SYBASE®

システム管理ガイド：第 2 巻

Sybase IQ

15.1

ドキュメント ID：DC01145-01-1510-01
改訂：2009 年 7 月

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

はじめに

このマニュアルの内容

Sybase® IQ は、データ・ウェアハウスやデータ・マート用に特化された高性能の意思決定支援サーバです。この『システム管理ガイド 第 2 巻』では Sybase IQ でのプログラミングで必要な概念と手順を説明しています。

対象読者

このマニュアルは Sybase IQ データベースのデータにアクセスするアプリケーションの開発者を対象にしています。リレーショナル・データベース・システムの基礎知識と、Sybase IQ のユーザ・レベルの基本的な経験があることを前提にしています。このマニュアルは、他のマニュアルと併用するように構成されています。

関連 Sybase IQ マニュアル

Sybase IQ 15.1 マニュアル・セットには、次のマニュアルがあります。

- 『リリース・ノート』では、製品およびマニュアルに加えられた最新の変更内容について説明しています。
- 『インストールおよび設定ガイド』では、プラットフォーム固有のインストール手順、新バージョンへのマイグレート、特定のプラットフォームでの Sybase IQ の設定について説明しています。
- 『Sybase IQ による高度なセキュリティ』では、Sybase IQ データ・レポジトリ内でのユーザによるカラムの暗号化の使用について説明しています。このオプションの製品をインストールするには、別のライセンスが必要です。
- 『エラー・メッセージ』では、Sybase エラー・コード、SQLCode、および SQLState によって参照される Sybase IQ エラー・メッセージ、および SQL プリプロセッサのエラーと警告を示します。
- 『IMSL 数値関数ライブラリ・ユーザ・ガイド : 第 2/2 巻 C 統計ライブラリ』には、IMSL C 統計ライブラリの時系列 C 関数の簡潔な説明が記載されています。このマニュアルは、RAP — The Trading Edition™ Enterprise のユーザのみが入手できます。
- 『Sybase IQ の概要』は、Sybase IQ や Sybase Central™ データベース管理ツールの操作に慣れていない場合、参照してください。実際に操作の練習ができます。

- 『Sybase IQ によるラージ・オブジェクト管理』では、Sybase IQ データ・レポジトリ内での BLOB (バイナリ・ラージ・オブジェクト) および CLOB (キャラクタ・ラージ・オブジェクト) の格納と取得について説明しています。このオプションの製品をインストールするには、別のライセンスが必要です。
- 『Sybase IQ 15.0 の新機能』では、バージョン 15.0 の新機能と動作変更について説明しています。
- 『新機能の概要 Sybase IQ 15.1』では、現在のバージョンの新機能と動作変更の概要について説明しています。
- 『パフォーマンス&チューニング・ガイド』では、巨大なデータベースのクエリ最適化、設計、チューニングについて説明しています。
- 『クイック・スタート』では、Sybase IQ ソフトウェアのインストールを検証するために Sybase IQ に付属しているデモ・データベースの構築とクエリを行う手順が記載されています。デモ・データベースのマルチプレックスへの変換についても説明しています。
- 『リファレンス・マニュアル』には、Sybase IQ の次の 2 つのリファレンス・ガイドがあります。
  - 『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』では、Sybase IQ でサポートされる SQL、ストアド・プロシージャ、データ型、およびシステム・テーブルについて説明しています。
  - 『リファレンス：文とオプション』は、Sybase IQ でサポートされる SQL 文およびオプションについて説明します。
- 『システム管理ガイド』は、2 巻構成です。
  - 『システム管理ガイド 第 1 巻』では、起動、接続、データベース作成、自動入力とインデックス作成、バージョン設定、照合、システムのバックアップとリカバリ、トラブルシューティング、およびデータベースの修復について説明しています。
  - 『システム管理ガイド 第 2 巻』では、プロシージャとバッチの作成および実行、OLAP でのプログラミング、リモート・データへのアクセス、Open Server としての IQ の設定、スケジューリングとイベント処理、XML でのプログラミング、およびデバッグについて説明しています。
- 『ユーザ定義関数ガイド』では、ユーザ定義関数、パラメータ、および使用のシナリオについて説明しています。

- 『Sybase IQ マルチプレックスの使用』では、複数のノードにまたがって発生する大きなクエリの負荷を管理するために設計された、マルチプレックス機能の使用方法について説明しています。
- 『ユーティリティ・ガイド』では、Sybase IQ ユーティリティ・プログラムのリファレンス項目 (使用可能な構文、パラメータ、オプション) について説明しています。

関連 SQL Anywhere マニュアル

Sybase IQ は SQL Anywhere® パッケージのコンポーネントである SQL Anywhere Server と多くのコンポーネントを共有しているため、Sybase IQ は SQL Anywhere Server と同じ機能を数多くサポートしています。IQ のマニュアル・セットは、SQL Anywhere のマニュアルの該当する箇所を参照しています。

SQL Anywhere には、次のマニュアルがあります。

- 『SQL Anywhere サーバ — データベース管理』では、SQL Anywhere データベースの実行、管理、構成方法について説明しています。このマニュアルでは、データベース接続、データベース・サーバ、データベース・ファイル、バックアップ手順、セキュリティ、高可用性、Replication Server® での複写、管理ユーティリティおよびオプションについても説明します。
- 『SQL Anywhere サーバ — プログラミング』では、C、C++、Java、PHP、Perl、Python、および Visual Basic や Visual C# などの .NET プログラミング言語を使用したデータベース・アプリケーションを構築し配備する方法について説明しています。このマニュアルでは、ADO.NET や ODBC などの各種プログラミング・インタフェースについても説明します。
- 『SQL Anywhere サーバ — SQL リファレンス』では、システム・プロシージャおよびカタログ (システム・テーブルおよびビュー) に関する参照情報を示します。SQL 言語 (検索条件、構文、データ型、および関数) の SQL Anywhere の実装の説明についても説明します。
- 『SQL Anywhere サーバ — SQL の使用法』では、データベースの設計／作成方法、データのインポート／エクスポート／変更方法、データの検索方法、およびストアド・プロシージャとトリガの作成方法について説明します。

Product Manuals (http://sybooks.sybase.com) および DocCommentXchange (http://dcx.sybase.com/dcx_home.php) の SQL Anywhere 11.0.1 コレクションの SQL Anywhere マニュアルも参照してください。

関連 SySAM マニュアル

Sybase ソフトウェア資産管理 (SySAM) には、次のマニュアルがあります。

- 『Sybase ソフトウェア資産管理 (SySAM) 2』では資産管理の概念を紹介し、SySAM 2 ライセンスの設定および管理方法について説明します。
- 『SySAM 2 クイック・スタート・ガイド』は、SySAM 対応の Sybase 製品を実行する方法について説明します。
- 『FLEXnet ライセンス・エンド・ユーザ・ガイド』では、管理者およびエンド・ユーザ向けに FLEXnet ライセンスについて説明し、Sybase から販売される標準的な FLEXnet ライセンス配布キットに含まれているツールの使用方法について説明しています。

その他の情報

Sybase Getting Started CD、SyBooks™ CD、Sybase Product Manuals Web サイトを利用すると、製品について詳しく知ることができます。

- Getting Started CD には、PDF 形式のリリース・ノートとインストール・ガイド、SyBooks CD に含まれていないその他のマニュアルや更新情報が収録されています。この CD は製品のソフトウェアに同梱されています。Getting Started CD に収録されているマニュアルを参照または印刷するには、Adobe Acrobat Reader が必要です (CD 内のリンクを使用して Adobe の Web サイトから無料でダウンロードできます)。
- SyBooks CD には製品マニュアルが収録されています。この CD は製品のソフトウェアに同梱されています。Eclipse ベースの SyBooks ブラウザでは、使いやすい HTML 形式のマニュアルにアクセスできます。

  一部のマニュアルは PDF 形式で提供されています。それらのマニュアルは SyBooks CD の PDF ディレクトリに収録されています。PDF ファイルを開いたり印刷したりするには、Adobe Acrobat Reader が必要です。

  SyBooks をインストールして起動するまでの手順については、Getting Started CD の『SyBooks インストール・ガイド』、または SyBooks CD の *README.txt* ファイルを参照してください。

- Sybase Product Manuals Web サイトは、SyBooks CD のオンライン版であり、標準の Web ブラウザを使ってアクセスできます。また、製品マニュアルのほか、EBFs/Maintenance、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

  Sybase Product Manuals Web サイトにアクセスするには、Product Manuals (http://www.sybase.com/support/manuals/) にアクセスしてください。

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

❖ **製品動作確認の最新情報にアクセスする**

1 Web ブラウザで Technical Documents (http://certification.sybase.com/ucr/search.do) を指定します。

2 [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。

3 [検索] をクリックして、入手状況と動作確認レポートを表示します。

❖ **コンポーネント動作確認の最新情報にアクセスする**

1 Web ブラウザで Availability and Certification Reports (http://certification.sybase.com/) を指定します。

2 [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。

3 [Search] をクリックして、入手状況と動作確認レポートを表示します。

❖ **Sybase Web サイト (サポート・ページを含む) の自分専用のビューを作成する**

MySybase プロファイルを設定します。MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用にカスタマイズできます。

1 Web ブラウザで Technical Documents (http://www.sybase.com/support/techdocs/) を指定します。

2 [MySybase] をクリックし、MySybase プロファイルを作成します。

Sybase EBF とソフトウェア・メンテナンス

❖ **EBF とソフトウェア・メンテナンスの最新情報にアクセスする**

1. Web ブラウザで Sybase Support Page (http://www.sybase.com/support) を指定します。
2. [EBFs/Maintenance] を選択します。ユーザ名とパスワードの入力が求められたら、MySybase のユーザ名とパスワードを入力します。
3. 製品を選択します。
4. 時間枠を指定して [Go] をクリックします。EBF/Maintenance リリースの一覧が表示されます。

   鍵のアイコンは、自分が Technical Support Contact として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。未登録ではあるが、Sybase 担当者またはサポート・コンタクトから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」役割を MySybase プロファイルに追加します。
5. EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

SQL 構文の表記規則

このマニュアルでは、構文の説明に次の表記規則を使用します。

- **キーワード** SQL キーワードは大文字で示します。ただし、SQL キーワードは大文字と小文字の区別がないので、入力するときはどちらで入力してもかまいません。たとえば、SELECT は Select でも select でも同じです。
- **プレースホルダ** 適切な識別子または式で置き換えられる項目は、斜体で表記します。
- **継続** 省略記号 (...) で始まる行は、前の行から文が続いていることを表します。
- **繰り返し項目** 繰り返し項目のリストは、リストの要素の後ろに省略記号 (ピリオド3つ ...) を付けて表します。1 つまたは複数の要素を指定できます。複数の要素を指定する場合は、各要素間はカンマで区切る必要があります。

- **オプション指定部分**　文のオプション指定部分は、角カッコで囲みます。次に例を示します。

  ```
  RELEASE SAVEPOINT [ savepoint-name ]
  ```

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

- **オプション**　項目リストから 1 つだけ選択しなければならない場合、また何も選択する必要のない場合は、項目間を縦線で区切り、リスト全体を角カッコで囲みます。次に例を示します。

  ```
  [ ASC | DESC ]
  ```

  この例では、ASC、DESC のどちらか 1 つを選択するか、どちらも選択しないことができます。角カッコは入力しないでください。

- **選択肢**　オプションの中の 1 つを必ず選択しなければならない場合は、選択肢を中カッコ { } で囲みます。次に例を示します。

  ```
  QUOTES { ON | OFF }
  ```

  大カッコは、ON か OFF のいずれかを含めなければいけないことを示します。大カッコは入力しないでください。

書体の表記規則

表 1 に、このマニュアルで使用している書体の表記規則を示します。

表 1: 書体の表記規則

項目	説明
`Code`	SQL およびプログラム・コードは等幅 (固定幅) 文字フォントで表記します。
`User entry`	ユーザが入力するテキストには等幅 (固定幅) 文字フォントが使用されます。
「強調」	強調する言葉は「 」で囲みます。
file names	ファイル名は斜体で表記します。
database objects	テーブル、プロシージャなどのデータベース・オブジェクトの名前は、印刷物では太字の sans serif フォントで、オンラインでは斜体で表記します。

アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

Sybase IQ 15.1 の HTML マニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

アクセシビリティ・ツールの設定

アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPER CASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定することをおすすめします。スクリーン・リーダの使用方法については、使用しているツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility (http://www.sybase.com/accessibility) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報のリンクもあります。

Sybase IQ の第 508 条準拠の声明については、Sybase Accessibility (http://www.sybase.com/products/accessibility) を参照してください。

デモ・データベース

Sybase IQ にはデモ・データベース (*iqdemo.db*) を作成するスクリプトが用意されています。このマニュアルで紹介している多くのクエリおよびコード例は、このデモ・データベースをデータ・ソースに使用しています。

