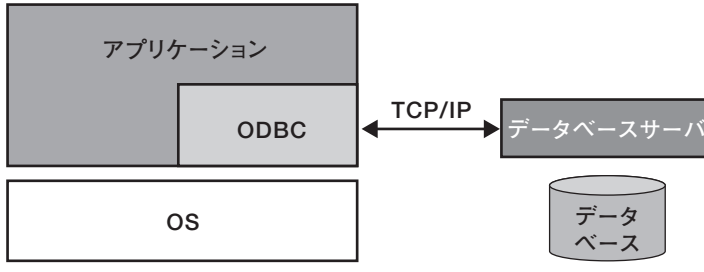


図8.1：Ultra LightとSQL Anywhereとのアーキテクチャの比較

■ SQL Anywhere for CEとの棲み分け

Windows CE上のデータベースとして、Ultra LightとSQL Anywhere for CEとのどちらを選択すべきだろうか。いくつかの視点から比較してみよう。

省リソース性

Ultra Lightは動作時の最小消費メモリサイズが150KBであることからもわかるように、リソースの限られたシステムに適するデータベースだ。一方、SQL Anywhereの最小消費メモリサイズは8MBだ。

データベース性能

Ultra Lightの機能はSQL Anywhereのサブセットとなるので、複雑な処理をす

るにはSQL Anywhereを使う必要がある。とはいえ、Ultra Lightがサポートする SQLはモバイルデバイスの用途として十分な機能を備える。

SQL Anywhereとの互換性

WindowsやLinuxなどのほかのプラットフォーム上のSQL Anywhereとの互換性を重視するなら、SQL Anywhere for CEを選択すべきだ。SQL Anywhereのデータベースファイルはバイナリレベルで互換性があるので、Windows CE上のデータベースファイルをコピーするだけで、ほかのプラットフォーム上でもそのデータベースを利用できる。

一方、Ultra Lightは、データベースエンジンとアプリケーションとが一体なので、アプリケーションがマルチプラットフォームを意識して作られていない限り、別のプラットフォーム上では動作しない。データベースファイルはSQL Anywhereとの互換性はなく、データを抽出するにはいったんXMLに変換する処理などが必要になる。

開発容易性

Ultra LightのAPIは決して難しくはないが、他社の開発環境との互換性はなく（ADO.NETとの互換性はある）、移植性は低い。一方、SQL Anywhereの場合はADO.NET、ODBC、JDBCといった汎用性の高いAPIが利用でき、SQL Anywhere同士や他社製データベースでの再利用がしやすい。

配布の容易さ

Ultra Lightが必要とするライブラリは数個であるため、アプリケーションの配布は容易だ。一方、SQL Anywhereは原則インストーラでインストールする。

表8.1：Ultra LightとSQL Anywhere for CEとの比較

	省リソース	性能	ASAとの互換性	開発容易性	配布
Ultra Light	◎	○	×	△	◎
SQL Anywhere for CE	○	◎	○	○	△

8.1.1 Ultra Lightの特徴と制限

Ultra LightはRDBMSの基本機能をサポートする。主な特徴と制限を以下に示す。

■ 主な特徴

- インデックス

  複数カラムのインデックスを作成できる。

- キー

  主キーや外部キーに対応して参照整合性を設定することができる。

- Mobile Link同期クライアント

  パブリケーションを設定して、Mobile Link同期ができる。

- トランザクションとリカバリ

  トランザクションに対応し、コミットやロールバックができる。また、自動リカバリも可能で、トランザクションの途中でアプリケーションやデバイスに障害が起きてもデータベースは自動的に修復される。

- セキュリティ

  データベース接続時に認証可能。データベースファイルの暗号化やMobile Link同期通信の暗号化にも対応する。

- 高パフォーマンス

  リソースの限られたデバイス上で省リソース・高パフォーマンスに動作するよ

う、データ構造やキャッシュアルゴリズムが最適化されている。

■ 主な制限

項目	制限
データベースへの接続数	14接続まで
テーブルのカラム数	65,535カラム。ただし、行サイズによる制限もある。行は4KBまでなので、実際には4,000カラム程度が上限となる
インデックスの数	65,535
テーブル内の行数	65,534
データベース内の行数	制限なし（ほかの制約から制限される）
データベース内のテーブル数	およそ1,000テーブル
1つのトランザクションから参照可能なテーブル数	制限なし
行サイズ	およそ4KB（圧縮後）。なお、LONG VARCHAR型とLONG BINARY型の値は別に保存されるので、これには含まれない
データベースファイルのサイズ	2GB（またはOSによるファイルサイズ制限）

8.1.2 Ultra Lightのアーキテクチャ

アプリケーションとUltra Lightデータベースとの関係は次の3通りある。

- 1つのアプリケーションが1つのUltra Lightデータベースに対して、同時に1つまたは複数のコネクションを利用できる（後者はマルチスレッドアプリケーションの場合）[3]。
- 1つのアプリケーションは、最大4つのUltra Lightデータベースを利用できる。

3 ただし、Ultra Light for MobileVBやUltra Light ActiveX、Ultra Light for M-Business Anywhere、Palm OS上のUltra Lightでは、言語・開発環境の制限によりマルチスレッドで利用できないものもある。

- Ultra Light Engineを利用すれば、1つのUltra Lightデータベースに対して複数のアプリケーションが接続できる。

しかし、1つのアプリケーションが1つのUltra Lightデータベースを利用するのがUltra Lightの主な利用方法なので本書ではこの場合のみを扱い、Ultra Light Engineには言及しない。

アプリケーションはUltra Light SDKを通してUltra Lightデータベースのデータにアクセスする。このときのアーキテクチャは図8.2のようになる。

図8.2 : Ultra Lightのアーキテクチャ

■ アプリケーションとUltra Light SDK

Ultra Light SDKが提供するAPIを利用して、データベース操作をするアプリケーションを構築する。SDKは言語ごとに用意されている。

■ ランタイムライブラリ

実行時に必要なライブラリ（DLL）のこと。言語やプラットフォームごとに用意されている。

■ Ultra Light スキーマファイル

テーブル定義などが書かれたバイナリファイル。Ultra Light スキーマペインタなどを利用して作成し、アプリケーションと一緒に配備する。

■ Ultra Light データベースファイル

データベースファイルは、通常、アプリケーションの初回起動時にスキーマファイルから生成される。データやインデックスなどデータベースに関する情報はすべてデータベースファイルに記録される。

■ テンポラリファイル

アプリケーション実行時にテンポラリファイルが利用されることもある。

Column　Ultra Light にはトランザクションファイルがない

Ultra Light には、SQL Anywhere でいうところのトランザクションログファイルはない。その代わりに、データベースの各行にトランザクションに関する情報などを記録する 1 バイト分の領域があり、そこでステータスが管理される。よって、トランザクションログファイルがなくても、Ultra Light はシステム障害からのリカバリ機能や同期クライアントとしての機能を持つ。

8.1.3 Ultra Light の種類

Ultra Light SDK には、大きく以下の2種類がある。本書では Ultra Light コンポーネントを主に扱う。

- Ultra Light コンポーネント
  SQL Anywhere Studio 8.0.2 から導入され 9.0.1 で強化されたもの。SQL 文を

通常のプログラムソース内に記述でき、開発が容易になった。利用可能なSQL構文も多彩で、これらは動的SQLと呼ばれる。動的SQLは、SQL AnywhereのSQLのサブセットとなっている。

- 静的インターフェイス
  SQL Anywhere Studio 9以前のバージョンからあったもの。Ultra Lightで使用するSQL文をあらかじめSQL Anywhereに登録しておき、プリコンパイルして利用するなど、開発の手間が多い。

Ultra Lightはアプリケーションに組み込まれるものなので、アプリケーションの開発環境（開発言語やプラットフォーム）ごとにSDKが提供されている。言語が異なっても開発手順は同じで、APIも似ている。Ultra Lightコンポーネントと静的インターフェイスとに分けてUltra Light SDKを列挙する。

表8.2 : Ultra Light コンポーネント

コンポーネント	説明
Ultra Light for MobileVB	Microsoft Visual Basicに統合可能なモバイルアプリケーション開発環境であるMobileVB向け
Ultra Light ActiveX	eMbedded Visual Basic 3.0での開発やPocket Internet Explorer（Pocket IE）のJScript向け
Native Ultra Light for Java	Java用のモジュールであるが、ネイティブコードを使ってパフォーマンスを向上させたもの
Ultra Light .NET	.NET Compact Framework用のモジュールで、Visual Basic .NETやC#向け
Ultra Light C++	C/C++向けのコンポーネント
Ultra Light for M-Business Anywhere	iAnywhere Solutions社のM-Business Anywhere（日本では未発売）と組み合わせて使う

表8.3：静的インターフェイス

インターフェイス	説明
Embedded SQL and Static C++ API	Embedded SQLを使ったC/C++向け
Ultra Light Static Java	JDBCを使ったPure Javaモジュール

各SDKが対応する動作プラットフォームは、表8.4のとおりである。なお、開発プラットフォームとしては、Windows NT/2000/XPに対応する（一部についてはWindows 98 SEも可能）。

表8.4：Ultra Lightの動作プラットフォームの概要。バージョンなどの詳細については、製品マニュアルおよびWebサイト（http://www.ianywhere.jp/sas/index.html）を参照

Ultra Light SDK	Windows CE	Palm	Windows XP[4]
Ultra Light static C/C++ API	○	○	○
Ultra Light Embedded SQL	○	○	○
Ultra Light static Java API	○		○
Ultra Light for AppForge MobileVB	○	○	○
Ultra Light ActiveX	○		○
Ultra Light .NET	○		○
Native Ultra Light for Java	○		○
Ultra Light C++ component	○	○	○
Ultra Light for M-Business Anywhere	○	○	
TCP/IP synchronization	○	○	○
HTTP synchronization	○	○	○
HTTPS synchronization	○	○	○
HotSync synchronization		○	
ActiveSync synchronization（3.5 and 3.6）	○		

4　Windows環境ではTablet PCでの使用を想定しているので、Windows XPのみのサポートとなる。

Ultra Light SDK	Windows CE	Palm	Windows XP[4]
Transport-layer security over HTTP or TCP/IP	○	○	○

このようにさまざまな種類のUltra Lightがあるので、利用したい言語や動作プラットフォームに合ったUltra Lightを選択する。

本書ではSQL Anywhere Studio 9.0.1で大幅に強化されたUltra Lightコンポーネントを取り上げ、静的インターフェイスには触れない。 以後、断りのない限り、Ultra Lightコンポーネントのことを単にUltra Lightと記述する。

Column　静的インターフェイスとは

Ultra Lightコンポーネントでは、SQL文はアプリケーション内に記述できる。一方、静的インターフェイスではリファレンスデータベースとなるSQL Anywhereを用意して、そこにテーブル定義やSQL文をあらかじめ定義する。そこからスキーマファイルとプログラムソースとをUltra Lightのツールを利用して生成し、生成されたコードをアプリケーションに組み込んでデータ操作を行う。このように、静的インターフェイスではアプリケーションのコーディングとSQL文のコーディングとが分断されてしまう。

しかし、静的インターフェイスの長所もある。あらかじめSQL文が定義されるため、スキーマやコードを生成した時点でSQL文の構文解析は済んでおり、一般にアプリケーションは高速だ。さらに、生成されたコードは定義されたデータ操作に必要十分な機能だけを含むので、アプリケーションのサイズも小さくできる。

8.1.5 Ultra Lightスキーマファイル

Ultra Lightスキーマファイルは Ultra Lightデータベースを生成するための設計図であり、独自のバイナリファイルになっている。Ultra Lightスキーマファイルを

作成するには、以下の4つの方法がある。

- Ultra Light スキーマペインタ
- リファレンスデータベース（SQL Anywhere）
- XML ファイル
- Ultra Light データベースファイル

Ultra Light スキーマペインタ

Ultra Light スキーマペインタとは、スキーマファイル作成のためのGUIツールだ。これを使って、スキーマファイル作成に関するすべての作業を行うことができる。

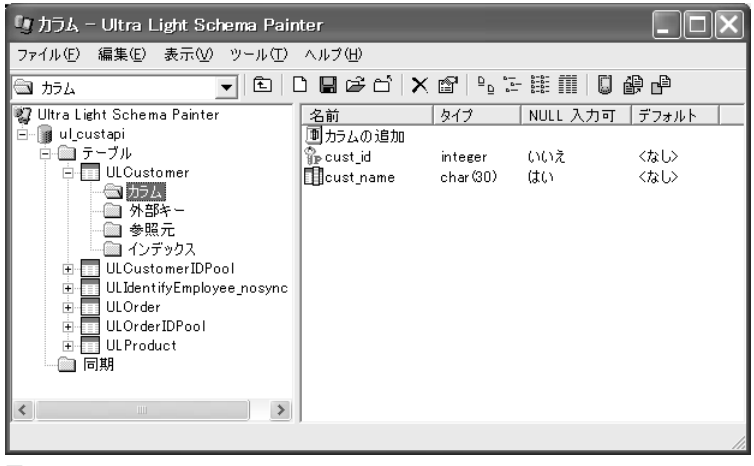


図8.3 : Ultra Light スキーマペインタ

SQL Anywhere から作成

SQL Anywhere データベースのスキーマをもとにしてUltra Light スキーマファイルを作成できる。このときのSQL Anywhere データベースをリファレンスデータ