デモ・データベースは、小規模会社の内部情報 (従業員、部署、財務データ) に加えて、製品と販売情報 (注文、顧客、担当者) で構成されています。

デモ・データベースの詳細については、使用しているプラットフォームの『Sybase IQ インストール・ガイド』を参照するか、システム管理者に相談してください。

不明な点があるときは

サポート契約を購入済みの Sybase 製品のインストールには、定められた 1 人以上のユーザに対して、Sybase 製品の保守契約を結んでいるサポート・センタを利用する権利が付属します。マニュアルやオンライン・ヘルプで解決できない問題がある場合は、この担当者を通して最寄りの Sybase のサポート・センタまでご連絡ください。

第 1 章

プロシージャとバッチの使用

この章について

この章では、Sybase IQ で使用するプロシージャとバッチの作成方法について説明します。

プロシージャは、すべてのアプリケーションで使えるように、手続き型 SQL 文をデータベースに格納します。これによりデータベースのセキュリティ、効率、標準化を高めることができます。ユーザ定義関数は、クエリやその他の SQL 文で使うための結果を返すプロシージャの一種です。バッチは、データベース・サーバにグループとして送られる SQL 文のセットです。制御文などのプロシージャで使用できる機能の多くは、バッチ内でも使用できます。

さまざまな用途で、サーバ側の JDBC は、SQL ストアド・プロシージャよりも柔軟にデータベースにロジックを構築します。SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — プログラミング > SQL Anywhere データ・アクセス API > SQL Anywhere JDBC ドライバ』の「JDBC の概要」を参照してください。

内容

プロシージャの概要

プロシージャは、手続き型 SQL 文をすべてのアプリケーションで使えるようにデータベースに格納します。プロシージャでは、制御文を使用して、SQL 文の繰り返し (LOOP 文) や条件付き実行 (IF 文と CASE 文) ができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「プロシージャとトリガの概要」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

プロシージャの利点

プロシージャの定義はデータベースに置かれており、各データベース・アプリケーションとは区別されています。この区別には多くの利点があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「プロシージャとトリガの利点」を参照してください。

プロシージャ入門

プロシージャを使用するには、次の方法について理解する必要があります。

- プロシージャの作成
- データベース・アプリケーションからの呼び出し
- プロシージャの削除
- プロシージャ使用可能パーミッションの制御

この項では、これらの項目に加え、通常とは異なるプロシージャの使用方法についても説明します。

ストアド・プロシージャを扱うときに役立つシステム・ストアド・プロシージャとして、sp_iqprocedure と sp_iqprocparm の 2 つがあります。sp_iqprocedure ストアド・プロシージャは、データベース内のシステムおよびユーザ定義プロシージャに関する情報を表示します。sp_iqprocedure ストアド・プロシージャは、次のカラムのようなストアド・プロシージャのパラメータに関する情報を表示します。

- proc_name
- proc_owner
- parm_name
- parm_type
- parm_mode
- domain_name
- width、scale
- default

プロシージャの作成

プロシージャは CREATE PROCEDURE 文を使用して作成します。プロシージャを作成するには、RESOURCE 権限が必要です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「プロシージャの作成」を参照してください。

Sybase IQ の例

注意 次の例では、Sybase IQ デモ・データベース、iqdemo.db を使用します。インストールに関する情報については、「デモ・データベース」(xvi ページ) を参照してください。

```
create procedure new_dept(IN id INT, IN name CHAR(35),
IN head_id INT)
BEGIN
INSERT
INTO GROUPO.departments(DepartmentID, DepartmentName,
```

```
    DepartmentHeadID)
    values (id, name, head_id);
    END
```

注意 IQ でリモート・プロシージャを作成するには、CREATE PROCEDURE の AT ***location-string*** SQL 構文を使用して、プロキシ・ストアド・プロシージャを作成します。この機能は、現時点では Windows と Sun Solaris でのみ動作確認されています。詳細については、「リモート・プロシージャ・コール (RPC) の使用」(115 ページ) を参照してください。Sybase Central の [リモート・プロシージャの追加] ウィザードは、リモート サーバでのみ使用できます。

プロシージャの変更

Sybase Central または Interactive SQL のいずれかを使用して、既存のプロシージャを修正できます。それには、DBA 権限を持っているか、またはプロシージャの所有者でなくてはなりません。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「プロシージャの変更」を参照してください。

データベース・オブジェクト・プロパティの変更の詳細については、『Sybase IQ の概要』の「第 4 章 データベースの管理」を参照してください。

プロシージャに対するパーミッションの付与または取り消しについては、『システム管理ガイド 第 1 巻』の「第 8 章 ユーザ ID とパーミッションの管理」の「プロシージャに対するパーミッションの付与」と「ユーザ・パーミッションの取り消し」を参照してください。

『リファレンス：文とオプション』の「ALTER PROCEDURE 文」と「CREATE PROCEDURE 文」も参照してください。

プロシージャの呼び出し

プロシージャの呼び出しには CALL 文を使用します。プロシージャは、アプリケーション・プログラムから呼び出すことも、他のプロシージャから呼び出すこともできます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「プロシージャの呼び出し」を参照してください。

また、次も参照してください。

- 『リファレンス: 文とオプション』の「第 1 章 SQL 文」の「CALL 文」
- 「プロシージャを実行するためのパーミッション」(5 ページ)

Sybase Central でのプロシージャのコピー

プロシージャのコードは、あるデータベースから別の接続されたデータベースへコピーできます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「Sybase Central におけるプロシージャのコピー」を参照してください。

プロシージャの削除

いったん作成したプロシージャは、誰かが明示的に削除するまではデータベースに残っています。プロシージャの所有者か DBA 権限を持つユーザだけがプロシージャをデータベースから削除できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「プロシージャの削除」を参照してください。

プロシージャを実行するためのパーミッション

プロシージャの所有者はそれを作成したユーザです。所有者はパーミッションなしでそのプロシージャを実行できます。プロシージャを実行するパーミッションを他のユーザに付与するには、GRANT EXECUTE コマンドを使用します。

たとえば、プロシージャ new_dept の所有者が another_user に new_dept の実行パーミッションを付与するには、次の文を使用します。

```
GRANT EXECUTE ON new_dept TO another_user
```

パーミッションを取り消す文は、次のようになります。

```
REVOKE EXECUTE ON new_dept FROM another_user
```

詳細については、『システム管理ガイド 第 1 巻』の「プロシージャに対するパーミッションの付与」を参照してください。

プロシージャの結果をパラメータとして返す

プロシージャは次のいずれかの方法で、呼び出し元の環境に結果を返します。

- 個別の値を OUT パラメータまたは INOUT パラメータとして返す。
- 結果セットを返す。
- RETURN 文を使って結果を 1 つ返す。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「プロシージャの結果をパラメータとして返す」を参照してください。

Sybase IQ の例

注意 次の例では、Sybase IQ デモ・データベース、iqdemo.db を使用します。インストールに関する情報については、「デモ・データベース」(xvi ページ) を参照してください。

```
CREATE PROCEDURE SalaryList (IN department_id INT)
RESULT ( "Employee ID" INT, "Salary" NUMERIC(20,3) )
BEGIN
SELECT EmployeeID, Salary
FROM Employees
WHERE Employees.DepartmentID = department_id;
END
```

プロシージャの結果を結果セットとして返す

プロシージャは、個別のパラメータとして呼び出しを行った環境に結果を返すだけでなく、結果セットとして情報を返すこともできます。通常、結果セットになるのはクエリの結果です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「プロシージャの結果を結果セットとして返す」を参照してください。

テンポラリ・テーブルの作成とテーブルからの選択

ストアド・プロシージャ内でテンポラリ・テーブルを動的に作成した後、そのテーブルに対する選択を実行する場合は、EXECUTE IMMEDIATE WITH RESULT SET ON 構文を使用して、“カラムが見つかりません。”というエラーが起きないようにします。

次に例を示します。

```
CREATE PROCEDURE p1 (IN @t varchar(30))
   BEGIN
      EXECUTE IMMEDIATE
      'SELECT * INTO #resultSet FROM ' || @t;
      EXECUTE IMMEDIATE WITH RESULT SET ON
      'SELECT * FROM #resultSet';

   END
```

ユーザ定義関数入門

ユーザ定義関数は、呼び出しを行った環境に単一の値を返すプロシージャのクラスです。ここでは、ユーザ定義関数の作成、使用、削除について説明します。

ユーザ定義関数の作成

ユーザ定義関数を作成するには、CREATE FUNCTION 文を使用します。ただし、RESOURCE 権限を持っていなくてはなりません。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「ユーザ定義関数の作成」を参照してください。

パフォーマンスの考慮事項や、SQL Anywhere と IQ の違いなど、CREATE FUNCTION の構文の詳細については、『リファレンス：文とオプション』の「第 1 章 SQL 文」を参照してください。

ユーザ定義関数の呼び出し

ユーザ定義関数は、パーミッションがあれば、集合関数以外の組み込み関数を使用できるどの場所でも使用できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「ユーザ定義関数の呼び出し」を参照してください。

Sybase IQ の例

注意 次の例では、Sybase IQ デモ・データベース、iqdemo.db を使用します。インストールに関する情報については、「デモ・データベース」(xvi ページ) を参照してください。

```
SELECT fullname (GivenName, SurName)
FROM Employees;
```

fullname (GivenName, SurName)

Fran Whitney

Matthew Cobb

Philip Chin

...

ユーザ定義関数の削除

ユーザ定義関数が作成されると、明示的に削除されるまでデータベースに存在します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャの概要』の「ユーザ定義関数の削除」を参照してください。

ユーザ定義関数を実行するためのパーミッション

ユーザ定義関数の所有者はそれを作成したユーザです。所有者はパーミッションなしでそれを実行できます。ユーザ定義関数の所有者は、GRANT EXECUTE コマンドを使用して、他のユーザにパーミッションを付与できます。

たとえば、関数 fullname の作成者は、次の文を使用して another_user に fullname を使用するパーミッションを付与できます。

```
GRANT EXECUTE ON fullname TO another_user
```

パーミッションを取り消す文は、次のようになります。

```
REVOKE EXECUTE ON fullname FROM another_user
```

『システム管理ガイド 第 1 巻』の「第 8 章 ユーザ ID とパーミッションの管理」の「プロシージャに対するパーミッションの付与」を参照してください。

バッチ入門

簡単なバッチは、セミコロン (;) で区切られた SQL 文のセットから構成されます。たとえば、次の一連の文はバッチを構成します。このバッチは、Eastern Sales という部署を作成し、マサチューセッツ (MA) のすべての営業担当者をこの部署に移動します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「バッチの概要」を参照してください。

Sybase IQ の例

注意 次の例では、Sybase IQ デモ・データベース、iqdemo.db を使用します。インストールに関する情報については、「デモ・データベース」(xvi ページ) を参照してください。

```
INSERT
INTO Departments ( DepartmentID, DepartmentName )
VALUES ( 220, 'Eastern Sales' ) ;
UPDATE Employees
SET DepartmentID = 220
WHERE DepartmentID = 200
AND state = 'GA' ;
COMMIT ;
```

制御文

プロシージャの本文またはバッチの中には、論理フローや意思決定のための制御文が多く含まれています。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「制御文」を参照してください。

各文の詳細については、『リファレンス：文とオプション』の「第 1 章 SQL 文」の各項目を参照してください。

複合文の使用

複合文はネストが可能です。また、他の制御文と組み合わせて、プロシージャ内またはバッチ内の実行フローを定義できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > 制御文』の「複合文の使用」を参照してください。

複合文での宣言

複合文中のローカル宣言文は、キーワード BEGIN のすぐ後に続きます。このローカル宣言は複合文中だけに存在します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > 制御文』の「複合文での宣言」を参照してください。

アトミックな複合文

「**アトミック**」な文とは、完全に実行されたか、まったく実行されなかったかのいずれかの文です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > 制御文』の「アトミックな複合文」を参照してください。

プロシージャの構造

プロシージャの本体は、「複合文の使用」(10 ページ) で説明したように、複合文から構成されています。複合文は、一連の SQL 文を囲む BEGIN と END で構成されています。各文はセミコロンで区切られています。

プロシージャで使用可能な SQL 文については、「プロシージャで使用可能な SQL 文」(11 ページ) を参照してください。

プロシージャで使用可能な SQL 文

プロシージャでは、次に示すように、ほぼすべての SQL 文を使用できます。

- SELECT、UPDATE、DELETE、INSERT、SET VARIABLE
- 他のプロシージャを実行する CALL 文
- 制御文 (「制御文」(10 ページ) を参照)
- カーソル文 (「プロシージャでのカーソルの使用」(14 ページ) を参照)
- 例外処理文 (「プロシージャでの例外ハンドラの使用」(17 ページ) を参照)
- EXECUTE IMMEDIATE 文

一部の SQL 文はプロシージャ内で使用できません。たとえば次のものです。

- CONNECT 文
- DISCONNECT 文

COMMIT、ROLLBACK、SAVEPOINT 文は、プロシージャ内で使用できますが、一部制約があります (「プロシージャでのトランザクションとセーブポイント」(18 ページ) を参照)。