ベースと呼ぶ。リファレンスデータベースは実際に運用するデータベースであってもかまわないが（たとえば統合データベース）、あくまでもリファレンスデータベースの役割はUltra Lightの実行とは関係がなく、スキーマファイルを作成するためだけのものである。

リファレンスデータベースに対してUltra Lightで使用するテーブルなどを設定し、ulinitコマンドでスキーマファイルを作成する。

■ XMLファイルから生成する

XMLファイルに設定を書くこともできる。ulxmlコマンドを使って、対象のXMLファイルからUltra Lightスキーマファイルを生成する（逆も可能）。XMLファイルは、%ASANY9%¥win32¥usm.xsdをもとにして一から書くこともできるが、Ultra Lightスキーマペインタで行った設定をXML形式で保存できるので、最初はUltra Lightスキーマペインタで作成してからXML形式に変換し、そのXMLファイルを変更していくことも可能だ。

■ Ultra Lightデータベースファイルから生成する

ulconvコマンドで、Ultra Lightデータベースからスキーマファイルを再生成することもできる。

これらの4つをまとめ、スキーマファイルを作成する方法とコマンドとの相関を図8.4に示す。

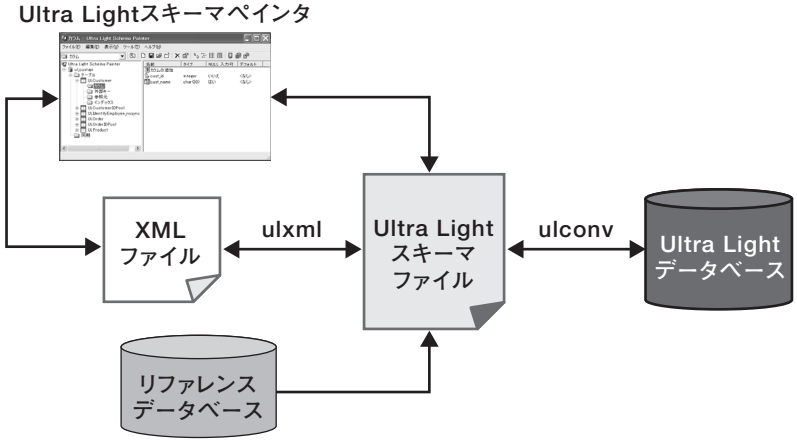


図8.4 : Ultra Light スキーマファイル生成の相関。一方から他方へ変換できることを矢印で示している

8.2 Ultra Light アプリケーションの開発例

では、実際にUltra Lightアプリケーションを開発してみよう。第7章「Mobile Link」の例で用いたデータベーススキーマを使ってUltra Lightアプリケーションを構築し、Mobile Linkサーバとの同期も試みる。

ここではNative Ultra Light for Javaを用いて、Javaによるコード例を示す。Ultra Lightデータベースの扱いは言語によらず、APIも同様に設計されているので、Java以外の言語のUltra Lightを利用する場合でも参考になるはずだ。

8.2.1 スキーマファイルの作成

Ultra Lightスキーマペインタを使って以下のような設定を行い、remote.usmとしてスキーマを保存する。

1. Productテーブルを作成する
2. product_id（char型、データ長16バイト）、quantity（integer型）の2つのカラムを作成する
3. productテーブルをパブリケーションに設定する

設定項目の参照として、Ultra LightスキーマをXMLで保存したものを以下に示す。

```
<?xml version="1.0" encoding="Shift-JIS" standalone="no"?>
<ul:ulschema xmlns:ul="urn:UltraLight">
  <collation name="932JPN" case_sensitive="no"/>
  <options>
    <option name="dateformat" value="YYYY-MM-DD"/>
    <option name="dateorder" value="YMD"/>
    <option name="nearestcentury" value="50"/>
    <option name="precision" value="30"/>
    <option name="scale" value="6"/>
    <option name="timeformat" value="HH:NN:SS.SSS"/>
    <option name="timestampformat" value="YYYY-MM-DD HH:NN:SS.SSS"/>
    <option name="timestampincrement" value="1"/>
  </options>
  <publications>
    <publication name="product" tables="product"/>
  </publications>
  <tables>
    <table name="product" sync="changes">
      <columns>
        <column name="product_id" type="char(16)" null="no"/>
        <column name="quantity" type="integer" null="yes"/>
      </columns>
      <primarykey>
        <primarycolumn name="product_id" direction="asc"/>
      </primarykey>
        <indexes/>
    </table>
  </tables>
</ul:ulschema>
```

8.2.2 アプリケーション開発

ここからJavaのコーディングとなる。Native Ultra Light for Javaのライブラリは、%ASANY9%¥ultralite¥NativeUltra LiteForJava¥jul9_ja.jarにある。以下、Ultra Light開発に関連する部分のみをコード例として示す。

■ DatabaseManagerインスタンス

スキーマファイルからのデータベース作成やデータベース接続はすべてianywhere.native_ultralite.DatabaseManagerオブジェクトから行う。このオブジェクトはアプリケーションに1つあればよい。

```
ianywhere.native_ultralite.DatabaseManager dm =
new ianywhere.native_ultralite.DatabaseManager();
```

データベースの作成はDatabaseManager#createDatabase()を利用し、データベースの接続はDatabaseManager#openConnection()を利用する。これらを実行するには次の接続パラメータが必要だ。

■ 接続パラメータCreateParmsの作成

接続パラメータはianywhere.native_ultralite.CreateParmsオブジェクトとして表現する。

```
ianywhere.native_ultralite.CreateParms params =
new ianywhere.native_ultralite.CreateParms();
params.schema.schemaOnDesktop = "D:¥¥asa9book¥¥UltraLight¥¥remote.usm";
params.databaseOnDesktop = "D:¥¥asa9book¥¥UltraLight¥¥remote.udb";
```

CreateParmsオブジェクトに、Ultra Lightスキーマファイル名（remote.usm）を設定する。この情報を基にして指定されたデータベース名（remote.udb）でUltra

Lightデータベースが作成される。

■ データベースの作成と接続

はじめてアプリケーションを起動するときはデータベースを作成してから接続し、次回起動以降は既存のデータベースに接続する。よって、データベースの有無（初回起動か否か）を判定して、データベースを作成する必要がある。判定の面倒を回避するため、多少趣向をこらしてみる。

```
ianywhere.native_ultralite.Connection conn = null;
try {
 conn = dm.openConnection(params);
} catch (SQLException e) {
if (e.getErrorCode() == SQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUND) {
 conn = dm.createDatabase(params);
} else {
 throw e;
 }
}
```

DatabaseManager#openConnection()は接続パラメータに指定されたデータベースに接続し、ianywhere.native_ultralite.Connectionオブジェクトを返す。このとき、データベースファイルがなければSQLCode.SQLE_ULTRALITE_DATABASE_NOT_FOUNDエラーが発生する。

そこで、まずデータベースに接続してみて、エラーが発生したらデータベースを作成する。データベースはDatabaseManager#createDatabase()で作成し、接続パラメータに指定されたスキーマに従って生成される。このメソッドはデータベース作成後、自動的に接続してConnectionオブジェクトを返す。

■ SQL文の処理：INSERT / UPDATE / DELETE

SQL文の実行は、“prepared statementを作成して、実行”の流れになる。結果

セットを返さないSQL文（INSERT/UPDATE/DELETE）の実行は次のようになる。

```
ianywhere.native_ultralite.PreparedStatement stmt =
conn.prepareStatement("INSERT INTO product(product_id, quantity)
values(?,?)");
stmt.setStringParameter(1, "eggplant");
stmt.setIntParameter(2, 50);
stmt.executeStatement();
stmt.close();
conn.close();
```

SQL文の処理：SELECT

結果セットを返すSQL文では、ianywhere.native_ultralite.ResultSetオブジェクトからデータを取得する。

```
ianywhere.native_ultralite.PreparedStatement stmt =
conn.prepareStatement("SELECT product_id, quantity FROM product");
ianywhere.native_ultralite.ResultSet rs = stmt.executeQuery();
rs.moveBeforeFirst();
while (rs.moveNext()) {
 System.out.println(rs.getString(1) + "¥t" + rs.getInt(2));
}
rs.close();
stmt.close();
conn.close();
```

以上がSQL文を実行する一連の流れだ。多少の違いはあるが、JDBCを使った流れとほぼ一致するので、JDBCの扱いに慣れていれば迷うことはないだろう。

8.2.3 Mobile Link同期

Mobile Link同期を行うにはConnectionオブジェクトに対して同期クライアントの設定を行い、Connection#synchronize()を実行する。

```
conn.syncParms.setStream(StreamType.TCPIP);
conn.syncParms.setStreamParms("host=localhost;port=2439");
conn.syncParms.setVersion("default");
conn.syncParms.setUserName("ul_user");
conn.synchronize();
```

Column Ultra Lightでのパブリケーション

Ultra Lightではデフォルトですべてのテーブルが同期対象となる。このため、Ultra Light内のテーブルをすべて同期したいならパブリケーションを作成する必要はない。逆に、一部のテーブルを同期対象としたいのならそのためのパブリケーションを作成し、パブリケーションを指定して同期する[5]。

なお、Ultra Lightではテーブルのすべてのカラムが同期対象となり、カラムを絞って同期することはできない。ここはSQL Anywhereと違う点で、SQL Anywhereではパブリケーションにてテーブルやそのカラムを同期対象として指定できる。

8.2.4 配備（deployment）

作成したアプリケーションを配備するには、アプリケーションプログラムとスキーマファイルのほかに、次のライブラリが必要だ。

5 テーブル名が「nosync」で終わるものは自動的に同期対象から外されるという機能がUltra Lightにはある。パブリケーションではなく、この機能を利用してもよい。

- jul9_ja.jar
  開発時に利用したもの。%ASANY9%¥UltraLite¥NativeUltraLiteForJava¥jul9_ja.jarにある。
- jul9.dll
  実行時ライブラリ。%ASANY9%¥UltraLite¥NativeUltraLiteForJavaにWidows XP用のものとWindows CE用のものとが用意されているので、適切なものを選ぶ。java実行時には、-Djul.library.dirでこのDLLが置かれたディレクトリを指定する。

8.2.5 実行

アプリケーションをSample.jarにアーカイブして、メインクラスはApplication.classで、jul9_ja.jarとjul9.dllとはC:¥UltraLight¥lib¥に置いたとすると、実行時のjavaコマンドは次のようになる。

```
$ java -Djul.library.dir=C:¥UltraLight¥lib -classpath C:¥
UltraLight¥lib¥jul9_ja.jar Application
```

8.3 その他

8.3.1 Ultra Lightデータベースからデータの抽出

Ultra Lightデータベースのデータを XMLに変換したり、逆に、データの書かれたXMLからUltra Lightデータベースを作成したりできる。Ultra Lightデータベースのデータを Ultra Lightアプリケーション以外から利用したい場合などに効果的な方法だ。

ulconvコマンドで、Ultra LightデータベースとXMLファイルとを相互変換する。Ultra LightデータベースをXMLファイルに変換するには、次のように実行する。

```
$ ulconv unload -c DBF=original.udb unloaded.xml
```

逆に、XMLファイルをUltra Lightデータベースに変換するには、次のように実行する。

```
$ ulconv load -u -c DBF=newdb.udb original.xml
```

8.3.2 接続時のセキュリティ

Ultra Lightの接続時に、デフォルトでは認証は不要だが、ユーザIDとパスワードを設定して認証を必須とすることもできる[6]。

Ultra Lightデータベースを作成する際、ユーザID「DBA」・パスワード「SQL」のユーザが自動的に作られる。接続時にユーザIDとパスワードが省略されるとこのユーザIDとパスワードが使われる。よって、接続時に認証情報を渡さなくても自動的にユーザID「DBA」で接続される。

そこで、明示的に認証するには新たなユーザを作成して、「DBA」ユーザを削除すればよい。ユーザの作成と削除は、Native Ultra Light for Javaを例にとると以下のようになる。

```
ユーザの作成：
Connection#grantConnectionTo(user_id, password)

ユーザの削除：
Connection#revokeConnectionFrom(user_id)
```

6 このユーザID・パスワードは接続時の認証のためだけのものであり、テーブルなどへのアクセス権としての機能はない。

8.3.3 暗号化

Ultra Lightも、SQL Anywhereと同様にデータベースファイルの暗号化やMobile Link同期通信の暗号化に対応する。

■ データベースファイルの暗号化

データベースファイルを作成するときに鍵を指定するとデータベースファイルは暗号化され、以後、鍵を指定しないと接続できないようになる。鍵は、Native Ultra Light for Javaであれば、ConnecionParms # encrptionKeyで指定する。

■ Mobile Link同期通信の暗号化

HTTP/Sによる暗号化通信ができる。同期クライアントのストリームタイプとしてStreamType.HTTPSを指定し、trusted_certificatesなど必要なパラメータを設定すればよい。

第9章 XMLとHTTP

9.1 XMLの扱いとHTTPサーバ機能

SQL Anywhere 9ではXMLの扱い方が強化され、SQL AnywhereにHTTPサーバ機能が追加された。これら2つの機能を組み合わせて、SOAPメッセージをやり取りするWebサービスをSQL Anywhereだけで構築できる。そのため、XMLやHTTPアクセスを利用する既存システムにSQL Anywhereは馴染みやすい。

リレーショナルデータベースが採用しているリレーションモデルは依然としてデータモデルの主流であり、リレーショナルデータベース上に保存されているデータの量も膨大である。だが、昨今XMLの重要性が増していることも事実である。システム間でデータを交換するためのデータ形式としてXMLは広く利用されており、階層構造を表すデータモデルとしてもXMLは優れている。そのため、リレーショナルデータベース内のデータとXMLとを連携させる手段、とりわけリレーショナルデータベースへの問い合わせ結果をXMLで出力したいという要望が多い。SQL Anywhereは、標準規格であるSQL/XMLを採用し、リレーショナルデータベース内のデータをXMLとして活用しやすくなっている。ただし、SQL Anywhereはあくまでも RDBMSであって、決してオブジェクトデータベースや階層データベースを指向するようになったわけではない。リレーショナルデータベースを基盤として、XMLの扱いを容易にしようとしているのである。

さらに、SQL AnywhereにはHTTPサーバの機能が組み込まれた。データベース上の処理とURLとを対応させておき、そのURLにHTTP経由で問い合わせると、結果セットがHTMLやXMLで取得できる。HTTPやXML変換の処理はSQL Anywhereが自動的に行うので、データベース管理者はSQL文やストアドプロシージャの処理をWebサービスとして登録するだけでよい。ユーザがWebブラウザでデータを閲覧できるシステムをSQL Anywhereだけで構築してもよいし、システム間のデータ転送手段としてこのHTTPサーバ機能を利用してもよい。SOAP（Simple

Object Access Protocol）にも対応しているので、SQL Anywhere上にSOAPサービスを登録してWebサービスとしてデータを提供することも可能だ。

データベースサーバとHTTPとの組み合わせから、SQL Anywhereがアプリケーションサーバを目指そうとしているように連想されてしまうかもしれないがそのような意図はなく、データベースへのアクセス手段としてHTTPも利用可能になったということである。その意味を込めて、これらの機能を本章では、「Webサーバ」ではなく「HTTPサーバ」と記述する。Webサーバと書いてしまうと、HTMLファイルなどを提供するアプリケーションサーバのように思われてしまうからだ。

このように、ODBCやJDBCといった従来のアクセス方法だけでなくXMLやHTTPも利用できるので、SQL Anywhere上のデータはほかのシステムと連携しやすくなっている。次節以降、XML機能とHTTPサーバ機能とを紹介していくが、これらの分野ではデータベース以外のさまざまな技術が登場する。個別の技術を説明することは本書の範囲を超えるので、それらの解説は専門書に任せることにする。

9.2 XML機能

リレーショナルデータベースとXMLとではデータの表現形式が大きく異なる。リレーショナルデータベースでは2次元の表でデータを表し、カラムの順序や行の順序は問われない。一方、XMLはタグを使ってデータをツリー構造で階層的に表し、要素間に上下や前後の関係がある。このような差異があるため、リレーショナルデータベースとXMLとでデータをやり取りするにはデータ形式を変換する手段が必要となる。そのひとつがSQL/XMLだ[1]。

1 リレーショナルデータベースとの関連からは離れるが、XMLを処理する技術として、XSLTやXPath、XQueryなどさまざまなものがある。

SQL/XMLは、XMLデータを扱うために拡張されたSQLの規格のことで、2003年にANSI/ISOによってSQL Standardとして認められた。XMLを扱うための機能はデータベースベンダーが独自に拡張している部分が多いが、標準規格となったSQL/XMLに関してはベンダーの足並みが揃っているようだ。SQL Anywhereは、SQL/XMLで定められている関数（XMLAGG、XMLCONCAT、XMLELEMENT、XMLFOREST、XMLGEN）やXML型などに対応している。SQL/XMLはSQLの一部としてXMLを扱えるようになっているので、すでにSQLに慣れているユーザならばいくつかの関数の使い方を覚えるだけで習得できる。

SQL Anywhere上でXMLを扱う機能は、大きく3つに分類される。

1. XML文書の保存
2. リレーショナルデータ（結果セット）からXML文書への変換
3. XML文書からリレーショナルデータへの変換

以下では、それぞれについて紹介していこう。

9.2.1 XML文書の保存

整数をINTEGER型、日付をDATE型で表現するのと同じように、XML文書を表現するデータ型としてXML型がSQL/XMLで規定され、SQL Anywhereも採用している。XML文書としての文字列をXML型へキャストしたり、XML文書を直接XML型のカラムに保存したりすることができる[2]。

なお、XML文書全体を1つのデータとして見るのではなく、XMLの個々の要素（データ）をテーブル構造の中に保存するには「9.2.3 XML文書から結果セットへ」で説明している方法を用いる。

2　ただし、文字列をXML型にキャストしたり保存する際には、well-formedのチェックは行われない。

■ XML型とLONG VARCHAR型との違い

SQL Anywhere内部では、XML型は基本的にLONG VARCHAR型と同じで、文字列であることに変わりはない。しかし、XMLにおけるエスケープ文字の扱いが異なる。LONG VARCHAR型とXML型のカラムを持つテーブルを作成して、違いを見てみよう。次のテーブルを用意する。

```
CREATE TABLE "testxml" (
    "id"   integer      NOT NULL,
    "col1" long varchar NULL,
    "col2" xml NULL,
    PRIMARY KEY ( "id" )
);
```

col1とcol2のそれぞれに、「<hat>cap</hat>」という文字列を挿入する。

```
INSERT INTO testxml VALUES (1, '<hat>cap</hat>', '<hat>cap</hat>')
```

さて、次のXML文書を出力したいとする。

```
<product><hat>cap</hat></product>
```

それには、SQL/XMLで定義されているXMLFOREST関数を利用すればよい。LONG VARCHAR型のcol1に対して実行してみると、以下のようになる。

```
SELECT XMLFOREST( col1 AS product )
  FROM testxml
実行結果：
<product>&lt;hat&gt;cap&lt;/hat&gt;</product>
```

col1のデータは単純な文字列と認識され、山括弧（<>）がエスケープされる。一方、XML型のcol2に対する実行結果は、以下のようになる。

```
SELECT XMLFOREST( col2 AS product )
  FROM testxml
実行結果：
<product><hat>cap</hat></product>
```

col2のデータがXMLとして正しく認識されているのがわかる。

9.2.2 結果セットからXML文書へ

SQLによる問い合わせの結果セットをXML文書として出力するには、3つの方法がある。

① OUTPUT文で、ファイルとして出力
② SELECT文でFOR XML節を利用して出力
③ SQL/XMLを利用して出力

テーブルのデータをファイルとして保存したいならば①を、XML文書への変換を簡単に行いたいならば②を、出力されるXML文書を細かく制御したいならば③を選ぶとよいだろう。

■ ① OUTPUT文で、ファイルとして出力

OUTPUT文は、実行された結果セットをファイルに出力する機能だ。出力データ形式としてXMLを指定すれば、テーブルのデータがXMLに変換されてファイルに書き出される[3]。次のように、SELECT文の最後にOUTPUT節を加え、出力ファイル名と出力形式とを指定すればよい。

3　その他のデータ形式としては、ASCII、DBASEII、DBASEIII、EXCEL、FIXED、FOXPRO、HTML、LOTUS、SQLがある。

```
SELECT *
  FROM product
 WHERE id = 300;
OUTPUT TO 'D:¥¥sample.txt' FORMAT XML
結果
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resultset [
    <!ELEMENT resultset (resultsetdata) >
    <!ELEMENT resultsetdata (row)* >
    <!ELEMENT row (column)+ >
    <!ELEMENT column (#PCDATA)>
    <!ATTLIST column
        null (true | false) "false"
        name CDATA #IMPLIED
    >
]>
<resultset>
    <row>
        <column name="id">300</column>
        <column name="name">Tee Shirt</column>
        <column name="description">Tank Top</column>
        <column name="size">Small</column>
        <column name="color">White</column>
        <column name="quantity">28</column>
        <column name="unit_price">9.00</column>
    </row>
</resultset>
```

■ ② SELECT文でFOR XML節を利用して出力

SELECT文でFOR XML節を利用すれば、結果セットをXML形式で出力できる。その際、データのネストの仕方として、以下の3つのモードがある。

1. RAW

   行を<row>タグで囲む。

2. AUTO

   行をテーブル名のタグで囲む。さらに、ELEMENTSオプションでカラムの表

現方法として属性か要素かを指定できる。

3. EXPLICIT

行の表現方法をSELECT文中に明示する。

AUTOモードの出力例を示す。

```
SELECT *
  FROM product
 WHERE id = 300
   FOR XML AUTO, ELEMENTS
結果：
<product>
<id>300</id>
<name>Tee Shirt</name>
<description>Tank Top</description>
<size>Small</size>
<color>White</color>
<quantity>28</quantity>
<unit_price>9.00</unit_price>
</product>
```

■ ③ SQL/XMLを利用して出力

SQL/XMLで規定された関数を用いて、結果セットをXML文書に変換する。SQL Anywhereは、5つの関数をサポートしている。

- XMLAGG
- XMLCONCAT
- XMLELEMENT
- XMLFOREST
- XMLGEN

XMLELEMENTとXMLFORESTとを使った実行例を以下に示す。個々の関数の説明はマニュアルを参照していただきたい。

```
SELECT id, XMLELEMENT( NAME item_description,
                       XMLFOREST( name,
                                  color,
                                  unit_price AS price )
                     ) AS product_info
FROM product
WHERE id > 600
結果：
601,'<item_description>
<name>Sweatshirt</name>
<color>Blue</color>
<price>24.00</price>
</item_description>'
700,'<item_description>
<name>Shorts</name>
<color>Black</color>
<price>15.00</price>
</item_description>'
```

これと同じ結果となるものをXMLGENで書くと次のようになる。

```
SELECT id, XMLGEN( '<item_description>
<name>{$name}</name>
<color>{$color}</color>
<price>{$price}</price>
</item_description>',
                       name,
                       color,
                       unit_price as price)
FROM product
WHERE id > 600
```

9.2.3 XML文書から結果セットへ

SQL AnywhereにはOPENXMLというプロシージャがあり、これを用いるとXML文書の内容をテーブルのように扱えるので、XMLから結果セットを生成できる。その際、XML内の要素や属性を指示するのにXPathを用いる。実行例を以下に示す。

```
SELECT * FROM OPENXML( '<inventory>
                         <product>Tee Shirt
                          <quantity>54</quantity>
                          <color>Orange</color>
                         </product>
                         <product>Baseball Cap
<quantity>112</quantity>
                          <color>Black</color>
                         </product>
                        </inventory>',
                       '/inventory/product' )
WITH ( quantity CHAR(3) 'quantity',
       color CHAR(20) 'color')
結果：
'54','Orange'
'112','Black'
```

9.3 HTTPサーバ機能

SQL AnywhereのHTTPサーバ機能を利用すれば、データベースにURLでアクセスしてSQL文やストアドプロシージャの実行をさせ、その結果をHTMLやXMLなどで取得できる。URLと処理との組み合わせは「Webサービス」としてSQL Anywhereに登録する。ただし、データベース処理へのアクセス手段としてのHTTPサーバなので、ファイルシステム上のHTMLファイルや画像ファイルを配信したり、

CGIを実行したりするような「Webサーバ」の機能はない。

さらに、SOAPによるWebサービスも構築可能だ。Webサービス登録時にSOAPオプションを選択すれば、そのサービスは自動的にWebサービスとして利用できる。加えて、SQL Anywhere上に登録されているWebサービスに対するWSDL（Web Service Definition Language）の生成もできる。