詳細については、『リファレンス : 文とオプション』の「第 1 章 SQL 文」で、各 SQL 文の「**使用法**」の項を参照してください。

プロシージャ・パラメータの宣言

プロシージャのパラメータは、CREATE PROCEDURE 文でリストとして記述します。パラメータ名は、カラム名など他のデータベース識別子に関するルールに従って付けてください。パラメータには有効なデータ型を指定する必要があります (『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 3 章 SQL データ型」を参照)。また、IN、OUT、または INOUT のいずれかのキーワードを先頭に指定する必要があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガの構造』の「プロシージャ・パラメータの宣言」を参照してください。

パラメータをプロシージャに渡す

ストアド・プロシージャ・パラメータのデフォルト値は、CALL 文の 2 通りの形式のどちらでも使用できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガの構造』の「パラメータをプロシージャに渡す」を参照してください。

パラメータを関数に渡す

ユーザ定義関数は、CALL 文で呼び出すのではなく、組み込み関数と同じように使用します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガの構造』の「パラメータを関数に渡す」を参照してください。

プロシージャから返される結果

プロシージャは結果を 1 つまたは複数のローとして返します。単一のローのデータからなる結果の場合は、プロシージャへの引数で返すことができます。複数のローのデータからなる結果の場合は、結果セットで返します。また、プロシージャは RETURN 文で 1 つの値を返すこともできます。

プロシージャから結果を返す簡単な例については、「プロシージャ入門」(2 ページ) を参照してください。詳細については、次の項を参照してください。

RETURN 文を使って値を返す

RETURN 文は、呼び出し元の環境に単一の整数値を返し、プロシージャを直ちに終了します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャから返される結果』の「RETURN 文を使って値を返す」を参照してください。

結果をプロシージャのパラメータとして返す

プロシージャは、プロシージャのパラメータで呼び出しを行った環境に結果を返すことができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャから返される結果』の「結果をプロシージャのパラメータとして返す」を参照してください。

プロシージャから結果セットを返す

結果セットを使うと、プロシージャから複数ローの結果を呼び出し元の環境に返すことができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャから返される結果』の「プロシージャから結果セットを返す」を参照してください。

プロシージャから複数の結果セットを返す

プロシージャから呼び出元の環境に、複数の結果セットを返すことができます。

複数の結果セットを返すための方法は、dbisql と dbisqlc では異なります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャから返される結果』の「プロシージャから複数の結果セットを返す」を参照してください。

プロシージャから変数結果セットを返す

プロシージャで RESULT 句を省略することもできます。RESULT 句を省略すると、実行方法に応じてさまざまなカラム数やカラム型を使った異なる結果セットを返すプロシージャを記述できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャから返される結果』の「プロシージャから変数結果セットを返す」を参照してください。

DESCRIBE 文の詳細については、『リファレンス：文とオプション』の「第 1 章 SQL 文」を参照してください。

プロシージャでのカーソルの使用

カーソルは、結果セットに複数のローがあるクエリやストアド・プロシージャからローを 1 つずつ取り出します。「カーソル」は、クエリまたはプロシージャに対するハンドルまたは識別子で、結果セットの中の現在の位置を示します。

カーソル管理の概要

カーソル管理はプログラミング言語のファイル管理に似ています。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガでのカーソルの使用』の「カーソル管理の概要」を参照してください。

sp_iqcursorinfo ストアド・プロシージャは、サーバ上で現在開いているカーソルに関する情報を表示します。詳細については、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 7 章 システム・プロシージャ」の「sp_iqcursorinfo プロシージャ」を参照してください。

カーソル位置

カーソルはさまざまな位置に置くことができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — プログラミング > SQL Anywhere でのプログラミングの概要 > アプリケーションでの SQL の使用 > カーソルを使用した操作』の「カーソル位置」を参照してください。

注意 Sybase IQ は、FIRST、LAST、ABSOLUTE の各オプションを結果セットの先頭から開始するものとして扱います。負数のロー・カウントの RELATIVE は、現在の位置から開始するものとして扱います。

プロシージャの SELECT 文でのカーソルの使用

次に、SELECT 文でカーソルを使用するプロシージャの例を示します。これは、「プロシージャから結果セットを返す」で説明した ListCustomerValue プロシージャで使用したのと同じクエリを基にしており、ストアド・プロシージャ言語のいくつかの機能を使用しています。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガでのカーソルの使用』の「プロシージャの SELECT 文でのカーソルの使用」を参照してください。

プロシージャでのエラーと警告

アプリケーション・プログラムでは、SQL 文を実行した後、「**リターン・コード**」(ステータス・コード) をチェックできます。リターン・コードは文が正しく実行されたかどうかを表示して、エラーの場合はその理由を提示します。

『SQL Anywhere サーバ — SQL の使用法』の「プロシージャとトリガでのエラーと警告」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

プロシージャでのデフォルトのエラー処理

この項では、プロシージャ内でエラー処理を指定しなかった場合に、プロシージャの実行中に発生したエラーを Sybase IQ が処理する方法について説明します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガでのエラーと警告』の「プロシージャとトリガでのデフォルトのエラー処理」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

ON EXCEPTION RESUME を使ったエラー処理

CREATE PROCEDURE 文に ON EXCEPTION RESUME 句が含まれていた場合は、エラーが発生すると、その次の文が検査されます。文がエラーを処理する場合は、エラーが起きても呼び出しを行った環境に制御が戻りません。エラーを起こした文の次の文から実行を再開します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガでのカーソルの使用』の「ON EXCEPTION RESUME を使ったエラー処理」を参照してください。

プロシージャでのデフォルトの警告処理

エラーと警告は処理が異なります。デフォルトのエラー処理では、エラーが発生した場合、SQLSTATE 変数と SQLCODE 変数に値が設定され、呼び出し元の環境に制御が戻されます。一方、デフォルトの警告処理では、SQLSTATE と SQLCODE に値が設定され、プロシージャの実行が継続されます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガでのエラーと警告』の「プロシージャとトリガでのデフォルトのエラー処理」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

プロシージャでの例外ハンドラの使用

エラーのタイプによっては、呼び出しを行った環境へ戻すよりも、プロシージャの内部で傍受して処理できます。それには「**例外ハンドラ**」を使用します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガでのエラーと警告』の「プロシージャとトリガでの例外ハンドラの使用」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

ネストされた複合文と例外処理

ネストされた複合文を使用すると、エラーの後に実行する文と実行しない文を制御しやすくなります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用 > プロシージャとトリガでのエラーと警告』の「ネストされた複合文と例外処理」を参照してください。

プロシージャでの EXECUTE IMMEDIATE 文の使用

EXECUTE IMMEDIATE 文を使うと、引用符で囲んだ文字列と変数を使ってプロシージャ内に文を組み立てることができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「プロシージャでの EXECUTE IMMEDIATE 文の使用」を参照してください。

プロシージャでのトランザクションとセーブポイント

プロシージャまたはトリガ内の SQL 文は、現在のトランザクションの一部です。1 つのトランザクション中で複数のプロシージャを呼び出したり、1 つのプロシージャ中に複数のトランザクションを保持したりできます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「プロシージャとトリガでのトランザクションとセーブポイント」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

詳細については、以下を参照してください。

- 『システム管理ガイド 第 1 巻』の「第 10 章 トランザクションとバージョン管理」の「トランザクション内のセーブポイント」
- 『システム管理ガイド 第 1 巻』の「第 10 章 トランザクションとバージョン管理」

プロシージャを作成するときのヒント

この項では、プロシージャを作成するためのヒントをいくつか説明します。ここでは、次の項目を説明します。

- コマンド・デリミタを変更する必要があるかどうかチェックする
- プロシージャの中で文を区切る
- プロシージャ内のテーブルに完全修飾名を使用する
- プロシージャの中で日付と時刻を指定する

  日付と時刻の詳細については、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 3 章 SQL データ型」の「日付と時刻のデータ型」を参照してください。
- プロシージャの入力引数が正しく渡されたことを検証する

  MESSAGE 文の出力先の特定に関する詳細については、『リファレンス：文とオプション』の「第 1 章 SQL 文」の「MESSAGE 文」を参照してください。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「プロシージャを作成するときのヒント」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

プロシージャ、関数、ビューの内容の隠蔽

場合によっては、アプリケーションとデータベースを配布するときに、プロシージャ、関数、トリガ、ビューの中身のロジックを隠す方がよいかもしれません。ALTER PROCEDURE 文、ALTER FUNCTION 文、ALTER VIEW 文では、セキュリティ強化策の一環として、SET HIDDEN 句を使用してこれらのオブジェクトの内容を隠蔽できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「プロシージャ、関数、ビューの内容を隠す」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

詳細については、『リファレンス：文とオプション』の「ALTER FUNCTION 文」、「ALTER PROCEDURE 文」、「ALTER VIEW 文」を参照してください。

バッチで使用できる文

バッチでは大部分の SQL 文を使用できますが、一部例外があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「プロシージャ、トリガ、イベント、バッチで使用できる文」を参照してください。

注意 Sybase IQ ではトリガをサポートしていません。SQL Anywhere マニュアルのトリガについての情報は無視してかまいません。

バッチでの SELECT 文の使用

バッチには 1 つまたは複数の SELECT 文を含めることができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、トリガ、バッチの使用』の「バッチで SELECT 文を使用する」を参照してください。

Sybase IQ の例

注意 次の例では、Sybase IQ デモ・データベース、iqdemo.db を使用します。インストールに関する情報については、「デモ・データベース」(xvi ページ) を参照してください。

```
EIF EXISTS(SELECT *
FROM isystab
WHERE table_name='Employees' )
THEN
SELECT Surname AS LastName,
emp_fname AS FirstName
FROM Employees;
SELECT Surname, GivenName
FROM Customers;
SELECT Surname, GivenName
FROM Contacts;
END IF
```

IQ UTILITIES を使用した独自のストアド・プロシージャの作成

Sybase IQ に用意されているシステム・ストアド・プロシージャは、この章で説明した手法を用いて、SQL で実装されています。これらのプロシージャのいくつかから派生した独自のプロシージャを作成できます。それには次のような方法があります。

- システム・ストアド・プロシージャを呼び出すプロシージャを作成します。
- システム・ストアド・プロシージャとは別個に、同様の機能を実行するプロシージャを作成します。

- システム・ストアド・プロシージャと同じ構造を使用し、かつ独自の機能を追加したプロシージャを作成します。たとえば、フロントエンド・ツールやブラウザで、テキストではなくグラフィカル形式でプロシージャの結果を表示するなどです。

2 つ目や 3 つ目の方法を使用する場合は、IQ UTILITIES 文と、その使用方法の厳密な要件を理解する必要があります。

IQ による IQ UTILITIES コマンドの使用

IQ UTILITIES は、IQ のシステム・プロシージャの多くで、実行時に背後で動作する文です。多くの場合は、IQ UTILITIES が動作していることは、ユーザの関知するところではありません。IQ UTILITIES をユーザが直接実行するのは、IQ バッファ・キャッシュ・モニタを起動するときだけです。

IQ UTILITIES は、IQ のシステム・テーブルに保持されている情報を体系的に収集およびレポートするための方法となります。一般的なユーザ・インタフェースはありません。既存のシステム・プロシージャと同じように IQ UTILITIES を使用するという方法だけです。

システム・プロシージャは、情報を格納するローカル・テンポラリ・テーブルを宣言します。システム・プロシージャは IQ UTILITIES を実行してシステム・テーブルから情報を取得し、それをローカル・テンポラリ・テーブルに格納します。システム・プロシージャは、ローカル・テンポラリ・テーブルから情報をレポートするだけのこともあれば、追加的な処理を実行することもあります。

システム・プロシージャの中には、あらかじめ定義された番号を IQ UTILITIES 文の引数として指定するものもあります。この番号に応じて特定の機能が実行されます。たとえば、システム・テーブルの情報から値を派生するなどです。IQ UTILITIES の引数として使用される番号のリストについては、表 1-1 (24 ページ) を参照してください。

IQ UTILITIES を使用するための要件

この章全体で説明されている要件は、IQ UTILITIES を使用してストアド・プロシージャを作成するときにも当てはまります。加えて、きわめて重大ないくつかの要件に従う必要があります。

IQ UTILITIES をプロシージャで使用する場合には、既存のプロシージャとまったく同じように使用しなくてはなりません。具体的には、次のようにします。

- プロシージャからの結果を格納するローカル・テンポラリ・テーブルを宣言します。このテーブルは、システム・ストアド・プロシージャとまったく同じスキーマを持たなくてはなりません。カラム名、カラム幅、カラムの順序、データ型、精度などです。
- EXECUTE IMMEDIATE コマンドを発行して IQ UTILITIES を実行し、その結果をテンポラリ・テーブルに格納します。
- IQ UTILITIES 文に番号を指定する場合は、システム・ストアド・プロシージャとまったく同じ番号を、まったく同じ目的で使用しなくてはなりません。独自の番号を作成したり、既存の番号の使用方法を変更することはできません。