HTTPメッセージやSOAPメッセージのやり取りはSQL Anywhereが自動的に解釈するので、ユーザが行うのはSQL文やストアドプロシージャの処理を書くことと、それをWebサービスとして登録することだけだ。

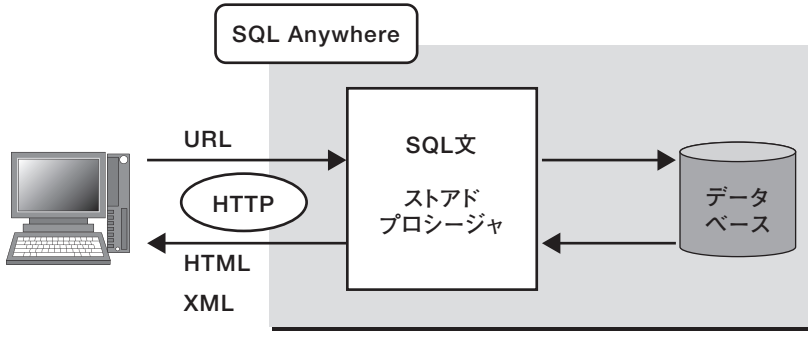


図9.1 : HTTP サーバのアーキテクチャ

コンテンツの生成方法は、以下の2通りある。

1. 結果セットの自動フォーマット機能を利用する方法

   SQL文やストアドプロシージャの結果を、HTMLやXMLへ自動的に整形して出力する。SQL文やストアドプロシージャを作成し、整形方式とともにWebサービスとして登録するだけでよい。SQL文やストアドプロシージャの実行結果を、ほかのシステムがHTTP経由で取得したい場合に効果的だ。

2. 自分でフォーマットプログラムを書く方法
   SQL文を実行した結果のデータだけでなくHTMLデザインなども必要な場合は、スクリプトを自分で書く必要がある。HTMLドキュメントを生成する文字列処理をストアドプロシージャで書けば、任意のドキュメントを出力可能だ。

以下では、HTTPサーバの起動からWebサービスの設定までを見ていく。

9.3.1 起動

HTTPサーバ機能は、データベースサーバのオプション設定となっている。言い換えれば、データベースサーバを起動せずにHTTPサーバだけを起動することはできない。データベースサーバ起動オプションとして-xsを指定すると、HTTPサーバも機能する。デフォルトではオフなので、HTTPサーバ機能を利用する場合は必ず明示する。

```
$ dbsrv9 -xs http asademo.db
```

HTTPサーバの設定は-xsに続けて書く。たとえば、待ち受けポート番号を8080にし、アクセスログをC:¥httpd.logに出力するようにしたければ、以下のように設定する。

```
$ dbeng9 -xs http(port=8080;LogFile=C:¥httpd.log) asademo.db
```

9.3.2 自動フォーマット機能による出力

さて、http://localhost/customersというURLにアクセスして、Customerテーブルの全データをHTMLで出力するWebサービスを作成するとしよう。実行したい

SQL 文は次のとおりだ。

```
SELECT * from customer
```

このSQL 文をWebサービスとして登録するには、CREATE SERVICE 文を使って次のように記述する[4]。

```
CREATE SERVICE customers TYPE 'HTML'
AUTHORIZATION OFF USER DBA
AS SELECT * FROM customer
```

1行目でcustomersというURLに反応するようにし、出力形式（サービスタイプ）は「HTML」としている。

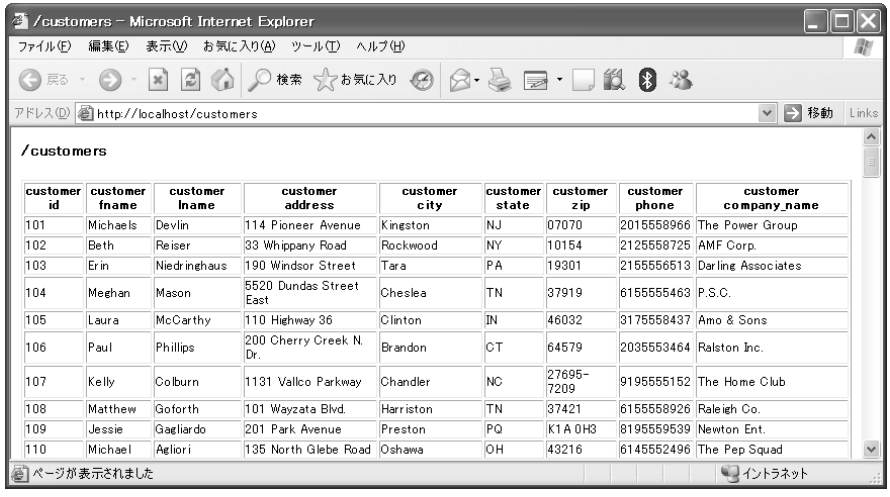
2行目はアクセス権限の指定で、ここでは誰でもアクセス可能とした。

3行目のAS句以降に実行したいSQL 文が記述されている[5]。

WebブラウザからHttp://localhost/customersにアクセスすると、customer テーブルの全データがHTMLのTABLEタグで整形されて表示される（図9.2）。

4　Sybase Centralからも作成可能。

5　もしストアドプロシージャを実行したいならば、ここで呼び出せばよい。



customer id	customer fname	customer lname	customer address	customer city	customer state	customer zip	customer phone	customer company_name
101	Michaels	Devlin	114 Pioneer Avenue	Kingston	NJ	07070	2015558966	The Power Group
102	Beth	Reiser	33 Whippany Road	Rockwood	NY	10154	2125558725	AMF Corp.
103	Erin	Niedringhaus	190 Windsor Street	Tara	PA	19301	2155556513	Darling Associates
104	Meghan	Mason	5520 Dundas Street East	Cheslea	TN	37919	6155555463	P.S.C.
105	Laura	McCarthy	110 Highway 36	Clinton	IN	46032	3175558437	Amo & Sons
106	Paul	Phillips	200 Cherry Creek N. Dr.	Brandon	CT	64579	2035553464	Ralston Inc.
107	Kelly	Colburn	1131 Vallco Parkway	Chandler	NC	27695-7209	9195555152	The Home Club
108	Matthew	Goforth	101 Wayzata Blvd.	Harriston	TN	37421	6155558926	Raleigh Co.
109	Jessie	Gagliardo	201 Park Avenue	Preston	PQ	K1A 0H3	8195559539	Newton Ent.
110	Michael	Agliori	135 North Glebe Road	Oshawa	OH	43216	6145552496	The Pep Squad

図9.2 : 自動生成されたHTML

■ パラメータの指定

URLパラメータをSQL文に渡すことも可能だ。先ほどのサービスを修正して、顧客の所在地でフィルタできるようにしてみよう。次のようなSQL文を実行する。

```
SELECT * FROM customer
 WHERE state = 'NY'
```

このように、「NY」といった文字列がURL内のパラメータ（/customers?state=NY）として渡されるものとする。

Webサービス内のSQL文では、:parameter_nameという形式でパラメータを記述できる。URLに?parameter_name=valueという指定があれば、SQL文内の:parameter_nameがvalueで置換される。よって、Webサービスの登録は次のようになる。

```
ALTER SERVICE customers TYPE 'HTML'
```

```
AUTHORIZATION OFF USER DBA
AS SELECT * FROM customer
    WHERE state = :state
```

Webブラウザから、http://localhost/customers?state=NYにアクセスすると、ニューヨーク州に住む顧客だけが表示される。ここではHTTPのGETメソッドを用いてアクセスしたが、POSTメソッドでも動作する。

9.3.3 自分でフォーマットプログラムを作成

文字列操作を行うスクリプトを書いて、HTMLドキュメントを自由に作成できる。以下では、Hello Worldサンプルを作ってみることにする。/hello_worldにアクセスすると、「Hello World!」と書かれたHTMLが表示されるようにしたい。まず、文字を連結してHTMLを書くストアドプロシージャを作成する。

```
-- stored procedure
CREATE PROCEDURE hello_world_proc()
RESULT (html_doc long varchar)
BEGIN
   CALL dbo.sa_set_http_header( 'Content-Type', 'text/html' );
   SELECT '<html>¥n'
        || '<head><title>Hello World</title></head>¥n'
        || '<body>¥n'
        || '<h1>Hello World!</h1>¥n'
        || '</body>¥n'
        || '</html>¥n';
END;
```

次に、/hello_worldというURLに対して、このストアドプロシージャが実行されるようにWebサービスを設定する。Webサービスのサービスタイプは自動整形機能を使わないため、「RAW」を指定する。

```
-- Webサービス
CREATE SERVICE hello_world TYPE 'RAW'
AUTHORIZATION OFF
USER DBA
AS CALL hello_world_proc;
```

これでWebブラウザからhttp://localhost/hello_worldにアクセスすると、「Hello World!」と書かれたWebページが表示される。

静的なHTMLではなくデータベース内のデータを用いた動的なコンテンツを生成したいならば、ストアドプロシージャ内でデータベースにアクセスし、データを文字列でつなげていけばよい。

9.3.4 SOAP

SQL Anywhereでは、SOAPに対応したWebサービスを作成するのも容易だ。CREATE SERVICE文のサービスタイプで「SOAP」と指定するだけで、指定されたサービスがSOAPメッセージを解するWebサービスとなる。

たとえば、departmentテーブルの全データを返す処理をSOAPでやり取りしたいとする。処理は簡単なSELECT文ではあるが、ここではストアドプロシージャとして実装する。

```
CREATE PROCEDURE "sp_sample"()
BEGIN
  SELECT * FROM department;
END;
```

次に、CREATE SERVICE文でこのストアドプロシージャをWebサービス化する。今までと同様であるが、サービスタイプの指定を「SOAP」とすればよい。

```
CREATE SERVICE "sqlservice/sample" TYPE 'SOAP'
```

```
AUTHORIZATION OFF SECURE OFF USER DBA
AS CALL sp_sample();
```

これで/sqlservice/sampleというURLを持つWebサービスが登録された。

Webサービスに対するWSDLを提供するには、新たなWebサービスを作成する。

```
CREATE SERVICE "sqldish" TYPE 'DISH'
AUTHORIZATION OFF USER DBA
USING sqlservice;
```

サービスタイプの指定を「DISH」にするとWSDLを提供するサービスとなる。あとはUSING句でこのWSDLの管理下となるURLを指定すればよい。

付録 A

SQL Anywhere Studio 9.0 新機能ガイド

A.1 SQL Anywhere Studio 9.0概要

SQL Anywhere Studio 9.0は前バージョンに比べ、Adaptive Server Anywhereのスケーラビリティとパフォーマンスが大幅に向上している。キャッシュ管理システムは再設計され、多数のユーザによる同時実行性の性能も向上している。Adaptive Server Anywhereは、これまでも数百の同時ユーザ環境で動作していたが、今回の改善により、処理できる規模の向上が期待できる。

クエリ処理の性能も向上している。新しいアルゴリズムを採用することにより、以前のバージョンの何分の1かの時間で、より多くのクエリを実行できるようになった。

パフォーマンスの向上を目的として、データベースサーバには多数の変更が加えられている。新機能のIndex Consultant（インデックスコンサルタント）を使用すると、管理者は適切なインデックスを選択するだけでパフォーマンスを簡単に最適化できるようになる。

以前のバージョンでMobile Linkを実行するには、クライアントが同期を開始する必要があった。Adaptive Server Anywhere 9.0ではサーバから同期を開始できる機能が追加されたため、緊急な変更をリモートデータベースに送信することができる。同期処理をサーバから開始する方式は、クライアントデータベースが埋め込みシステムで動作している場合にも適している。

新バージョンのUltra Lightでは、リレーショナルデータストレージの利点をより多くのアプリケーション開発者が簡単に利用できるようにUltra Lightコンポーネントが用意された。

次節以降では、代表的な新機能について簡単に説明する。そのほかの新機能については、最後の「A.5 その他の新機能一覧」に掲載している。そちらもご覧いただきたい。

A.2 Adaptive Server Anywhere 9.0 の主な新機能

A.2.1 パフォーマンスの向上

次に示す新機能は、パフォーマンスの向上のための代表的な機能の一部である。これらの機能を使用するのに、ユーザが意識する必要はない。アプリケーションを書き換えなくても、バージョンアップするだけでパフォーマンスは向上する。

1. コストベースのサブクエリの最適化

オプティマイザは、サブクエリを使用したときの最適化の範囲を大幅に拡張された。結合（JOIN）処理として記述し直すには複雑すぎるサブクエリも、クエリの一部としてそのまま最適化できるようになった。このため、サブクエリ内部のクエリと、本体部分のクエリを合わせた形でアクセスプランが生成されるようになった。