つまり、ローカル・テンポラリ・テーブルと IQ UTILITIES 文を、システム・ストアド・プロシージャとまったく同じように使用しなくてはならないということです。

- カラムの削除や追加を行わないでください。
- システム・プロシージャで使用されているテーブルの内容を変更しないでください。作成したプロシージャを呼び出すユーザが、同じテーブルを使用する他のプロシージャも呼び出す可能性があります。

警告！ これらの規則を破ると、IQ サーバやデータベースに重大な問題が生じる可能性があります。

IQ システム・プロシージャは、IQ のインストール・ディレクトリの *scripts* ディレクトリの *iqprocs.sql* にあります。

IQ UTILITIES の構文は次のとおりです。

IQ UTILITIES MAIN INTO *local-temp-table-name arguments*

このコマンドの使用例については、*iqprocs.sql* ファイルを参照してください。

IQ モニタへの IQ UTILITIES コマンドは、『リファレンス：文とオプション』にのみ掲載されています。使用に厳密な規則があることと、不正に使用した場合にシステムの処理にリスクを及ぼすことがその理由です。

IQ システム・プロシージャの番号は固定されています。リリースごとの違いはありません。ただし、今後のリリースで新しい番号が追加される可能性はあります。

プロシージャに付ける名前は、システム・プロシージャとは別のものにします。

呼び出すプロシージャの選択

データベースの情報をレポートする、ドキュメントに記載済みのシステム・プロシージャについては、IQ UTILITIES を使用して独自バージョンを作成するのが安全です。たとえば、sp_iqspaceused は、IQ メイン・ストアと IQ テンポラリ・ストアの使用済み領域と空き領域についての情報を表示します。作成したプロシージャの所有者をシステム・ストアド・プロシージャに照らし、正しい所有者となっていることを確認してください。

IQ 処理を制御するシステム・プロシージャについては、独自バージョンを作成してはなりません。IQ の処理を制御するプロシージャを修正すると、重大な問題につながる可能性があります。

IQ UTILITIES で使用される番号

次の表は、IQ UTILITIES コマンドで引数として使用される番号と、各番号を使用するシステム・プロシージャを示します。これらのプロシージャの機能については、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 7 章 システム・プロシージャ」を参照してください。

表 1-1：システム・プロシージャで使用される IQ UTILITIES の番号

番号	プロシージャ	コメント
10000	sp_iqtransaction	
20000	sp_iqconnection、 sp_iqmpxcountdbremote	
30000	sp_iqspaceused	
40000	sp_iqspaceinfo	
50000	sp_iqlocks	
60000	sp_iqmpxversionfetch	使用できません
70000	sp_iqmpxdumptlvlog	

番号	プロシージャ	コメント
80000	sp_iqcontext	
100000	sp_iqindexfragmentation	
110000	sp_iqrowdensity	

作成したプロシージャのテスト

作成したプロシージャのテストは、必ず開発環境から始めてください。運用環境で実行する前にプロシージャをテストすれば、IQ サーバとデータベースの安定性を維持するのに役立ちます。

第 2 章

OLAP の使用

この章について

オンライン分析処理 (OLAP: Online Analytical Processing) は、リレーショナル・データベースに格納されている情報を効率的に分析するための手法です。OLAP を使用すると、データをさまざまな次元で分析し、小計ローを含んだ結果セットを取得し、データを多次元キューブに編成するという処理をすべて 1 つの SQL クエリで行うことができます。また、フィルタを使用してデータを絞り込み、結果セットを迅速に返すことができます。この章では、Sybase IQ がサポートする SQL/OLAP 関数について説明します。

注意 以降で紹介する OLAP の例に出てくるテーブルは、iqdemo データベースに含まれています。

内容

OLAP について

1999 年の SQL 標準の改正で、ANSI SQL 標準に複雑なデータ分析機能を含めるための拡張が導入されました。Sybase IQ では、これらの SQL 拡張機能の一部が取り入れられており、これらの拡張を包括的に追加サポートしています。

この分析機能を使って複雑なデータ分析を 1 つの SQL 文で実行することができますが、これはオンライン分析処理 (OLAP) と呼ばれるソフトウェア・テクノロジに基づいています。OLAP の関数には、次のようなものが含まれています。

- GROUP BY 句の拡張機能 — CUBE、ROLLUP
- 分析関数
  - 単純な集合 — AVG、COUNT、MAX、MIN、SUM、STDDEV、VARIANCE

    注意 Grouping() 以外の単純な集合関数は OLAP ウィンドウ関数と併用できます。
  - ウィンドウ関数
    - ウィンドウ集合 — AVG、COUNT、MAX、MIN、SUM
    - ランク付け関数 — RANK、DENSE_RANK、PERCENT_RANK、NTILE
    - 統計関数 — STDDEV、STDDEV_SAMP、STDDEV_POP、VARIANCE、VAR_POP、VAR_SAMP、REGR_AVGX、REGR_AVGY、REGR_COUNT、REGR_INTERCEPT、REGR_R2、REGR_SLOPE、REGR_SXX、REGR_SXY、REGR_SYY、CORR、COVAR_POP、COVAR_SAMP、CUME_DIST、EXP_WEIGHTED_AVG、WEIGHTED_AVG
    - 分散統計関数 — PERCENTILE_CONT、PERCENTILE_DISC
  - 数値関数 — WIDTH_BUCKET、CEIL、LN、EXP、POWER、SQRT、FLOOR

データベース製品によっては、OLAP モジュールが独立しており、分析前にデータをデータベースから OLAP モジュールに移動しなければならないものもあります。一方、Sybase IQ では OLAP 機能がデータベースそのものに組み込まれているため、ストアド・プロシージャなどのデータベース機能との配備や統合を簡単かつシームレスに行うことができます。

OLAP の利点

OLAP 関数を GROUPING、CUBE、ROLLUP という拡張機能と組み合わせて使用すると、2 つの大きな利点があります。第一に、多次元のデータ分析、データ・マイニング、時系列分析、傾向分析、コストの割り当て、ゴール・シーク、一時的な多次元構造変更、非手続き型モデリング、例外の警告を多くの場合 1 つの SQL 文で実行できます。第二に、OLAP のウィンドウおよびレポート集合関数では、**ウィンドウ**という関係演算子を使用することができ、これはセルフジョインや相関サブクエリを使用するセマンティック的に等価なクエリよりも効率的に実行できます。OLAP を使用して取得した結果セットには小計ローを含めることができ、この結果セットを多次元キューブに編成することもできます。詳細については、「ウィンドウ」(46 ページ) を参照してください。

さまざまな期間での移動平均と移動和を計算したり、選択したカラムの値が変化したときに集約とランクをリセットしたり、複雑な比率を単純な言葉で表現したりできます。1 つのクエリ式のスコープ内で、それぞれ独自の分割ルールを持ついくつかの異なる OLAP 関数を定義することができます。

OLAP の評価について

OLAP の評価は、最終的な結果に影響を及ぼすクエリ実行のいくつかのフェーズとして概念化できます。OLAP の実行フェーズは、クエリ内の対応する句によって識別されます。たとえば、SQL クエリの指定にウィンドウ関数が含まれている場合は、WHERE、JOIN、GROUP BY、および HAVING 句が先に処理されます。GROUP BY 句でグループが定義された後、クエリの ORDER BY 句に含まれる最後の SELECT リストが評価される前に、パーティションが作成されます。

グループ化の際には、NULL 値はすべて同じグループと見なされます (それぞれの NULL 値が等しくない場合でも同様です)。

HAVING 句は、WHERE 句に似ており、GROUP BY 句の結果に対するフィルタとして機能します。

ANSI SQL 標準に基づく SQL 文と SELECT、FROM、WHERE、GROUP BY、HAVING 句を含んだ単純なクエリ仕様のセマンティックを考えてみます。

1　クエリにより、FROM 句のテーブル式を満たすロー・セットが取得されます。

2 WHERE 句の述部が、テーブルから取得したロー・セットに適用されます。WHERE 句の条件を満たさない (条件が true にならない) ローが除外されます。

3 残りの各ローについて、SELECT リストおよび GROUP BY 句に含まれている式 (集合関数を除く) が評価されます。

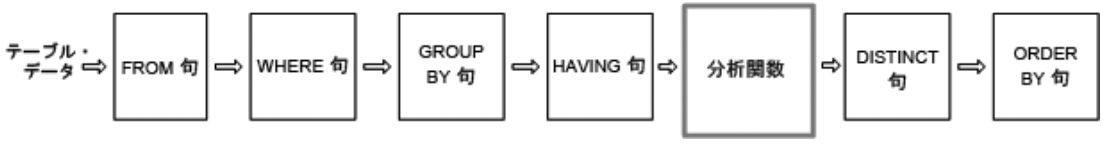
4 GROUP BY 句の式の重複しない値に基づいて、結果のローがグループ化されます (NULL はそれぞれのドメインで特殊な値として扱われます)。PARTITION BY 句がある場合は、GROUP BY 句の式はパーティション・キーとして使用されます。

5 各パーティションについて、SELECT リストまたは HAVING 句の集合関数が評価されます。いったん集合関数を適用すると、中間の結果セットには個々のテーブル・ローが含まれなくなります。新しい結果セットには、GROUP BY の式と、各パーティションについて計算した集合関数の値が含まれます。

6 HAVING 句の条件が結果グループに適用されます。HAVING 句の条件を満たさないグループが除外されます。

7 PARTITION BY 句で定義された境界に基づいて結果が分割されます。結果ウィンドウについて、OLAP ウィンドウ関数 (ランク付け関数および集合関数) が計算されます。

図 2-1：OLAP の SQL 処理



詳細については、「文法規則 2」(87 ページ) を参照してください。詳細については、「OLAP 関数の BNF 文法」(87 ページ) を参照してください。

GROUP BY 句の拡張機能

GROUP BY 句の拡張機能により、次のような処理を行う複雑な SQL 文を書くことができます。

- 入力ローを複数の次元に分割し、結果グループの複数のサブセットを組み合わせる。

- 「データ・キューブ」を作成し、データ・マイニング分析のための疎密度の多次元結果セットを用意する。
- 元のグループを含んだ結果セットを作成する (必要に応じて、小計ローと合計ローを含める場合もある)。

ROLLUP や CUBE などの OLAP の Grouping() (グループ化) 操作は、プレフィクスや小計ローとして概念化できます。

プレフィクス

GROUP BY 句を含むクエリでは、**プレフィクス**のリストが作成されます。プレフィクスとは、GROUP BY 句の項目のサブセットであり、クエリの GROUP BY 句の項目のうち最も右にある 1 つまたは複数の項目を除外することで作成されます。残りのカラムは**プレフィクス・カラム**と呼ばれます。

ROLLUP 例 1　次に示す ROLLUP のクエリの例では、GROUP BY のリストに 2 つの変数 (*Year* と *Quarter*) が含まれています。

```
SELECT year (OrderDate) AS Year, quarter(OrderDate)
   AS Quarter, COUNT(*) Orders
FROM SalesOrders
GROUP BY ROLLUP(Year, Quarter)
ORDER BY Year, Quarter
```

このクエリには次の 2 つのプレフィクスがあります。

- Quarter を除外するプレフィクス ― プレフィクス・カラムには 1 つのカラム (Year) が含まれます。
- Quarter と Year の両方を除外するプレフィクス ― プレフィクス・カラムは存在しません。

Quarter と Year を除外するプレフィクス

Quarter を除外するプレフィクス

Year	Quarter	Orders
(NULL)	(NULL)	648
2000	(NULL)	380
2000	1	87
2000	2	77
2000	3	91
2000	4	125
2001	(NULL)	268
2001	1	139
2001	2	119
2001	3	10

注意　GROUP BY リストには、項目と同じ数のプレフィクスが含まれます。

GROUP BY での ROLLUP と CUBE

プレフィクスに関する一般的なグループ化を簡単に指定するために、2 つの重要な構文簡略化パターンが用意されています。1 つ目のパターンは ROLLUP、2 つ目のパターンは CUBE と呼ばれます。

GROUP BY ROLLUP

ROLLUP 演算子には、引数として適用するグループ化式を、次の構文の中で順序リストで指定します。

SELECT … [**GROUPING** (*column-name*) …] …
GROUP BY [*expression* [, …]
| **ROLLUP** (*expression* [, …])]

GROUPING は、カラム名をパラメータとして受け取り、表 2-1 に示すようにブール値を返します。

表 2-1：ROLLUP 演算子が指定された GROUPING によって返される値

結果値の種類	GROUPING の戻り値
ROLLUP 演算子によって作成された NULL	1 (真)
ローが小計であることを示す NULL	1 (真)
ROLLUP 演算子によって作成された以外の NULL	0 (偽)
格納されていた NULL	0 (偽)

ROLLUP は、まず GROUP BY 句に指定された標準的な集合関数値を計算します。次に、ROLLUP はグループ化を行うカラムのリストを右から左に移動し、より高いレベルの小計を連続して作成します。最後に総計が作成されます。グループ化するカラムの数が *n* 個の場合、ROLLUP は *n*+1 レベルの小計を作成します。