2. 逐次スキャンのパフォーマンスを向上

逐次テーブルスキャンでデータベースページから行を読み込む場合、行をバッファにコピーしてからユーザに行を返すように変更された。それ以降のクエリで同一のテーブルへアクセスする場合、バッファから読み込むことでパフォーマンスが向上する。

3. プロシージャの文でプランが高速にキャッシュされる

アクセスプランをキャッシュする範囲が拡張された。プロシージャが結果セットを呼び出し元の環境に返すストアドプロシージャ内のクエリも含まれるようになった。この機能強化により、クエリ最適化のコストを低減できる。

4. 各インデックスが更新されるたびにインデックスの統計情報が維持される

各インデックスが更新されるたびに、カタログテーブルのインデックスを含めてすべてのインデックス統計情報が維持されるようになった。この改善により、実質的にパフォーマンスを損なうことなくオプティマイザに対して正確な統計情報を取得できる。正確な統計情報が維持されるため、候補となるインデックスの選択性を高め、それを基準としたオプティマイザの負荷を軽減できる。

5. 効率的に実行されるSELECT～TOP *n*句

「売上金額の上位10社」などを求めるTOP *n*句のクエリ実行アルゴリズムが新しくなり、実行速度が向上した。

6. Windows CEの最大キャッシュサイズ

SQL Anywhere Studio 9.0.2からは、Windows CEの最大キャッシュサイズが「デバイス上で使用可能なメモリ」まで利用できるようになった（以前のバージョンでは、最大キャッシュサイズは32MBだった）。

A.2.2 基本機能の強化

1. CREATE INDEX文で計算カラムをインデックス化

テーブルに新しい計算カラムを追加し、その計算カラムに対してインデックスを作成できるようになった。たとえば計算カラムとして、数量×単価の金額カラムを追加し、そのカラムにインデックスを作成することで、「金額が1万円以上」といった検索がインデックススキャンで実行できる。

2. すべてのコンテキストでORDER BY句を使用可能

以前のバージョンでは、ビュー定義、サブクエリ、またはUNION演算内の多くの

SELECT文ではORDER BY句を使用できなかったが、その制限が取り除かれた。

3. TOP句にSTART AT句を指定

TOP句にオフセット（取得する位置）を指定するSTART AT句が追加された。たとえば、「売上の高い」企業順に並べ、そのうち11～15番目の会社を表示したい、といった際には以下のクエリで対応できる。

```
SELECT TOP 5 START AT 11 …;
```

4. EXIT文の強化

Interactive SQLのEXIT文は、Interactive SQLに終了コードを設定できるようになった。このEXIT文を利用すれば、バッチファイルの中で、処理の成功や失敗などをERRORLEVELとして取得可能になるため、エラーハンドリングなどが容易になる。

5. オンライン分析処理（OLAP）機能の関数が追加

OLAP機能として合計と小計を示す結果セットを取得するROLLUP・CUBE演算や、ランキング表示するのに便利なRANK・DENSE_RANK関数、文字列の暗号化や圧縮・解凍などの統計・文字列関数が追加された。

6. SELECT ... INTO 新規テーブル名

SELECT … INTO句は、選択（SELECT）した結果セットの内容で新規テーブルの作成とデータの挿入を行う。テーブル名が#から始まる場合は、テンポラリテーブルとして作成される。#以外の場合は、通常のテーブルとして作成される。

A.2.3 チューニング・運用管理支援

1. Index Consultant（インデックスコンサルタント）

Index Consultantは、システムの負荷情報を取得し、その内容を分析・評価し推奨するインデックスのSQLスクリプトを自動生成する。

Interactive SQLからは、単一のクエリをIndex Consultantを使用して分析を行える。Sybase CentralからIndex Consultantを起動すると、明示的に開始、終了を指定し、データベースに対する各種要求を収集し、分析することが可能になる。

2. テーブルヒント句の追加

SELECT文で使用される「WITH（XLOCK）」は、FROM句の新しいテーブルヒント機能だ。XLOCKでは、ヒントが指定したテーブルの抽出した行を対象に排他的ロックする機能が追加された。影響を受ける行は、トランザクションの終わりまでロックされたままになる。

3. インデックスヒント句の追加

SELECT文の拡張機能である「WITH (INDEX (インデックス名))」を使えば、クエリオプティマイザが選択するプランに対して、ヒント句で指定したインデックスを強制的に指定できる。

4. 要求ログのフィルタリングとホスト変数のサポート

パフォーマンス分析などを行うときに有用な、要求レベルログの出力機能が強化された。特定の接続や特定のデータベースの要求をフィルタリングしてロギングしたり、ホスト変数値を要求ログに出力する機能が追加された。

5. テンポラリテーブルをNOT TRANSACTIONALとして宣言

NOT TRANSACTIONAL句はテンポラリテーブルに対してのみ使用できる。

NOT TRANSACTIONALを指定して作成されたテーブルは、コミットまたはロールバックの影響を受けない。

状況によっては、NOT TRANSACTIONAL句を使用するとパフォーマンスが向上する。これは、NOT TRANSACTIONALを指定したテンポラリテーブルへの操作は、ロールバックログへの書き込みを行わないためである。

6. 詳細なリターンコードを返す検証ユーティリティ

検証ユーティリティ（dbvalid）は、異常の発生原因を示す、より詳細なリターンコードを返すようになった。以前のバージョンでは、成功か失敗しか返さなかった。

7. Sybase CentralとInteractive SQLの高速起動

Windows上のSybase CentralまたはInteractive SQLには、アプリケーションの起動時間の短縮を目的とする高速ランチャが組み込まれている。Adaptive Server Anywhereが起動すると、2つのバックグラウンドプロセスが開始する。この2つのプロセスはdbisqlg.exe（Interactive SQL）とscjview.exe（Sybase Central）のインスタンスである。これらの実行プログラムは両方とも、ユーザがログインすると起動するが、起動させないようにすることもできる。

8. Sybase Centralの操作性向上

Sybase Centralのユーザインターフェイスが変更され、操作性が向上した。たとえば、これまでプロパティシートやダイアログボックスなど左ウィンドウ枠内のフォルダに表示されていた情報の多くが、右ウィンドウ枠内のタブに表示されるようになった。

また、データのアンロード（unload）の際には、1回の操作でデータをアンロードできるようになった。

9. デバッガ機能をSybase Centralに統合

ストアドプロシージャとJavaクラスの両方をデバッグできるデータベースオブジェクトデバッガがSybase Centralに統合された。ユーザインターフェイスもより使いやすいように変更された。

10. グラフィカルなプランの強化

Interactive SQLのグラフィカルなクエリアクセスプランの表示が強化された。特に、以下の点が強化されている。

- アクセスプランの処理を表すコンテナ間で、対象の行が多い場合、線の太さを変えて表示される。
- 特に処理時間がかかっているコンテナは、赤い枠線で強調表示される。
- 統計情報の表示が拡張された。
- アクセスプランを印刷できるようになった。

11. SNMPエージェント

SNMP（Simple Network Management Protocol）管理アプリケーションから、Adaptive Server Anywhereデータベースを管理することができる。1つのエージェントから、複数の異なるマシンで動作している複数のデータベースサーバをモニタリングすることもできる。

12. ディスクが満杯の場合のコールバックサポート

データベースサーバオプションの-fcを使用すると、ファイルシステムが満杯の状態になったときに、ユーザへの通知を行い、適切に対処できるDLLを指定することができる。

13. データベースのデッドロックをレポートする機能

新たに追加されたデータベースオプションLOG_DEADLOCKSと、システムストアドプロシージャのsa_report_deadlocksを使用して、デッドロックに関係する情報を取得できるようになった。

LOG_DEADLOCKS オプションをオンにすると、データベースサーバは、デッドロックに関する情報を内部バッファに記録する。そして、sa_report_deadlocksを呼び出すことによって、この内部バッファからデッドロック情報を取得できる。

14. FIPS承認セキュリティのサポート

WindowsプラットフォームでCerticomのFIPS 140-2承認ソフトウェアで保護された、安全な通信を使用できるようになった。

A.2.4 パッケージなどへの組み込み利用のしやすさ

1. データベースサーバのキャッシュウォーミング

キャッシュウォーミングは、データベースに対して実行される初期クエリの実行時間を減らすのに役立つ。キャッシュウォーミングとは、前回起動したときにデータベースがメモリ内にキャッシュした内容を事前にロードする機能である。

データベースページの収集は、データベース起動時に指定した最大数に到達するか（値はキャッシュサイズとデータベースサイズに基づく）、データベースが停止するなどの条件に合致するまで収集を続ける。収集が完了すると、収集したページはデータベースに記録される。データベースを次回起動したときに、同一または類似のクエリがデータベースに対して実行された場合、記録されたキャッシュを使用することでパフォーマンスを向上させることができる。パッケージなどで組み込みデータベースとして利用する場合や、データベースの停止・起動が多いパッケージでは効果的な機能だ。

2. InstallShieldプロジェクト

SQL Anywhere StudioにはInstallShield のMerge ModuleとObjectのプロジェクトが用意されている。このプロジェクトファイルを利用して、SQL Anywhere Studioと作成したアプリケーションを一緒にインストールすることもできる。現在マシンにインストールされているソフトウェアを再配備する場合などにも有効だ。

A.2.5 最新のIT技術との融合

1. Microsoft .NETとの密接な連携

Adaptive Server Anywhere 9.0は、Microsoft .NETアーキテクチャで動作するように作成されている。Visual Studio .NETを使用してAdaptive Server Anywhereにアクセスするための「Adaptive Server Anywhere .NET」「OLE DB .NET」「ODBC.NET」データプロバイダが提供されている。

.NET環境において、Adaptive Server Anywhere .NETデータプロバイダは、Adaptive Server Anywhereに対するネイティブアクセスを提供する。サポートされているほかのプロバイダとは異なり、このデータプロバイダは Adaptive Server Anywhereと直接通信を行うため、ブリッジテクノロジを必要としないという利点がある。また、Mobile LinkやUltra Lightコンポーネントも、.NETを使用した実装・開発を行うことが可能となっている。

2. Perlインターフェイス

DBIモジュールとDBD::ASAnyドライバをインストールすると、Perlで作成されたスクリプトからAdaptive Server Anywhereデータベースにアクセスできる。

3. 64ビットバージョン対応

インテルItanium IIチップ搭載のWindows Server 2003では、Adaptive Server

Anywhereデータベースサーバ、クライアント、Mobile Link同期サーバ、SQL Remoteのコンポーネントは、64ビットバージョンとして動作する。

4. PHPモジュール

SQL Anywhere PHPモジュールを使用すると、PHPスクリプト言語から Adaptive Server Anywhereデータベースにアクセスできる。

A.2.6 Webシステムへの対応

1. データベース内のHTTPサーバ

Adaptive Server Anywhereデータベースサーバが Webサーバ（HTTPサーバ）として動作できるようになった。このため、Adaptive Server Anywhereデータベースと任意のWebブラウザだけを使用して、Webベースのアプリケーションを作成し、実行できる。

データベースサーバは、標準のSOAP（Simple Object Access Protocol）要求に加えて、標準のHTTP要求とHTTP/S要求も処理する。使用できるサービスタイプは、HTTP、HTTPS、XML、RAW、SOAP、DISH。なお、DISHは、SOAPサービスのハンドラである。

2. Webサービスクライアント

Adaptive Server Anywhere 9.0.2より、上記のWebサーバに加え、Webサービスクライアントとしても動作できるようになった。このため、インターネット上で使用できる標準のWebサービスに加えて、Adaptive Server Anywhere Webサービスにアクセスするストアドプロシージャとストアド関数を作成できる。

3. XMLのサポート

Adaptive Server Anywhere 9.0では、XML文書の保存、リレーショナルデータをXML形式でエクスポート、XML形式のファイルのインポート、リレーショナルデータのクエリからXML形式のデータを返すなど、XMLを広範囲にわたってサポートしている。

A.3 データ同期（Mobile Link）の進化

1. サーバによって開始される同期

以前は、Mobile Linkを実行する場合、クライアントから同期を開始する必要があったが、Adaptive Server Anywhere 9.0では、サーバから同期を開始できる機能が追加された。このため、緊急度の高い変更情報をリモートデータベースに送信することができるようになった。

サーバ起動同期を使用すると、Mobile Link同期を統合データベースから開始できる。これは、リモートデータベースが統合データベースにデータをアップロードできることに加え、データの更新をリモートデータベースにプッシュできることを意味している。このMobile Linkコンポーネントには、どの変更内容が統合データベースで発生したら同期を開始するかを決定したり、プッシュするメッセージを受信する、ユーザを選択する方法などのオプションが用意されている。