SQL 構文の例	定義されるセット
`GROUP BY ROLLUP (A, B, C);`	(A, B, C) (A, B) (A) ( )

ROLLUP と小計ロー

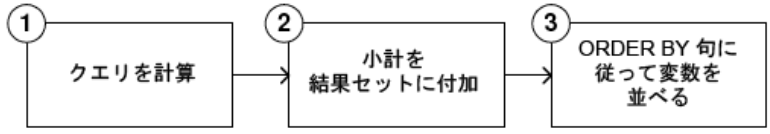
ROLLUP は、GROUP BY のクエリ・セットに対して UNION を行うのと同じことです。次の 2 つのクエリの結果セットは等しくなります。GROUP BY (A, B) の結果セットは、A と B に定数が含まれているすべてのローについての小計から成ります。UNION を可能にするために、カラム C には NULL が割り当てられます。

ROLLUP クエリの例	ROLLUP を使用せずに記述した同じ内容のクエリ
SELECT A, B, C, SUM(D) FROM T1 GROUP BY ROLLUP (A, B, C);	SELECT * FROM (T1 (SELECT A, B, C, SUM(D) GROUP BY A, B, C) UNION ALL (SELECT A, B, NULL, SUM(D) GROUP BY A, B) UNION ALL (SELECT A, NULL, NULL, SUM(D) GROUP BY A) UNION ALL (SELECT NULL, NULL, NULL, SUM(D)))

小計ローはデータの分析に役立ちます。特に、データが大量にある場合、データにさまざまな次元がある場合、データがさまざまなテーブルに含まれている場合、あるいはまったく異なるデータベースに含まれている場合に威力を発揮します。たとえば販売マネージャが、売上高についてのレポートを営業担当者別、地域別、四半期別に整理して、売上パターンの理解に役立てることができます。データの小計は、販売マネージャが売上高の全体像をさまざまな視点から分析するのに役立ちます。販売マネージャが比較したいと考える基準に基づいて要約情報が提供されていれば、データの分析を容易に行うことができます。

OLAP を使用すると、ローおよびカラムの小計を分析、計算する処理をユーザの目から隠すことができます。図 2-2 に、Sybase IQ での小計の計算の概念を示します。

図 2-2：小計



1　このステップで、まだ ROLLUP とは見なされない中間の結果セットが生成されます。

2　小計が評価され、結果セットに付加されます。

3　クエリ内の ORDER BY 句に従ってローが並べられます。

NULL 値と小計ロー

GROUP BY 操作に対する入力のローに NULL が含まれているときは、その中に、ROLLUP または CUBE 操作によって追加された小計ローと、最初の入力データの一部として NULL 値を含んでいるローが混在している可能性があります。

Grouping() 関数は、小計ローをその他のローから区別します。具体的には、GROUP BY リストのカラムを引数として受け取り、そのカラムが小計ローであるために NULL になっている場合は 1 を返し、それ以外の場合は 0 を返します。

次の例では、結果セットの中に Grouping() カラムが含まれています。強調表示されているローは、小計ローであるために NULL を含んでいるのではなく、入力データの結果として NULL を含んでいるローです。Grouping() カラムは強調表示されています。このクエリは、employee テーブルと sales_order テーブルの間の外部ジョインです。このクエリでは、テキサス、ニューヨーク、またはカリフォルニアに住んでいる女性従業員を選択しています。営業担当者でない (したがって売上がない) 女性従業員については、カラムに NULL が表示されます。

注意 たとえば、Sybase IQ デモ・データベース iqdemo.db を使用します。インストールについては、「デモ・データベース」(xvi ページ) を参照してください。

```
SELECT Employees.EmployeeID as EMP, year(OrderDate) as
    YEAR, count(*) as ORDERS, grouping(EMP) as
    GE, grouping(YEAR) as GY
    FROM Employees LEFT OUTER JOIN SalesOrders on
    Employees.EmployeeID =
SalesOrders.SalesRepresentative
    WHERE Employees.Sex IN ('F') AND Employees.State
    IN ('TX', 'CA', 'NY')
GROUP BY ROLLUP (YEAR, EMP)
ORDER BY YEAR, EMP
```

このクエリの結果セットを次に示します。

```
EmployeeID  YEAR      Orders     GE     GY
------      ----      ------     --     --
NULL        NULL           1      1      0
NULL        NULL         165      1      1
1090        NULL           1      0      0
NULL        2000          98      1      0
 667        2000          34      0      0
 949        2000          31      0      0
1142        2000          33      0      0
NULL        2001          66      1      0
 667        2001          20      0      0
 949        2001          22      0      0
1142        2001          24      0      0
```

個々のプレフィクスについて、プレフィクス・カラムに同じ値が含まれているすべてのローに関する**小計ロー**が作成されます。

ROLLUP の結果を具体的に説明するために、前述のクエリの例をもう一度詳しく見ていきます。

```
SELECT year (OrderDate) AS Year, quarter
   (OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
   GROUP BY ROLLUP (Year, Quarter)
   ORDER BY Year, Quarter
```

このクエリでは、Year カラムを含んでいるプレフィクスにより、Year=2000 の合計ローと Year=2001 の合計ローが作成されます。このプレフィクスに関する 1 つの合計ローはカラムを含んでいません。これは、中間の結果セットに含まれているすべてのローの小計です。

小計ローの各カラムの値は、次のようになっています。

- プレフィクスに含まれているカラム — そのカラムの値です。たとえば前述のクエリでは、Year=2000 のローに関する小計ローの Year カラムの値は 2000 になります。
- プレフィクスから除外されたカラム — NULL です。たとえば、Year カラムから成るプレフィクスにより生成された小計ローでは、Quarter カラムの値は NULL になります。
- 集合関数 — 除外されているカラムの値を計算した結果です。

  小計値は、集約されたローではなく基本データのローに対して計算されます。多くの場合、たとえば SUM や COUNT などでは結果は等しくなりますが、AVG、STDDEV、VARIANCE などの統計関数では結果が異なってくるため、この区別は重要です。

ROLLUP 演算子には次の制限があります。

- ROLLUP 演算子は、COUNT DISTINCT と SUM DISTINCT を除き、GROUP BY 句で使用可能なすべての集合関数をサポートしています。
- ROLLUP は SELECT 文でのみ使用できます。サブクエリでは ROLLUP を使用できません。
- 1 つの GROUP BY 句の中で複数の ROLLUP、CUBE、および GROUP BY カラムを組み合わせるグループ化の指定は、現時点ではサポートされていません。
- GROUP BY のキーに定数式を指定することはできません。

式の一般的なフォーマットについては、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「式」と「SQL 言語の要素」を参照してください。

ROLLUP 例 2 次は、ROLLUP と GROUPING の使用例です。GROUPING によって作成される一連のマスク・カラムを表示します。カラム S、N、C に表示されている数字 0 と 1 は、GROUPING からの戻り値で ROLLUP の結果の値を表現しています。マスクが“011”であれば小計のローであり、“111”であれば総計のローであると特定できます。これを利用して、クエリの結果をプログラムで分析することが可能です。

```
SELECT size, name, color, SUM(quantity),
    GROUPING(size) AS S,
    GROUPING(name) AS N,
    GROUPING(color) AS C
FROM Products
GROUP BY ROLLUP(size, name, color) HAVING (S=1 or N=1
or C=1)
ORDER BY size, name, color;
```

このクエリの結果セットを次に示します。

size	name	color	SUM	S	N	C
(NULL)	(NULL)	(NULL)	496	1	1	1
Large	(NULL)	(NULL)	71	0	1	1
Large	Sweatshirt	(NULL)	71	0	0	1
Medium	(NULL)	(NULL)	134	0	1	1
Medium	Shorts	(NULL)	80	0	0	1
Medium	Tee Shirt	(NULL)	54	0	0	1
One size fits all	(NULL)	(NULL)	263	0	1	1
One size fits all	Baseball Cap	(NULL)	124	0	0	1
One size fits all	Tee Shirt	(NULL)	75	0	0	1
One size fits all	Visor	(NULL)	64	0	0	1
Small	(NULL)	(NULL)	28	0	1	1
Small	Tee Shirt	(NULL)	28	0	1	1

注意 ROLLUP 例 2 の結果では、SUM カラムは *SUM(products.quantity)* で表示します。

ROLLUP 例 3　次の例は、GROUPING を使用して、最初から格納されていた NULL 値と ROLLUP 操作によって生成された“NULL”値とを区別する方法を示しています。このクエリで指定されているとおり、最初から格納されていた NULL 値はカラム prod_id に [NULL] として表示され、ROLLUP によって生成された“NULL”値はカラム PROD_IDS で ALL に置き換えられます。

```
SELECT year(ShipDate) AS Year, ProductID, SUM(quantity)
AS OSum, CASE WHEN GROUPING(Year) = 1 THEN 'ALL' ELSE
CAST(Year AS char(8)) END, CASE WHEN
GROUPING(ProductID) = 1 THEN 'ALL' ELSE CAST(ProductID
as char(8)) END
FROM SalesOrderItems
GROUP BY ROLLUP(Year, ProductID) HAVING OSum > 36
ORDER BY Year, ProductID;
```

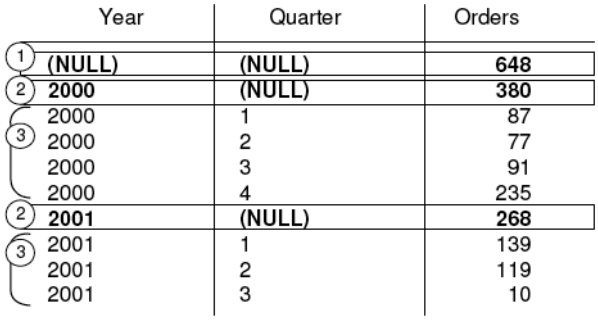
このクエリの結果セットを次に示します。

```
Year      ProductID   0SumPROD_IDS
---------     -------   ---   ----------   --------
NULL           NULL   28359   ALL          ALL
2000           NULL   17642   2000         ALL
2000           300     1476   2000         300
2000           301     1440   2000         301
2000           302     1152   2000         302
2000           400     1946   2000         400
2000           401     1596   2000         401
2000           500     1704   2000         500
2000           501     1572   2000         501
2000           600     2124   2000         600
2000           601     1932   2000         601
2000           700     2700   2000         700
2001           NULL   10717   2001         ALL
2001           300      888   2001         300
2001           301      948   2001         301
2001           302      996   2001         302
2001           400     1332   2001         400
2001           401     1105   2001         401
2001           500      948   2001         500
2001           501      936   2001         501
2001           600      936   2001         600
2001           601      792   2001         601
2001           700     1836   2001         700
```

ROLLUP 例 4 次のクエリ例は、注文数を年別および四半期別に集計したデータを返します。

```
SELECT year (OrderDate) AS Year, quarter
(OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
GROUP BY ROLLUP (Year, Quarter)
ORDER BY Year, Quarter
```

次の図は、このクエリの結果を示しています。結果セット内の小計ローは強調表示されています。各小計ローでは、その小計の計算対象になったカラムに NULL 値が格納されています。



	Year	Quarter	Orders
1	**(NULL)**	**(NULL)**	**648**
2	**2000**	**(NULL)**	**380**
3	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	235
2	**2001**	**(NULL)**	**268**
3	2001	1	139
	2001	2	119
	2001	3	10

ロー [1] は、両方の年 (2000 年および 2001 年) のすべての四半期の注文数の合計を示しています。このローは、Year カラムと Quarter カラムの両方が NULL であり、すべてのカラムがプレフィクスから除外されています。

注意 ROLLUP 操作によって返される結果セットには、集合カラムを除くすべてのカラムが NULL であるローが必ず 1 つ含まれています。このローは、集合関数に対する全カラムの要約を表しています。たとえば、集合関数として SUM を使用している場合は、このローはすべての値の総計を表します。

ロー [2] は、2000 年および 2001 年の注文数の合計をそれぞれ示しています。どちらのローも、Quarter カラムの値は NULL になっています。このカラムの値を加算して、Year の小計を出しているためです。結果セットにこのようなローがいくつ含まれるかは、ROLLUP クエリに登場する変数の数によって決まります。

[3] としてマークされている残りのローは要約情報を示し、それぞれの年の各四半期の注文数の合計を表しています。

ROLLUP 例 5　この ROLLUP 操作の例では、年別、四半期別、地域別の注文数を集計するというやや複雑な結果セットを返します。この例では、第 1 および第 2 四半期と 2 つの地域 (カナダと東部地区) だけを分析します。

```
SELECT year(OrderDate) AS Year, quarter(OrderDate)
AS Quarter, region, COUNT(*) AS Orders
FROM SalesOrders WHERE region IN ('Canada',
'Eastern') AND quarter IN (1, 2)
GROUP BY ROLLUP (Year, Quarter, Region)
ORDER BY Year, Quarter, Region
```