2. ファイルベースのダウンロード

ファイルベースのダウンロードでは、ダウンロードした同期の変更内容をファイルに保存し、リモートデータベースに配布する機能が追加された。配布方法としては、CDやフロッピーディスクなどのメディアを送付する方法や、電子メール、FTPサイトからダウンロードするなど一般的な方法で配布できる。

3. アップロード／ダウンロード専用同期処理

Adaptive Server Anywhereクライアントは、アップロード専用とダウンロード専用の同期を選択できるようになった。また、同期処理を制御する新しいスクリプトが追加された。

4. リダイレクタの強化

Apache Webサーバ用のネイティブなリダイレクタが追加された。このため、Apacheを使用する場合、Tomcatは必要なくなる。また、M-Business Anywhereリダイレクタも提供している。

5. カスタムユーザ認証機能の追加

カスタムユーザ認証を行うための新しいスクリプト機能が追加された。カスタムユーザ認証メカニズムを使用する理由として、既存のユーザ認証スキームとの統合や、Mobile Linkメカニズムにはない、パスワードの最小長チェックや有効期限といったカスタム機能の提供がある。LDAP、POP3、IMAPサーバを使用した認証を簡単に行えるスクリプトも用意されている。

6. リモートデータベースの進行状況把握

リモートデータベースの進行状況（オフセット、最終アップロード時間、最終ダウンロード時間）を同期サーバ側で確認する機能が追加された。

7. Mobile Linkのリターンコードの強化

dbmlsyncセッションの中で複数の同期処理を実行している場合などで、同期処理の成功や失敗を追跡し、記録することを支援するために、sp_hook_dbmlsync_process_return_codeという新しいクライアントイベントフックプロシージャが追加された。また、sp_hook_dbmlsync_abortフックの#hook_dictテーブルには、新し

い値（リターンコード）が設定される。

8. スキーマの変更がない場合のdbmlsyncのパフォーマンス向上

dbmlsyncはデフォルトで、各同期の前にスキーマ情報をロードしなくなった。これにより、低速なデバイスにおける同期パフォーマンスが向上した。

9. Adaptive Server Anywhereクライアントのアプリケーション統合の向上

dbmlsync統合コンポーネントは、同期処理をアプリケーションに追加するのに役立つ。dbmlsync統合コンポーネントはビジュアルまたは非ビジュアルな形式で使用でき、Adaptive Server Anywhereクライアントの動作を管理する一連のメソッド、プロパティ、イベントを提供する。

サポートしている開発環境としては、MicrosoftのVisual Studio .NET、Visual Basic 6.0、Embedded Visual Basicがある。

10. スキーマをアップグレードまたは変更する手段の追加

配備されたリモートデータベースのスキーマを変更する機能が追加された。この機能は、sp_hook_dbmlsync_download_endのあとに実行されるsp_hook_dbmlsync_schema_upgradeストアドプロシージャを使用して、スキーマを修正するSQLスクリプトを実行する。

11. クライアントのトランザクションレベルのアップロード

以前のバージョンでは、Mobile Linkはリモートデータベースで同じ行を複数回変更しても、アップロードストリームで1つのトランザクションにすることで、効率の良い処理を提供していた。しかし、特定の状況では、リモートデータベースでトランザクションが発生した単位でトリガを起動したい場合がある。そのような場合への対応として、ログごとにアップロードするオプションが追加された。

12. Mobile Linkモニタデータのエクスポート

Mobile Linkモニタデータの保存として、バイナリファイル、CSVファイルに加え、Microsoft Excelファイル、ODBCまたはjConnect接続を使用してリレーショナルデータベースに対してエクスポートする機能が追加された。

13. リモートデータベースの削除と再作成の簡単化

Adaptive Server Anywhereクライアントサブスクリプションの最初の同期が、いつでも動作するようになった。

14. Mobile Linkサーバがビジーな場合のクライアントの同期待機の回避

Mobile Linkサーバがビジーな場合、クライアントが同期を待ち続けることを回避できるようになった。

15. Ultra Light用の新しい同期設定ツール

Ultra Lightスキーマペインタが、Adaptive Server Anywhere統合データベース用のデータベーステーブルとトリガに加えて、Mobile Link同期スクリプトも生成できるようになった。

A.4 組み込み用・超小型データベースの革新（Ultra Light）

A.4.1 簡単な開発作業

Ultra Lightのプログラミングインターフェイスとして、従来の「静的インターフェイス」に加え、新しく「Ultra Lightコンポーネント」を利用してUltra Lightアプリケーションを開発することが可能になった。

「静的インターフェイス」の開発では、スキーマや利用するSQL文（静的SQL）をリファレンスデータベースに登録する。このリファレンスデータベースを、Ultra Lightジェネレータを使用して生成したライブラリと、アプリケーションをリンクさせてコンパイルすることでUltra Lightアプリケーションが作成される。

「Ultra Lightコンポーネント」の開発は、一般的なデータベースを利用したアプリケーション開発と類似している。スキーマを定義するには、Ultra Lightスキーマペインタを使用する。Ultra Lightデータベースには、簡単にアクセスできるオブジェクトベースのプログラミングインターフェイスを使用し、アプリケーション中のSQL文（動的SQL）からアクセスする。

サポートされている開発言語ごとに、以下のコンポーネントが用意されているため、開発作業もより簡単に行える。

Ultra Light コンポーネント

- Ultra Light .NET

  Ultra Light .NETは、C#またはVisual BasicによるUltra Light開発を.NET Frameworkと.NET Compact Frameworkに統合する。Ultra Lightコントロールが Visual Studio .NET 2003ツールボックスに追加され、利用しやすくなった。Ultra Light .NETは、ADO.NETプログラミングインターフェイスをサポートするようになった。ADO.NETは、Ultra Lightに対して業界標準のインターフェイスを提供し、大規模なアプリケーションの、Adaptive Server Anywhereへの簡単な移行パスも提供している。

- Ultra Light ActiveX

  Ultra Light ActiveXを使えば、Ultra Light開発を、Microsoft Embedded Visual BasicとPocket Internet Explorer（JavaScriptを使用）に統合できる。

- Native Ultra Light for Java

  Native Ultra Light for Javaは、小型デバイス用のJava開発モデルを提供する。

- Ultra Light C++コンポーネント

  Ultra Light C++コンポーネントを使えば、C++開発者は、その他のUltra Lightコンポーネントの機能にアクセスできる。これには、複数のアプリケーションからのアクセスをサポートする、Ultra Lightランタイムのバージョンへのアクセスも含まれる。

- Ultra Light for MobileVB

  Ultra Light for MobileVBは、Ultra Light開発を、Visual Basicの強力な拡張版に統合する。また、AppForge Crossfireを使用して、Visual Basic .NETプログラミングをサポートするようになった。AppForge MobileVBおよびAppForge Crossfireにより、Palm OSとWindows CEデバイスの両方において、Visual Basic、Visual Basic .NETを使用したアプリケーションの短期開発が可能になる。

- Ultra Light for M-Business Anywhere

  Ultra Light for M-Business Anywhereは、M-Business Anywhereを使用した開発を可能にする。

A.4.2 より便利になった機能

1. 動的SQLでCREATE文およびDROP文をサポート

CREATE TABLE/INDEX文とDROP TABLE/INDEX文が、動的SQLで使用できるようになった。Ultra Lightコンポーネントのユーザは、これらの文を使用して、Ultra Lightデータベースのスキーマを変更できる。

2. 動的SQLからのトランザクションの制御

COMMIT文とROLLBACK文が、動的SQLで使用できるようになった。Ultra Lightコンポーネントのユーザは、これらの文によって、SQL 文を使用してトランザクションを制御できる。

3. 動的SQL SELECTの強化

WHERE句またはHAVING句の探索条件でサブクエリを使用できるようになった。これらは、FROM句の抽出テーブルとしても使用できる。

4. 複数データベースの接続をサポート

Ultra Lightコンポーネントを用いて、データベースファイル名などを指定して、1つのアプリケーションから複数の接続要求を発行し、複数のデータベースに接続することができるようになった。

5. エラー情報の拡張

Ultra Lightコンポーネントを使用して構築されたアプリケーションに対して、より多くのエラー情報が提供されるようになった。

6. Ultra Light Interactive SQLユーティリティをサポート

Ultra Light Interactive SQLユーティリティを使えば、Ultra Lightデータベースに対するSQL文のテストと、Ultra Lightデータの変更が行える。Ultra Light Interactive SQLユーティリティはクエリプランも表示するので、パフォーマンスの問題も診断できる。

7. クエリ最適化の向上

以前のバージョンでは、テーブルがアクセスされる順序はクエリ内でのテーブルの

出現順序だったが、テーブルに効率良くアクセスできる順序になるようにクエリが最適化されるようになった。データベース内で適切なインデックスが定義されている限り、オプティマイザによるクエリの実行パフォーマンスの向上を期待できる。

8. データベース管理のためのコマンドラインユーティリティ

Windowsマシン上のコマンドラインユーティリティを使って、Ultra Lightデータベースを操作することが簡単になった。以下のユーティリティはアプリケーション開発時に特に効果的に使える。

- ulcreate ――Ultra Lightデータベースを作成する
- ulload ――XMLファイルからUltra Lightデータベースにデータをロードする
- ulsync ――Ultra Lightデータベースを同期する
- ulunload ――Ultra LightデータベースからXMLファイルにデータをアンロードする

A.5 その他の新機能一覧

以下の内容の詳細については、マニュアルを参照してもらいたい。

【Adaptive Server Anywhere 9.0のその他の新機能】

- SQLの強化
  - ・SELECTの前に共通表式WITH文をサポート
  - ・組織表や部品表などのアクセスに便利な再帰を実現する共通テーブル式であるWITH RECURSIVE文をサポート

・INTERSECT/EXCEPT集合演算のサポート
・SELECT文にストアドプロシージャの結果セットを指定可能
・制約に名前を指定可能
・FROM句の中でリテラル抽出テーブルによる外部参照が可能
・EXECUTE IMMEDIATE文でエスケープ文字を柔軟に処理
・EXECUTE IMMEDIATE文が結果セットを返すクエリをサポート
・CREATE FUNCTION文とALTER FUNCTION文でTransact-SQL構文を使用
・複数の行の挿入時にオートインクリメントカラムの値を使用
・リモートデータアクセスでUUID/GUIDカラムを処理
・リモートデータアクセスでリモート接続に名前を指定可能
・式のデータ型を返す新しい関数の追加
・OUTPUT文にASISキーワードを指定可能
・インデックスと外部キーを変更
・サポートされるすべての言語のイベントスケジュールで、英語のフルスペルの曜日名と省略形の曜日名を認識可能
・プロシージャテキストを隠してロジックの機密を保持
・LOAD TABLEの強化
・SQL文の変数の拡張サポート
・SET文の強化
・ALTER TABLE文の拡張
・ALTER VIEW文の強化
・MESSAGE文の強化
・ネイティブのUNIQUEIDENTIFIERデータ型の追加
・RESOLVE UPDATEトリガでCONFLICT関数を指定可能
・ALTER DOMAIN文の追加

・別の接続からメッセージを受信してWAITFORをウェイクアップ可能
・プロシージャのNO RESULT SET句をサポート
・プロシージャプロファイリングの強化
・作成または変更前にリモートサーバのテストが可能
・INPUT文とOUTPUT文にESCAPES句を指定可能
・クエリプランに抽出テーブルを表示可能
・インデックス作成時にカラム統計を更新可能
・dbfhideで初期化ファイルを難読化可能

- 管理とスケーラビリティの強化
  ・2つの新しいサーバプロパティ（CommandLineとPlatformVer）
  ・新しいストアドプロシージャ（sp_remote_primary_keys）
  ・接続の通信リンク名を返すプロパティの追加（connection_property）
  ・NetWareで完全な文字セット変換をサポート
  ・アンロードユーティリティでカラムリストのアンロード機能追加
  ・LDAPによるデータベースサーバの登録
  ・接続数が多い場合の処理の向上
  ・要求ログのフィルタリングとホスト変数のサポート
  ・BACKUP文とdbbackupでログコピーの名前を変更可能
  ・START DATABASE文でチェックポイントでのログトランケーションと読み込み専用モードを指定可能
  ・Adaptive Server Anywhereにおける異なる監査オプションのサポート
  ・event_parameter関数に渡せる3つの新しい値（ScheduleName/AppInfo/DisconnectReason）を追加
  ・ネットワークサーバに同時に接続しているユーザ数を示す新しいサーバプロパティ
  ・サービス作成（dbsvc）ユーティリティでサービスの起動と停止が可能