次の図は、このクエリの結果セットを示しています。各小計ローでは、その小計の計算対象になったカラムに NULL が格納されています。

	Year	Quarter	Region	Orders
1	**(NULL)**	**(NULL)**	**(NULL)**	**183**
	2000	**(NULL)**	**(NULL)**	**68**
	2000	1	(NULL)	36
	2000	1	Canada	3
	2000	1	Eastern	33
2	2000	2	(NULL)	32
	2000	2	Canada	3
	2000	2	Eastern	29
	2001	**(NULL)**	**(NULL)**	**115**
	2001	1	(NULL)	57
	2001	1	Canada	11
	2001	1	Eastern	46
	2001	2	(NULL)	58
	2001	2	Canada	4
	2001	2	Eastern	54

ロー [1] はすべてのローの集約結果であり、Year、Quarter、Region カラムに NULL が含まれています。このローの Orders カラムの値は、カナダと東部地区の 2000 年および 2001 年の第 1 および第 2 四半期の注文数の合計を示しています。

[2] としてマークされているローは、それぞれの年 (2000 年と 2001 年) におけるカナダと東部地区の第 1 および第 2 四半期の注文数の合計を示しています。ロー [2] の値を足すと、ロー [1] に示されている総計に等しくなります。

[3] としてマークされているローは、特定の年および四半期の全地域の注文数の合計を示しています。

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	**1**	**(NULL)**	**36**
2000	1	Canada	3
2000	1	Eastern	33
2000	**2**	**(NULL)**	**32**
2000	2	Canada	3
2000	2	Eastern	29
2001	(NULL)	(NULL)	115
2001	**1**	**(NULL)**	**57**
2001	1	Canada	11
2001	1	Eastern	46
2001	**2**	**(NULL)**	**58**
2001	2	Canada	4
2001	2	Eastern	54

[4] としてマークされているローは、結果セット内のそれぞれの年の各四半期の各地域の注文の合計数を示しています。

Year	Quarter	Region	Orders
(NULL)	(NULL)	(NULL)	183
2000	(NULL)	(NULL)	68
2000	1	(NULL)	36
2000	1	Canada	3
2000	1	Eastern	33
2000	2	(NULL)	32
2000	2	Canada	3
2000	2	Eastern	29
2001	(NULL)	(NULL)	115
2001	1	(NULL)	57
2001	1	Canada	11
2001	1	Eastern	46
2001	2	(NULL)	58
2001	2	Canada	4
2001	2	Eastern	54

GROUP BY CUBE

GROUP BY 句の CUBE 演算子は、データを複数の次元 (グループ化式) でグループ化することでデータを分析します。CUBE に次元の順序リストを引数として指定すると、SELECT 文の中で、そのクエリに指定した次元の考えられるすべての組み合わせの小計を計算し、選択した複数のカラムのすべての値の組み合わせについての要約を示す結果セットを生成することができます。

CUBE の構文は次のとおりです。

SELECT … [**GROUPING** (*column-name*) …] …
GROUP BY [*expression* [,…]
| **CUBE** (*expression* [,…])]

GROUPING は、カラム名をパラメータとして受け取り、表 2-2 に示すようにブール値を返します。

表 2-2：CUBE 演算子が指定された GROUPING によって返される値

結果値の種類	GROUPING の戻り値
CUBE 演算子によって作成された NULL	1 (真)
ローが小計であることを示す NULL	1 (真)
CUBE 演算子によって作成された以外の NULL	0 (偽)
格納されていた NULL	0 (偽)

CUBE は、同じ階層の一部ではない次元を扱うときに特に威力を発揮します。

SQL 構文の例	定義されるセット
`GROUP BY CUBE (A, B, C);`	(A, B, C) (A, B) (A, C) (A) (B, C) (B) (C) ( )

CUBE 演算子には次の制限があります。

- CUBE 演算子は GROUP BY 句で使用可能なすべての集合関数をサポートしますが、CUBE は現在 COUNT DISTINCT および SUM DISTINCT ではサポートされていません。

- CUBE は、現在、逆分散統計関数である PERCENTILE_CONT と PERCENTILE_DISC ではサポートされていません。
- CUBE は SELECT 文でのみ使用できます。CUBE を SELECT のサブクエリで使用することはできません。
- 1 つの GROUP BY 句の中で ROLLUP、CUBE、および GROUP BY カラムを組み合わせる GROUPING の指定は、現時点ではサポートされていません。
- GROUP BY のキーに定数式を指定することはできません。

注意 キューブのサイズがテンポラリ・キャッシュのサイズを超えると、CUBE のパフォーマンスが低下します。

GROUPING と CUBE 演算子を併用すると、格納されていた NULL 値と CUBE によって作成されたクエリ結果の NULL 値を区別することができます。

GROUPING 関数を使用して結果を分析する方法については、ROLLUP 演算子の説明で紹介した例を参照してください。

CUBE 操作が返す結果セットには、集約カラムを除くすべてのカラムの値が NULL であるローが少なくとも 1 つは含まれています。このローは、集合関数に対する全カラムの要約を表しています。

CUBE 例 1 次の例は、対象者の州 (地理的な位置)、性別、教育レベル、および収入などで構成される調査データを使用したクエリです。最初に紹介するクエリには GROUP BY 句が指定されています。この句は、クエリの結果を census テーブルの state、gender、education カラムの値に応じてロー・グループに分類し、収入の平均とローの合計数をグループごとに計算します。このクエリには GROUP BY 句だけを使用し、ローのグループ化に CUBE 演算子を使用していません。

```
SELECT State, Sex as gender, DepartmentID, COUNT(*),
CAST(ROUND(AVG(Salary),2) AS NUMERIC(18,2))
AS AVERAGE
FROM Employees WHERE state IN ('MA' , 'CA')
GROUP BY State, Sex, DepartmentID
ORDER BY 1,2;
```

このクエリの結果セットを次に示します。

```
state  gender  DepartmentID  COUNT(*)     AVERAGE
-----  ------  ------------  --------     --------
CA       F        200          2          58650.00
CA       M        200          1          39300.00
```

GROUP BY 句の CUBE 拡張機能を使用すると、調査データを 1 回参照するだけで、調査データ全体における州別、性別、教育別の平均収入を計算し、state、gender、education カラムの考えられるすべての組み合わせにおける平均収入を計算することができます。CUBE 演算子を使用すると、たとえば、すべての州における全女性の平均収入を計算したり、調査対象者全員の平均収入を、各自の教育別および州別に計算したりすることができます。

CUBE でグループを計算するときには、計算されたグループのカラムに NULL 値が挿入されます。最初からデータベース内に格納されていた NULL なのか、CUBE の結果として生成された NULL なのかを区別するためには、GROUPING 関数を使用する必要があります。GROUPING 関数は、指定されたカラムが上位レベルのグループにマージされている場合は 1 を返します。

CUBE 例 2　次のクエリは、GROUPING 関数を GROUP BY CUBE と組み合わせた使用例です。

```
SELECT case grouping(State) WHEN 1 THEN 'ALL' ELSE State

END AS c_state, case grouping(sex) WHEN 1 THEN 'ALL'

ELSE Sex end AS c_gender, case grouping(DepartmentID)

WHEN 1 THEN 'ALL' ELSE cast(DepartmentID as char(4)) end

AS c_dept, COUNT(*), CAST(ROUND(AVG(salary),2) AS

NUMERIC(18,2))AS AVERAGE

FROM employees WHERE state IN ('MA' , 'CA')

GROUP BY CUBE(state, sex, DepartmentID)

ORDER BY 1,2,3;
```

このクエリの結果は次のとおりです。CUBE が生成した小計ローを示す NULL 値が、クエリ内の指定によって小計ローで ALL に置き換わっています。

```
c_state   c_gender  c_dept    COUNT(*)   AVERAGE
-------   --------  -------    -----     --------
ALL       ALL        200         3       52200.00
ALL       ALL        ALL         3       52200.00
ALL       F          200         2       58650.00
ALL       F          ALL         2       58650.00
```

```
ALL     M       200     1       39300.00
ALL     M       ALL     1       39300.00
CA      ALL     200     3       52200.00
CA      ALL     ALL     3       52200.00
CA      F       200     2       58650.00
CA      F       ALL     2       58650.00
CA      M       200     1       39300.00
CA      M       ALL     1       39300.00
```

CUBE 例 3 この例のクエリは、注文数の合計を要約する結果セットを返し、次に、年別および四半期別の注文数の小計を計算します。

注意 比較する変数の数が増えると、キューブの計算のコストが急激に増大します。

```
SELECT year (OrderDate) AS Year, quarter
(OrderDate) AS Quarter, COUNT (*) Orders
FROM SalesOrders
GROUP BY CUBE (Year, Quarter)
ORDER BY Year, Quarter
```

次の図は、このクエリの結果セットを示しています。この結果セットでは、小計ローが強調表示されています。各小計ローでは、その小計の計算対象になったカラムに NULL が格納されています。

	Year	Quarter	Orders
1	**(NULL)**	**(NULL)**	**648**
2	**(NULL)**	**1**	**226**
	(NULL)	**2**	**196**
	(NULL)	**3**	**101**
	(NULL)	**4**	**125**
3	**2000**	**(NULL)**	**380**
	2000	1	87
	2000	2	77
	2000	3	91
	2000	4	125
3	**2001**	**(NULL)**	**268**
	2001	1	139
	2001	2	119
	2001	3	10

先頭のロー [1] は、両方の年のすべての四半期の注文数の合計を示しています。Orders カラムの値は、[3] としてマークされている各ローの値の合計です。これは、[2] としてマークされている 4 つのローの値の合計でもあります。

[2] としてマークされている一連のローは、両方の年の四半期別の注文数の合計を示しています。[3] としてマークされている 2 つのローは、それぞれ 2000 年および 2001 年のすべての四半期の注文数の合計を示しています。

分析関数

Sybase IQ では、1 つの SQL 文内で複雑なデータ分析を実行できる機能を備えた単純な集合関数とウィンドウ集合関数の両方を提供しています。これらの関数を使用して、たとえば「ダウ工業株 30 種平均の四半期の移動平均はどうなっているか」または「各部署のすべての従業員とその累積給与を一覧表示せよ」というクエリに対する結果を計算することができます。さまざまな期間における移動平均と累積和を計算したり、パーティション値が変化したときに集合計算がリセットされるような方法で集約とランクを分割したりできます。1 つのクエリ式のスコープ内で、それぞれ独自の分割ルールを持ついくつかの異なる OLAP 関数を定義することができます。統計関数は 2 つのカテゴリに分けられます。

- 単純な集合関数 (AVG、COUNT、MAX、MIN、SUM など) は、データベースに含まれるローのグループのデータを要約します。SELECT 文の GROUP BY 句を使ってグループを作成します。
- 1 つの引数を取る単項の統計集合関数には、STDDEV、STDDEV_SAMP、STDDEV_POP、VARIANCE、VAR_SAMP、および VAR_POP() があります。

単純な集合関数でも単項の集合関数でも、データベース内のローのグループに関するデータを要約することができ、ウィンドウ指定と組み合わせて、処理の際に結果セットに対する移動ウィンドウを計算することができます。

注意　集合関数 AVG、SUM、STDDEV、STDDEV_POP、STDDEV_SAMP、VAR_POP、VAR_SAMP、VARIANCE は、バイナリ・データ型である BINARY と VARBINARY をサポートしていません。

単純な集合関数

単純な集合関数 (AVG、COUNT、MAX、MIN、SUM など) は、データベースに含まれるローのグループのデータを要約します。SELECT 文の GROUP BY 句を使ってグループを作成します。集合関数は、select リストと、SELECT 文の HAVING 句および ORDER BY 句の中だけで使用できます。

注意 Grouping() 関数を除き、単純な集合関数と単項の集合関数はどちらも、SQL クエリの指定に「ウィンドウ句」(**ウィンドウ**) を組み込むウィンドウ関数として使用できます。これにより、処理時に結果セットに対して概念的に移動ウィンドウを作成することができます。詳細については、「ウィンドウ」(46 ページ) を参照してください。

『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 4 章 SQL 関数」の「集合関数」を参照してください。

ウィンドウ

OLAP に関する ANSI SQL 拡張で導入された主な機能は、「**ウィンドウ**」という名前の構成体です。このウィンドウ拡張により、ユーザはクエリの結果セット (クエリの論理パーティション) をパーティションと呼ばれるローのグループに分割し、現在のローについて集約するローのサブセットを決定することができます。

1 つのウィンドウで 3 つのウィンドウ関数クラスを使用できます。それらは、ランク付け関数、ロー・ナンバリング関数、およびウィンドウ集合関数です。

```
<WINDOWED TABLE FUNCTION TYPE> ::=
    <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
    | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
    | <WINDOW AGGREGATE FUNCTION>
```