・LocalOnly通信パラメータ（LOCAL）をサポートするネットワークサーバをサポート
・Address Windowing Extensions使用時の最小データベースサーバキャッシュサイズを変更可能
・ドライブタイプを指定する新しいデータベースプロパティ
・Adaptive Server Anywhere NetWareの高速化
・NetWareにおける外部関数の強化
・エラーメッセージの言語を接続ごとに指定可能
・プロセッサタイプを識別する2つの新しいサーバプロパティの追加
・データベースのパスワードの大文字と小文字の区別
・BACKUP文の強化
・アンロードユーティリティ（dbunload）の強化
・サーバ検索（dblocate）ユーティリティの強化
・コンソールユーティリティによる統合化ログインのサポート
・データベースサーバを再起動せずに要求レベルのログファイルを変更可能
・sa_procedure_profileおよびsa_procedure_profile_summaryシステムプロシージャでシステムトリガに関する情報を追加
・新しいシステムテーブルを追加
・新しい照合順の追加
・DEDICATED_TASK オプションの追加
・データベースページチェックサムをサポート
・ODBC_DESCRIBE_BINARY_AS_VARBINARYオプション
・新しいPREFETCHオプション値
・db_locate_servers_ex関数
・複数のWebサービスフォーマットのサポート
・要求ログのサイズの管理

- sa_index_density と sa_index_levels への新しいカラムの追加
- サービス作成 (dbsvc) ユーティリティの新しいオプション
- データソース (dbdsn) ユーティリティの新しいオプション
- データベース検証ウィザードの強化
- Windows ユーザグループを使用する統合化ログインのサポート
- バックアップによってファイルの名前が変更されるときにトランザクションログの末尾の空きページが削除される
- リモートサーバ接続を明示的に閉じることが可能

- パフォーマンスの強化 (クエリの最適化)
  - コストベースのサブクエリの最適化
  - ヒストグラムに保持する頻度を決定する新しいアルゴリズム
  - 現在キャッシュされているクエリプラン数を表示する QueryCachedPlans プロパティ
  - パフォーマンスモニタ統計値の追加
  - UNLOAD 文と UNLOAD TABLE 文への APPEND ON/OFF オプションの追加
  - OPTIMISTIC_WAIT_FOR_COMMIT オプションの追加
  - db_extended_property 関数の追加
  - 2つの新しいデータベースプロパティの追加 (FileSize と FreePages)
  - サーバのクワイエットモードの強化
  - NetWare の内部実行スレッドのデフォルトのスタックサイズの増加
- 開発ツールと管理ツール
  - Adaptive Server Anywhere プラグインにおけるクリップボードサポートの強化
  - コンソールユーティリティの強化

・Interactive SQLの構文を強調表示するエディタ
・データベースユーティリティで@filenameパラメータをサポート
・Interactive SQLで結果の横に行番号を表示
・Interactive SQLを.SQLファイルのデフォルトエディタとして設定
・Interactive SQLの［コマンド履歴］ダイアログボックスの強化
・警告メッセージに「W」プレフィクスを追加
・Interactive SQLを使用すると、ファイルの読み込みと書き込みに使用されるコードを指定
・Interactive SQLによる統合化ログインのサポート
・Interactive SQLでは、結果セットの表示に使用されるフォントを設定
・Interactive SQLの使用時に、ファイルのブラウズに使用される初期フォルダを指定可能
・Sybase Centralでは、結果セットの表示に使用されるフォントを設定
・リモートサーバ作成ウィザードが現在のユーザの外部ログインの作成をサポート
・Sybase Centralによる統合化ログインのサポート
・Sybase Centralの［ビュー］メニューを使用してカラムをソート可能
・［外部キー］プロパティシートから外部キー設定を変更可能
・プロキシテーブルウィザードで主キーカラム情報を表示可能
・ユーティリティウィザードをキャンセル可能
・Sybase Centralはサービスの作成と編集時にdomain¥userの形式のアカウント名をサポート

【Mobile Linkバージョン9.0のその他の新機能】

- Moblie Link同期サーバの強化
  ・接続スクリプトbegin_publicationとend_publicationの追加

・文字列に埋め込まれたブランクを削除するオプションの追加
・.old拡張子を持つ新しいログファイルを開始するオプションの追加
・.NETとJava同期論理におけるエラーと警告の処理
・Mobile Linkシステムテーブルへの追加
・統合データベースに格納されるバージョン
・Mobile Link同期サーバがサポートする新しいuniqueidentifierデータ型
・新しいリダイレクタ
・指定のホストを無視するようにプロトコルを設定可能

- Adaptive Server Anywhereクライアントの強化
  ・ウィンドウメッセージによる同期の起動
  ・起動時にDLLをロード（Windows CE）
  ・同期処理の成功／失敗を追跡し記録することを支援する新しいイベントフックプロシージャの追加
  ・スケジューリングの強化
  ・Adaptive Server Anywhereクライアントのアプリケーション統合の向上
  ・再起動可能なダウンロード
  ・Adaptive Server Anywhereクライアントのトランザクションレベルのアップロード
  ・Mobile LinkクライアントとHTTPインフラストラクチャの統合の向上
  ・Mobile Linkへの接続が失敗すると新しいdbmlsyncフックが呼び出される
  ・接続エラーの検出の支援
  ・Ultra Light用の新しい同期設定ツール
  ・ミラーログのロケーション
- Ultra Lightクライアントの強化
  ・HotSync conduitのトラブルシューティング支援の強化

- パフォーマンスとモニタリングの強化
  - ・スキーマの変更がない場合のdbmlsyncのパフォーマンス向上
  - ・Windows CE上でのdbmlsyncのパフォーマンスの向上
  - ・Mobile Linkモニタをコマンドラインから起動可能
  - ・警告メッセージに「W」プレフィクスを追加
- サーバによって開始される同期
  - ・自動デバイス追跡
  - ・Sybase Centralの設定
  - ・オプションの配信確認
  - ・新しいListenerオプション
  - ・Palmの設定
  - ・接続起動同期機能の強化
  - ・Listenerポストアクションの強化
  - ・新しいアクション変数
  - ・デバイスサポートの増加
- Mobile Linkモニタ
  - ・モニタデータのエクスポート
  - ・モニタの強化情報
  - ・ソート機能の向上
  - ・ユーザインターフェイスの強化

【Ultra Lightバージョン9.0のその他の新機能】

- ・Pocket Internet Explorerのサポート
- ・接続パラメータ
- ・MobileVBコンポーネントのドラッグ＆ドロップ

・マルチプロセスアクセス
・同時同期処理
・Palm OSの強化
・エラー情報の拡張
・Windows NT/2000/XP上で使用可能なUnicodeライブラリ
・配備プラットフォームとしてWindows XPをサポート
・Ultra Lightに対するODBCインターフェイス
・C++インターフェイスの混在
・Ultra Light C++コンポーネントのCodeWarriorステーショナリ
・Ultra Light C/C++のエラー処理の向上
・Ultra Lightコンポーネントは Ultra Lightエンジンを使用可能
・同期プログレスイベントの追加
・スキーマのアップグレードの監視
・再起動可能なダウンロード
・新しいWindows CEプラットフォームのサポート
・ULPalmLaunchとULPalmExitは不要
・Ultra Lightデータベースプロパティ
・UNIQUEIDENTIFIERデータ型の導入
・IF式とCASE式の追加
・テーブル名に所有者名を指定可能
・クエリプランの表示
・Ultra Lightクエリプランの記述の強化
・動的SQLの強化
・Ultra Light for MobileVBの強化
・Palm開発者はバージョンに依存しないプレフィクスファイルを使用可能
・Ultra Light for M-Business Anywhereの強化

・Native Ultra Light for Javaの強化
・信用された証明書を永続的な記憶領域から取得できる
・Mobile Link クライアントとHTTPインフラストラクチャの統合の向上
・参照整合性違反の同期通知
・スキーマペインタからの同期スクリプトの生成

【SQL Remoteバージョン9.0新機能】

・警告メッセージに「W」プレフィクスを追加
・ミラーログのロケーション
・RESOLVE UPDATEトリガのCONFLICT関数

付録B

SQL Anywhereのオンラインリソースと参考文献

B.1 オンラインリソース

SQL Anywhereの情報を得られるインターネット上の主なリソースを挙げる。

■ 日本語

- アイエニウェア・ソリューションズ社のWebサイト

  http://www.ianywhere.jp/

- ニュースグループ

  news://forums.sybase.com/ianywhere.public.japanese.general

  アイエニウェア・ソリューションズ社が発売している製品の開発に関する情報交換の場。製品の問題点の解決やソリューション情報の提供や、アイデアの交換などをユーザ同士で行うことを目的とし、アイエニウェア・ソリューションズ社が管理・運営している。NNTP対応ニュースリーダで購読できる。

■ 英語

- iAnywhere Solutions社のWebサイト

  http://www.ianywhere.com/

- ニュースグループ

  製品ごとのニュースグループがいくつかある。主なものは次のとおり。詳細は、http://www.ianywhere.com/support/newsgroups.htmlを参照。

  - □ news://forums.sybase.com/sybase.public.sqlanywhere.general
  - □ news://forums.sybase.com/sybase.public.sqlanywhere.mobilink
  - □ news://forums.sybase.com/sybase.public.sqlanywhere.replication
  - □ news://forums.sybase.com/sybase.public.sqlanywhere.linux
  - □ news://forums.sybase.com/sybase.public.sqlanywhere.ultralite

B.2 参考文献

本書を執筆するにあたり参考にした文献を以下に挙げる。これらは同時に推薦するものでもあるので、簡単な感想を載せておく。目次などの詳細は、オンラインで検索していただきたい。

- 製品マニュアル
  詳しく解説されているため、マニュアルを超える内容を執筆するのに苦労したくらいである。PDF形式とWinHelp形式の両方があり、日本語版には翻訳もある。Developer Editionにも製品マニュアルは同梱されている。
- TechnoteおよびWhitepaper
  製品Webサイト上にも情報があり、特に、TechnoteとWhitepaperは有益だ。最新の情報が頻繁に掲載されるので、定期的にチェックするとよい。
- 『SQL Anywhere Studio 9 Developer's Guide』Breck Carter著、Wordware Publishing, Inc.、2004
  ニュースグループで回答する側として活躍するTeam Sybaseの一員Breck Carter氏による解説。本書では省略したSQL構文やストアドプロシージャの書き方について詳しく述べられている。
- 『Database Management Systems, Third Edition』Raghu Ramakrishnan、Johannes Gehrke著、The McGraw-Hill Companies, Inc.、2002
  学術的な見地からデータベース（特にRDBMS）の仕組みが述べられ、データベースを開発するための基礎知識が書かれている。内容は少々お堅いが、ストレージやインデックス・オプティマイザ・排他制御などが詳しく、とても参考になる。

- 『おら！オラ！オラクル』木脇高太郎著、翔泳社、2003
  書き方はポップだが、実験しながら機能を説明する内容はとても興味深い。ただし、2003年5月に刊行された本なので、最新のOracleの機能を知るには物足りないかもしれない。RDBMSの仕組みを解説する日本語の書籍はOracleを対象にしたものが圧倒的に多く、避けて通れないのが残念だ。

付録C　付属CD-ROMについて

C.1 収録内容

付属CD-ROMには、日本語版SQL Anywhere Studio 9.0.2 Windows版とLinux版それぞれのDeveloper Editionが収録されています[1]。Developer Editionとは、社内評価や開発を目的としたSQL Anywhereの利用に対して無償でライセンスするもので、有償の製品版と比べて機能制限はありません。詳細は「特定製品に関するライセンス条件」をお読みください。

Developer Editionのインストール時にはライセンスキーが必要となります。アイエニウェア・ソリューションズ社のWebサイト（http://www.ianywhere.jp/）からユーザ登録（無償）を行うと、キーが発行されます。なお、キーを入力せずにインストールすると、Developer Editionではなく60日間有効の評価版となるのでご注意ください。

また、Disc 1の8Guide_PDFディレクトリには『SQL Anywhere Studio 8 公式デベロッパーズガイド』（森脇大悟著、サイベース株式会社監修、翔泳社）のPDFもあわせて収録されています。

- Disc 1
  - 日本語版SQL Anywhere 9.0.2 Windows版 Developer Edition（32-bit/Windows CE）
  - 『SQL Anywhere Studio 8 公式デベロッパーズガイド』PDF
- Disc 2
  - 日本語版SQL Anywhere 9.0.2 Linux版 Developer Edition（32-bit）

1　64ビット版のWindows/Linuxに対応したものは、アイエニウェア・ソリューションズ社のWebサイトからダウンロード可能。

C.2 特定製品に関するライセンス条件

SQL Anywhere® Developer Edition

ご使用にあたって

「ソフトウェア・ライセンス契約」に規定されたライセンス条件に加え、以下に追加した権利と制限が、第 1 条に規定する「本プログラム」に適用されるものとします。

注意 : SQL Anywhere Developer Edition は無償で提供されます。ただし、本ライセンス条件への同意により、配布、再許諾その他いかなる方法であっても本プログラムを第三者に譲渡または使用許諾しないことに同意したものとみなされます。

第1条（定義） 使用される用語は、特に指定しない限り、アイエニウェア・ソリューションズ株式会社(以下「当社」といいます）のソフトウェア・ライセンス契約に定めるものと同義です。本ライセンス条件において「本プログラム」は、SQL Anywhere Studioおよびそのすべてのコンポーネントを意味します。このバージョンのSQL Anywhere Studio は、以下のコンポーネントから構成されています。

- Adaptive Server® Anywhere DBMS（クライアント・バージョンおよびサーババージョン）
- Adaptive Server® Anywhere同期付きパーソナル・データベース
- Adaptive Server® Anywhere DBMS Ultra Lightエディション
- Mobile Link同期サーバ

- Sybase Central™
- jConnect® 5.5
- Physical Architect™（評価版）
- InfoMaker®（評価版）

注意：上記のコンポーネントは、ある特定のオペレーティング・システムについて提供されていない場合があります。たとえば、Ultra LightはLinux版では提供しておりません。製品構成および対応プラットフォームについては、各製品のマニュアルをご覧ください。

本プログラムのメディア・パッケージには、複数の異なるオペレーティング・システムに対応するSQL Anywhere Studioのサーバおよびクライアント・コンポーネントの複数のバージョンが含まれることがありますが、お客様はSQL Anywhere Studioのライセンス許諾を受けたSQL Anywhere Studioのサーバおよびクライアントのコンポーネントのそれぞれのコピーを（1）つのオペレーティング・システムにのみインストールすることができます。

第2条（各コンポーネントの個別使用）　前述のとおり、SQL Anywhere Developer Editionは、いくつかのコンポーネントソフトウェアからなる製品です。以下において特に認める場合を除き、本プログラムのいずれのコンポーネントも本プログラムのライセンスを取得している一または複数の特定の開発用シート（たとえばPC、（1）ユーザのワークステーション、またはリアルタイム・デバイスなど）上での使用に限定されます。本プログラムのコンポーネントをそれぞれ別のまたは追加のシートで使用することはできません。

第3条（開発者によるプログラムの使用）　このSQL Anywhere Developer Editionは社内評価、開発、および本プログラムと互換性のあるアプリケーションをテストすることのみを目的として、(1) コピーを単一のシートで使用することのみ許諾されています。通常の運用環境または Webサーバ上での使用等これ以外のいかなる目的の場合でも、SQL Anywhere Developer Editionまたはそのいずれのコンポーネントを配布または他の形で使用する場合は、事前に当社より配布ライセンスを取得する必要があります。本プログラムに含まれるクライアントシートの (1) ライセンスでは、1つのオペレーティング・システムを実行している (1) 台のマシンに Adaptive Server Anywhere DBMS をインストールし、(1) シートから Adaptive Server Anywhereサーバ・コンポーネントにアクセスすることを許諾しています。また接続された単一のクライアント・シートに以下のコンポーネントをインストールすることができます：(a) Adaptive Server Anywhereクライアント・ソフトウェア (b) jConnect、(c) Adaptive Server Anywhere 同期付パーソナルデータベース。上記のシートの同期は、Mobile LinkまたはSQL Remoteのどちらかを使用して実装することができます。また、Mobile Link同期サーバ (1) コピーをインストールしてSQL Anywhereデータベースと同期を行う目的で使用することができます。Ultra Light データベースとMobile Linkクライアント・ソフトウェアのコピーをそれぞれ1つ1台のクライアント・デバイス上で使用することができます。Adaptive Server AnywhereおよびMobile Linkと連携して使用する場合にのみ、Sybase Centralを (1) コピー使用できます。前述のプログラムは、Adaptive Server Anywhereサーバと Mobile Linkコンポーネントがインストールされているサーバ・マシンまたはAdaptive Server Anywhereサーバへのアクセスがライセンスされているクライアント・シートのいずれかに限られます。

Javaオプション：パッケージに含まれているAdaptive Server Anywhereのデータベース内Javaオプションは、個別ライセンスを必要に応じて別途購入しなければ

インストールまたは使用することはできません。

SQL Anywhere配布プラットフォーム：ユーザ・ツールの使用: Linuxなど特定のコンピュータ・プラットフォームでは、SQL Anywhereの一部のコア・コンポーネントのみが使用可能です。特定のユーザ・ツールを使用できないプラットフォームでライセンスを取得済みのSQL Anywhere Studioの各コピーについては、Windowsを実行するIntel（または互換性のある）マシン1台で、このようなユーザ・ツール（例: Sybase Central）を実行することが許諾されています。このIntelマシンでAdaptive Server Anywhereのデータベース・サーバを実行することは許可されていません。

セキュリティ・オプション：本プログラムに含まれているAdaptive Server Anywhere用128ビットCerticom暗号化は、個別のライセンスを別途購入することなくインストールしたり使用することはできません。必要なライセンスを購入した場合であっても、Certicom暗号化はSQL Anywhereと併用した場合に限り使用することができます。これらの暗号化プログラムやその関連マニュアルは暗号に関するものであるため、厳密な規制の対象となりますが、お客様が作成したアプリケーション、Certicom暗号化プログラム、または関連ドキュメントに必要な輸出許可または輸入許可の取得は、お客様が行うものとします。

サンプル・コード：サンプル・コード（コード例、サンプル・アプリケーション、マニュアルに記載されたその他素材など）として提供された、このプログラムの一部を構成するソース・コードのバージョンについては、必要に応じて、開発環境内で使用したり、修正することができます。そのようなソース・コードのバージョンを第三者に配布することはできません。当社およびそのサプライヤの著作権または特許権のいずれにおいても、明示的なライセンス許可を与えていません。

評価版ソフトウェア：このプログラムの評価版（上記に示す）については、開発環境内で評価する場合（アプリケーションのテストや開発以外の目的を含むが、これだけに限定されない）およびAdaptive Server Anywhereの使用と連携する場合にのみ、（1）台のマシンで各コピーを（1）つ使用できます。前述のプログラムの使用は、Adaptive Server Anywhereサーバ・コンポーネントがインストールされているサーバ・マシンまたはAdaptive Server Anywhereサーバへのアクセスがライセンスされているクライアント・シートのいずれかに限定されます。これ以外のいかなる目的（アプリケーションのテストと開発、一般の運用環境またはWebサーバでの使用を含むが、これだけに限定されない）の場合も、これらのプログラムのフルセットまたは一部を配布する前に、その目的のために配布ライセンスを 当社から取得する必要があります。

第4条　本契約は、メンテナンスその他のサービスや、本プログラムのアップデートまたは新しいバージョンの供給を受ける権利をお客様に与えるものではありません。メンテナンスおよびサポートの提供は,無償のものもあります。

第5条（期間）　本ライセンスはその終了時まで有効です。お客様は、プログラムおよび関連ドキュメントを破棄することによって、いつでも本ライセンスを終了させることができます。また、本ライセンスは、お客様が本契約条項の何れかに違反する行為（不作為を含みます）があった場合にも終了します。この場合、お客様はプログラムおよびドキュメントを破棄することに同意していただきます。本プログラムのライセンス契約に記載されているすべての保証の放棄および責任制限は、本プログラムおよびそのコンポーネントに適用され、本契約の終了後も存続します。

第6条（追加製品）　本プログラムには以下のバージョンのサード･パーティ製品が同梱されており（追加製品）、以降に示す諸条件がさらに適用されます。

Microsoft® Visual C++®、バージョン6.0、Runtimeファイル; Installer Components of Microsoft Data Access Components 2.7; Java Runtime Environment、Java 2 Standard Edition、J2SE™ バージョン1.4.1; Microsoft HTML Help Supportファイル、Palm Conduit Manager DLL; Merant ODBCドライバ; Microsoft XML Parser 4.0。

6.1　本プログラムとの併用

追加製品は、本プログラムと同時に用いることのみを目的に使用できます。

6.2　追加製品の配布可能ファイル

一部の追加製品には、配布可能ファイルとして再配布してもよいファイルが含まれていますがSQL Anywhere Developer Editionとの連携においてはその権利は許可しておりません。また、追加製品の所有者によってそのような権利が独自に認められる場合を除き、これらのファイルのいずれか、または追加製品を配布することはできません。

6.3　Java Platform InterfaceおよびJava Mark

上記の規定の他、お客様は、Java Platform Interface（JPI：“java”パッケージまたは“java”パッケージのサブパッケージに含まれているクラスとして識別）内でクラスを追加作成したり、JPIのクラスを結果的に追加または変更したりすることはできません。お客様は(i) Java環境の機能を拡張するAPIを作成し、(ii) そのような追加APIを実行する追加のソフトウェアを開発する目的でそのAPIを第三者のソフトウェア・ベンダーに提供する場合には、全ての開発者がそのようなAPIを無償で利用できるよう、正確な仕様を、速やかに、適切な形で広く公開しなければなりません。お客様はいかなる“java”クラスにも名前を付けることはできず、またSunが合

理的に要求する命名規則に従う必要があります。本ライセンスでは、Sunの名称、商標、ロゴ（をお客様が使用することをいっさい許諾しておらず、Javaの商標およびJava関連のすべての商標、ロゴ、Coffee Cupおよび Duke（Java Mark）などのアイコンがSunの所有物であることを了解するとともに、(i) Java Trademark Guidelines (http://java.sun.com/trademarks.html) を遵守し、(ii) Java Markに関して Sunが所有する権利を侵害または相反する行為をいっさい行わず、(iii) 任意の Java Markについてお客様が取得したいっさいの権利をSunへ譲渡することを含め、Sunのこれらの権利を保護するよう協力することに同意することになります。

6.4　追加製品のサポート

当社では、同梱したバージョンの追加製品を当社による保証は追加せずに現状のままお客様に提供しています。これらの追加製品の保証は、それらのサプライヤから提供されることもあります。当社では同梱した追加製品の本プログラムへの統合はサポートしますが、追加製品そのものに対するサポートや不具合の修正、または新規リリースへの対応を行う義務はいっさいありません。このようなサービスや製品の提供は、追加製品の各サプライヤから受けられることもあります。

6.5　本プログラムおよび追加製品に対して適用される当社のソフトウェア・ライセンス契約に定めるその他の免責条項および損害の限定に加え、すべての追加製品に対して次の規定が適用されます。いかなる場合も、当社およびその関連会社は、追加製品起因あるいは関連して発生した直接の損害に関して一切責任を負いません。

第7条（その他）　本プログラムを日本以外の国でダウンロードおよびインストールする場合、当社に変わりSybase Inc. およびその子会社（以下「サイベース」といいます）から使用許諾される場合があります。この場合、本契約書における「当社」は「サイベース」に読み替えられるものとします。

Index

記号

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Index

V

W

X

ア行

Index

タ行

Index

著者紹介 Profile

アイエニウェア・ソリューションズ株式会社
米国iAnywhere Solutions, Inc. 100%出資の子会社。2003年にサイベース株式会社の事業部より独立し、アイエニウェア・ソリューションズ株式会社として設立された。iAnywhere Solutions, Inc.は、組み込みおよびモバイル向けデータベース、モバイル・ミドルウェア、フロントライン管理ソリューションにおける世界的なリーディングプロバイダであり、そのSQL Anywhere技術とモバイル・エンタープライズ・ソリューションは15,000社を超える企業と1,000社以上のパートナー企業で利用されている。また、登録ユーザ数1,000万人を超える世界最大規模のモバイル・インターネット・サービスAvantGoを展開している。

森脇 大悟（もりわき だいご）

1974年生まれ。神奈川県川崎市出身。京都大学理学部物理学科卒業。同大学院修士課程中退後、有限会社グルージェント（現・株式会社グルージェント）入社。SIとして金融や物流システムを手がける。2003年アイエニウェア・ソリューションズ株式会社入社。エンジニアとして製品の導入支援やコンサルティング業務に携わり、Mobile Linkを使った大規模案件を得意とする。個人としてはRubyやDebianが好きで、それらのコミュニティにも参加している。

装幀・アートディレクション：犬飼健二
イラスト：さか井美ゆき
デザイン：浦和いずみ＋千田奈津子（犬飼デザインサイト）
編集協力・DTP：有限会社 風工舎

使える！ SQL Anywhere®（エスキューエル エニウェア）

2005年8月29日　初版第1刷発行

著者　アイエニウェア・ソリューションズ株式会社
　森脇 大悟（もりわき・だいご）
発行人　速水 浩二
発行所　株式会社翔泳社（http://www.seshop.com）
印刷・製本　株式会社 廣済堂

ISBN4-7981-0754-9　Printed in Japan