詳細については、「文法規則 6」(88 ページ) を参照してください。

ウィンドウ拡張は、ウィンドウ名または指定に対するウィンドウ関数の種類を指定し、1 つのクエリ式のスコープ内のパーティション化された結果セットに適用されます。ウィンドウ・パーティションは、特殊な OVER 句の 1 つ以上のカラムで定義されている、クエリから返されるローのサブセットです。

```
olap_function() OVER (PARTITION BY col1, col2...)
```

ウィンドウ操作では、パーティション内の各ローのランク付け、パーティション内のローの値の分布、および類似の操作などの情報を設定できます。また、データの移動平均や合計を計算し、データおよびそのデータの操作に対する影響を評価する機能を拡張することもできます。

OLAP ウィンドウの 3 つの重要な側面

OLAP ウィンドウは、ウィンドウ・パーティション、ウィンドウ順序、ウィンドウ・フレームという 3 つの重要な側面から成ります。それぞれの要素は、その時点でウィンドウ内で可視となるデータ・ローに大きな影響を与えます。また、OLAP の OVER 句は、次の 3 つの特徴的な機能により、OLAP 関数を他の統計関数やレポート関数から区別します。

- ウィンドウ・パーティションの定義 (PARTITION BY 句)。「ウィンドウ・パーティション」(48 ページ) を参照してください。
- パーティション内でのローの順序付け (ORDER BY 句)。詳細については、「ウィンドウ順序」(49 ページ) を参照してください。
- ウィンドウ・フレームの定義 (ROWS/RANGE 指定)。「ウィンドウ・フレーム」(50 ページ) を参照してください。

複数のウィンドウ関数を指定したり、冗長なウィンドウ定義を避けたりするために、OLAP ウィンドウ指定に関して名前を指定できます。その場合は、キーワード WINDOW の後に少なくとも 1 つのウィンドウ定義を指定します (複数指定する場合はカンマで区切ります)。ウィンドウ定義には、クエリ内でウィンドウを識別するための名前と、ウィンドウ・パーティション、順序、フレームを定義するためのウィンドウ指定の詳細を含めます。

```
<WINDOW CLAUSE> ::= <WINDOW DEFINITION LIST>

<WINDOW DEFINITION LIST> ::=
    <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
    } . . . ]

<WINDOW DEFINITION> ::=
    <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>

<WINDOW SPECIFICATION DETAILS> ::=
    [ <EXISTING WINDOW NAME> ]
    [ <WINDOW PARTITION CLAUSE> ]
    [ <WINDOW ORDER CLAUSE> ]
    [ <WINDOW FRAME CLAUSE> ]
```

ウィンドウ・パーティション内の各ローについて、ウィンドウ・フレームを定義することができます。ウィンドウ・フレームにより、パーティションの現在のローに対して計算を実行するときに使われるローの範囲を変更することができます。現在のローは、ウィンドウ・フレームの開始ポイントと終了ポイントを決定するための参照ポイントとなります。

ウィンドウのサイズは、物理的なローの数 (ウィンドウ・フレーム単位 ROWS を定義するウィンドウ指定を使用) または論理的な数値の間隔 (ウィンドウ・フレーム単位 RANGE を定義するウィンドウ指定を使用) に基づきます。詳細については、「ウィンドウ・フレーム」(50 ページ)を参照してください。

OLAP のウィンドウ操作では、次のカテゴリの関数を使用できます。

- 「ランク付け関数」(60 ページ)
- 「ウィンドウ集合関数」(65 ページ)
- 「統計集合関数」(67 ページ)
- 「分散統計関数」(70 ページ)

ウィンドウ・パーティション

ウィンドウ・パーティションとは、PARTITION BY 句を使用して、ユーザ指定の結果セット (入力ロー) を分割することです。パーティションは、カンマで区切られた 1 つ以上の値の式によって定義されます。パーティションに分割されたデータは暗黙的にソートされ、デフォルトのソート順序は昇順 (ASC) になります。

```
<WINDOW PARTITION CLAUSE> ::=
   PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

ウィンドウ・パーティション句を指定しなかった場合は、入力が 1 つのパーティションとして扱われます。

注意 統計関数に対して**パーティション**という用語を使用した場合は、結果セットのローを PARTITION BY 句に基づいて分割することだけを意味します。

ウィンドウ・パーティションは任意の式に基づいて定義できます。また、ウィンドウ・パーティションの処理はグループ化の後に行われるので (GROUP BY 句が指定されている場合)、SUM、AVG、VARIANCE などの集合関数の結果をパーティションの式で使用することができます。したがって、パーティションを使用すると、GROUP BY 句や ORDER BY 句とはまた別に、グループ化と順序付けの操作を実行することができます。たとえば、ある数量の最大 SUM を求めるなど、集合関数に対して集合関数を計算するクエリを記述できます。

GROUP BY 句がない場合でも、PARTITION BY 句を指定できます。

ウィンドウ順序

ウィンドウ順序とは、ウィンドウ・パーティション内の結果 (ロー) をウィンドウ順序句に基づいて並べることです。ウィンドウ順序句には、1 つ以上の値の式をカンマ区切りで指定します。ウィンドウ順序句を指定しなかった場合は、入力ローが任意の順序で処理されることがあります。

```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

OLAP のウィンドウ順序句は、非ウィンドウ・クエリの式に指定できる ORDER BY 句とは異なります。詳細については、「文法規則 31」(90 ページ) を参照してください。

OLAP 関数で使用する ORDER BY 句は、通常はウィンドウ・パーティション内のローをソートするための式を定義しますが、PARTITION BY 句がなくても ORDER BY 句を使用することができます。その場合は、このソート指定によって、確実に意味のある (かつ意図どおりの) 順序で並べられた中間の結果セットに OLAP 関数を適用することができます。

OLAP のランク付け関数には順序の指定が必須であり、ランキング値の基準は、ランク付け関数の引数ではなく ORDER BY 句で指定します。OLAP の集合関数では、通常は ORDER BY 句の指定は必須ではありませんが、ウィンドウ・フレームを定義するときには必須とされています (「ウィンドウ・フレーム」(50 ページ) を参照してください)。これは、各フレームの適切な集合値を計算する前に、パーティション内のローをソートしなければならないためです。

この ORDER BY 句には、昇順および降順のソートを定義するためのセマンティックと、NULL 値の取り扱いに関する規則を指定します。OLAP 関数は、デフォルトでは昇順 (最も小さい値が 1 番目にランク付けされる) を使用します。

これは SELECT 文の最後に指定する ORDER BY 句のデフォルト動作と同じですが、連続的な計算を行う場合にはわかりにくいかもしれません。OLAP の計算では、降順 (最も大きい値が 1 番目にランク付けされる) でのソートが必要になることがよくあります。この要件を満たすには、ORDER BY 句に明示的に DESC キーワードを指定する必要があります。

注意 ランク付け関数は、ソートされた入力のみを扱うように定義されているため、「ウィンドウ順序句」の指定を必要とします。「クエリ指定」の「order by 句」と同様に、デフォルトのソート順序は昇順です。

「ウィンドウ・フレーム単位」で RANGE を使用する場合も、「ウィンドウ順序句」を指定する必要があります。RANGE の場合は、「ウィンドウ順序句」に 1 つの式のみを指定します。「ウィンドウ・フレーム」を参照してください。

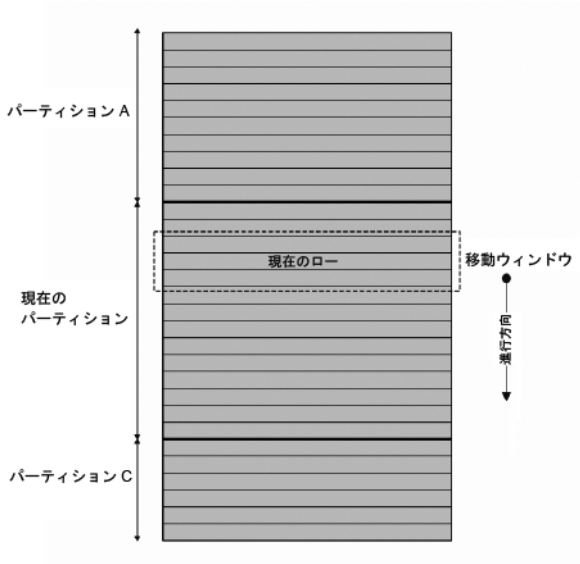
ウィンドウ・フレーム

ランク付け関数を除く OLAP 集合関数では、ウィンドウ・フレーム句を使用してウィンドウ・フレームを定義することができます。ウィンドウ・フレーム句には、現在のローを基準としてウィンドウの開始位置と終了位置を指定します。

```
<WINDOW FRAME CLAUSE> ::=
   <WINDOW FRAME UNIT>
   <WINDOW FRAME EXTENT>
```

これにより、パーティション全体の固定的な内容ではなく、移動するフレームの内容に対して OLAP 関数を計算できます。定義にもよりますが、パーティションには開始ローと終了ローがあり、ウィンドウ・フレームは開始ポイントからパーティションの終了位置に向けてスライドします。

図 2-3：分割された入力と、3 ロー分の移動ウィンドウ



UNBOUNDED PRECEEDING と FOLLOWING

ウィンドウ・フレームは、パーティションの先頭 (UNBOUNDED PRECEDING)、最後 (UNBOUNDED FOLLOWING)、または両方まで到達する無制限の集合グループによって定義されます。

UNBOUNDED PRECEDING には、パーティション内の現在のロー以前にあるすべてのローが含まれており、ROWS または RANGE で指定できます。UNBOUNDED FOLLOWING には、パーティション内の現在のロー以後にあるすべてのローが含まれており、ROWS または RANGE で指定できます。詳細については、「ROWS」(53 ページ) と「RANGE」(56 ページ) を参照してください。

FOLLOWING の値では、現在のロー以降にあるローの範囲または数を指定します。ROWS を指定する場合、その値には、ローの数を表す正の数を指定します。RANGE を指定する場合、そのウィンドウには、現在のローに指定の数値を足した数よりも少ないローが含まれます。RANGE を指定する場合、そのウィンドウ値のデータ型は、ORDER BY 句のソート・キー式の型に対応している必要があります。指定できるソート・キー式は 1 つだけで、このソート・キー式のデータ型は「加算」を許可していなければなりません。

PREDCEEDING の値では、現在のロー以前にあるローの範囲または数を指定します。ROWS を指定する場合、その値には、ローの数を表す正の数を指定します。RANGE を指定する場合、そのウィンドウには、現在のローから指定の数値を引いた数よりも少ないローが含まれます。RANGE を指定する場合、そのウィンドウ値のデータ型は、ORDER BY 句のソート・キー式の型に対応している必要があります。指定できるソート・キー式は 1 つだけで、このソート・キー式のデータ型は「減算」を許可していなければなりません。1 つ目のバインドされたグループで CURRENT ROW または FOLLOWING の値を指定している場合は、2 つ目のバインドされたグループにこの句を指定することはできません。

BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING の組み合わせを使用すると、グループ化したクエリとのジョインを構築しなくても、パーティション全体についての集合を計算できます。パーティション全体についての集合は、レポート集合とも呼ばれます。

CURRENT ROW の概念

物理的な集合グループでは、現在のローに対する相対位置に基づき、隣接するローの数に応じて、ローを含めるか除外するかが判断されます。現在のローは、クエリの中間結果における次のローへの参照にすぎません。現在のローが前に進むと、ウィンドウ内に含まれる新しいロー・セットに基づいてウィンドウが再評価されます。現在のローをウィンドウ内に含めるという要件はありません。

ウィンドウ・フレーム句を指定しなかった場合のデフォルトのウィンドウ・フレームは、ウィンドウ順序句を指定しているかどうかによって異なります。

- ウィンドウ指定にウィンドウ順序句が含まれている場合は、ウィンドウの開始ポイントは UNBOUNDED PRECEDING、終了ポイントは CURRENT ROW になり、累積値の計算に適した可変サイズのウィンドウになります。
- ウィンドウ指定にウィンドウ順序句が含まれていない場合は、ウィンドウの開始ポイントは UNBOUNDED PRECEDING、終了ポイントは UNBOUNDED FOLLOWING になり、現在のローに関係なく固定サイズのウィンドウになります。

注意 ウィンドウ・フレーム句はランク付け関数とは併用できません。

ローベース (ロー指定) または値ベース (範囲指定) のウィンドウ・フレーム単位を指定してウィンドウを定義することもできます。

```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW
FRAME BETWEEN>
```

ウィンドウ・フレーム句で BETWEEN を使用するときは、ウィンドウ・フレームの開始ポイントと終了ポイントを明示的に指定します。

ウィンドウ・フレーム句でこの 2 つの値のどちらか一方しか指定しなかった場合は、他方の値がデフォルトで CURRENT ROW になります。

ROWS

ウィンドウ・フレーム単位 ROWS では、現在のローの前後に指定の数のローを含んでいるウィンドウを定義します (現在のローは、ウィンドウの開始ポイントと終了ポイントを決定するための参照ポイントになります)。それぞれの分析計算は、パーティション内の現在のローに基づいて行われます。ローで表現されるウィンドウを使用して限定的な結果を生成するには、ユニークな順序付けの式を指定する必要があります。

どのウィンドウ・フレームでも、現在のローが参照ポイントになります。SQL/OLAP の構文には、ローベースのウィンドウ・フレームを、現在のローの前または後にある任意の数のロー (あるいは現在のローの前および後ろにある任意の数のロー) として定義するためのメカニズムが用意されています。

ウィンドウ・フレーム単位の代表的な例を次に示します。

- Rows Between Unbounded Preceding and Current Row — 各パーティションの先頭を開始ポイントとし、現在のローを終了ポイントとするウィンドウを指定します。累積和など、累積的な結果を計算するためのウィンドウを構築するときによく使用されます。
- Rows Between Unbounded Preceding and Unbounded Following — 現在のローに関係なく、パーティション全体についての固定ウィンドウを指定します。そのため、ウィンドウ集合関数の値は、パーティションのすべてのローで等しくなります。
- Rows Between 1 Preceding and 1 Following — 3 つの隣接するロー (現在のローとその前および後のロー) を含む固定サイズの移動ウィンドウを指定します。このウィンドウ・フレーム単位を使用して、たとえば 3 日間または 3 か月間の移動平均を計算できます。詳細については、図 2-3 (51 ページ) を参照してください。

ウィンドウ値にギャップがあると、ROWS を使用した場合に意味のない結果が生成されることがあるので注意してください。値セットが連続していない場合は、ROWS の代わりに RANGE を使用することを検討してください。RANGE に基づくウィンドウ定義では、重複する値を含んだ隣接ローが自動的に処理され、範囲内にギャップがあるときに他のローが含まれません。

注意 移動ウィンドウでは、入力の最初のローの前、および入力の最後のローの後ろには、NULL 値を含むローが存在することが想定されます。つまり、3 つのローから成る移動ウィンドウの場合は、入力の最後のローを現在のローとして計算するときに、直前のローと NULL 値が計算に含まれます。

- Rows Between Current Row and Current Row — ウィンドウを現在のローのみに制限します。
- Rows Between 1 Preceding and 1 Preceding — 現在のローの直前のローだけを含む単一ローのウィンドウを指定します。この指定を、現在のローのみに基づく値を計算する別のウィンドウ関数と組み合わせると、隣接するロー同士のデルタ (値の差分) を簡単に計算することができます。「隣接ロー間のデルタの計算」(58 ページ) を参照してください。

ローベースのウィンドウ・フレーム 図 2-4 の例では、ロー [1] ～ [5] は 1 つのパーティションを表しています。それぞれのローは、OLAP のウィンドウ・フレームが前にスライドするにつれて現在のローになります。このウィンドウ・フレームは Between Current Row And 2 Following として定義されているため、各フレームには、最大で 3 つ、最小で 1 つのローが含まれます。フレームがパーティションの終わりに到達したときは、現在のローだけがフレームに含まれます。網掛けの部分は、図 2-4 の各ステップでフレームから除外されているローを表しています。

図 2-4：ローベースのウィンドウ・フレーム

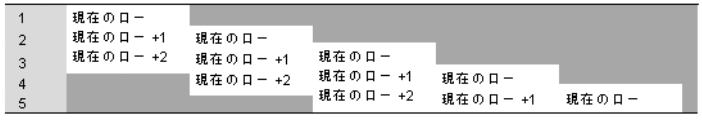
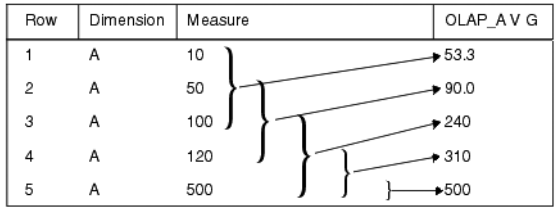


図 2-4 のウィンドウ・フレームは、次のような規則で機能しています。

- ロー [1] が現在のローであるときは、ロー [4] および [5] が除外される。

- ロー [2] が現在のローであるときは、ロー [5] および [1] が除外される。
- ロー [3] が現在のローであるときは、ロー [1] および [2] が除外される。
- ロー [4] が現在のローであるときは、ロー [1]、[2]、[3] が除外される。
- ロー [5] が現在のローであるときは、ロー [1]、[2]、[3]、[4] が除外される。

次の図では、この規則を具体的な値セットに適用し、OLAP の AVG 関数を使用して各ローの計算を行っています。スライド計算により、現在のローの位置に応じて、3 つまたはそれ以下のローを範囲として移動平均を算出しています。



次のクエリは、移動ウィンドウの定義の例を示しています。

```
SELECT dimension, measure,
   AVG(measure) OVER(partition BY dimension
      ORDER BY measure
      ROWS BETWEEN CURRENT ROW and 2 FOLLOWING)
      AS olap_avg
FROM ...
```

平均値は次のようにして計算されています。

- ロー [1] = (10 + 50 + 100)/3
- ロー [2] = (50+ 100 + 120)/3
- ロー [3] = (100 + 120 + 500)/3
- ロー [4] = (120 + 500 + NULL)/3
- ロー [5] = (500 + NULL + NULL)/3

結果セット内の以降のすべてのパーティション (たとえば B、C など) についても、同様の計算が実行されます。

現在のウィンドウにローが含まれていない場合、COUNT 以外のケースでは、結果は NULL になります。

RANGE

範囲ベースのウィンドウ・フレーム 前述の「ローベースのウィンドウ・フレーム」の例では、さまざまなローベースのウィンドウ・フレーム定義の中から 1 つを紹介しました。SQL/OLAP 構文では、別の種類のウィンドウ・フレームとして、物理的なローのシーケンスではなく、値ベース (または範囲ベース) のロー・セットに基づいて境界を定義する方法も用意されています。

値ベースのウィンドウ・フレームは、ウィンドウ・パーティション内で、特定の範囲の数値を含んでいるローを定義します。OLAP 関数の ORDER BY 句では、範囲指定を適用する数値カラムを定義します。このカラムの現在のローの値が、範囲指定の基準となります。範囲指定ではロー指定と同じ構文を使用しますが、構文の解釈の仕方は異なります。

ウィンドウ・フレーム単位 RANGE では、特定の順序付けカラムについて現在のローを基準とする値範囲を指定し、その範囲内の値を持つローを検索して、ウィンドウ・フレームに含めます。これは論理的なオフセットに基づくウィンドウ・フレームと呼ばれ、“3 preceding” などの定数を指定することも、評価結果が数値定数となる任意の式を指定することもできます。RANGE に基づくウィンドウを使用するときは、ORDER BY 句に数値式を 1 つだけ指定します。

注意 ORDER BY キーは、RANGE ウィンドウ・フレーム内の数値データであることが必要です。

たとえば、次のように指定すると、*year* カラムに現在のローの前後数年に当たる値を含んでいるロー・セットをフレームとして定義できます。

```
ORDER BY year ASC range BETWEEN CURRENT ROW and 1
PRECEDING
```

このクエリ例の 1 PRECEDING という部分は、現在のローの *year* 値から 1 を減算することを意味しています。

このような範囲指定は内包的です。現在のローの *year* 値が 2000 である場合は、ウィンドウ・パーティション内で、year 値が 2000 および 1999 であるすべてのローがこのフレームに含まれることになります。パーティション内での各ローの物理的な位置は問われません。値ベースのフレームでは、ローを含めたり除外したりする規則が、ローベースのフレームの規則とは大きく異なります (ローベースのフレームの規則は、ローの物理的なシーケンスに完全に依存しています)。

OLAP の AVG() 関数の例で考えてみます。次の部分的な結果セットは、値ベースのウィンドウ・フレームの概念を具体的に表しています。前述のように、このフレームには次のローが含まれます。

- 現在のローと同じ year 値を持つロー
- 現在のローから 1 を減算したのと同じ year 値を持つロー

Row	Dimension	Year	Measure	Olap_avg
1	A	1999	10000	10000
2	A	2001	5000	3000
3	A	2001	1000	3000
4	A	2002	12000	5250
5	A	2002	3000	5250

次のクエリは、範囲ベースのウィンドウ・フレーム定義の例を示しています。

```
SELECT dimension, year, measure,
   AVG(measure) OVER(PARTITION BY dimension
      ORDER BY year ASC
      range BETWEEN CURRENT ROW and 1 PRECEDING)
      as olap_avg
FROM ...
```

平均値は次のようにして計算されています。

- ロー [1] = 1999 のため、ロー [2] ～ [5] は除外。したがって AVG = 10,000/1
- ロー [2] = 2001 のため、ロー [1]、[4]、[5] は除外。したがって AVG = 6,000/2
- ロー [3] = 2001 のため、ロー [1]、[4]、[5] は除外。したがって AVG = 6,000/2
- ロー [4] = 2002 のため、ロー [1] は除外。したがって AVG = 21,000/4
- ロー [5] = 2002 のため、ロー [1] は除外。したがって AVG = 21,000/4

値ベースのフレームの昇順と降順　値ベースのウィンドウ・フレームを使用する OLAP 関数の ORDER BY 句では、範囲指定の対象となる数値カラムを特定するだけではなく、ORDER BY 値のソート順序も宣言できます。次の指定により、直前の部分のソート順序 (ASC または DESC) を設定できます。

```
RANGE BETWEEN CURRENT ROW AND n FOLLOWING
```

n FOLLOWING の指定には、次のような意味があります。

- パーティションがデフォルトの昇順 (ASC) でソートされている場合は、*n* は正の値として解釈されます。

- パーティションが降順 (DESC) でソートされている場合は、*n* は負の値として解釈されます。

たとえば、year カラムに 1999 ～ 2002 の 4 種類の値が含まれているとします。次のテーブルは、これらの値をデフォルトの昇順でソートした場合 (左側) と降順でソートした場合 (右側) を示しています。

ORDER BY year ASC	ORDER BY year DESC
1999	2002
2000	2001
2001	2000
2002	1999

現在のローが 1999 で、フレームが次のように指定されている場合、このフレームには値 1999 のローと値 1998 のロー (このテーブルには存在しません) が含まれます。

```
ORDER BY year DESC range BETWEEN CURRENT ROW and 1
FOLLOWING
```

注意 ORDER BY 値のソート順序は、値ベースのフレームに含まれるローの条件をテストするときに重要な要素です。フレームに含まれるか除外されるかは、数値だけでは決まりません。

無制限ウィンドウの使用 次のクエリでは、すべての製品と全製品の総数から成る結果セットが生成されます。

```
SELECT id, description, quantity,
    SUM(quantity) OVER () AS total
FROM products;
```

隣接ロー間のデルタの計算 現在のローと前のローをそれぞれ 1 つのウィンドウとして定義し、この 2 つのウィンドウを使用すると、隣接するロー間のデルタ (つまり差分) を直接的に計算することができます。

```
SELECT EmployeeID, Surname, SUM(salary) OVER (ORDER BY
BirthDate rows between current row and current row)

AS curr, SUM(Salary) OVER (ORDER BY BirthDate rows

between 1 preceding and 1 preceding) AS prev, (curr

-prev) as delta

FROM Employees WHERE State IN ('MA', 'AZ', 'CA', 'CO')
AND DepartmentID>10

ORDER BY EmployeeID, Surname;
```

このクエリの結果セットを次に示します。

EmployeeID	Surname	curr	prev	delta
148	Jordan	51432.000		
191	Bertrand	29800.000	39300.000	-9500.000
278	Melkisetian	48500.000	42300.000	6200.000
299	Overbey	39300.000	41700.750	-2400.750
318	Crow	41700.750	45000.000	-3299.250
586	Coleman	42300.000	46200.000	-3900.000
690	Poitras	46200.000	29800.000	16400.000
703	Martinez	55500.800	51432.000	4068.800
949	Savarino	72300.000	55500.800	16799.200
1101	Preston	37803.000	48500.000	-10697.000
1142	Clark	45000.000	72300.000	-27300.000

ここではウィンドウ関数 SUM() を使用していますが、ウィンドウの指定方法により、この合計には現在のローまたは前のローの salary 値だけが含まれています。また、結果セットの最初のローには前のローが存在しないため、最初のローの prev 値は NULL になります。したがって、delta も NULL になります。

ここまでの例では、OVER() 句と一緒に SUM() 集合関数を使用しました。

明示的なウィンドウ句とインラインのウィンドウ句

SQL OLAP では、クエリ内でウィンドウを指定する方法が 2 とおり用意されています。

- 明示的なウィンドウ句。HAVING 句の後でウィンドウを定義します。OLAP 関数を呼び出すときには、このようなウィンドウ句で定義したウィンドウを、ウィンドウの名前を指定して参照します。たとえば次のようにします。

  ```
  SUM ( ...) OVER w2
  ```

- インラインのウィンドウ指定。クエリ式の SELECT リスト内でウィンドウを定義します。これにより、HAVING 句の後のウィンドウ句でウィンドウを定義し、それをウィンドウ関数呼び出しから名前で参照するという方法に加えて、関数呼び出しと一緒にウィンドウを定義するという方法が可能になります。

注意 インラインのウィンドウ指定を使用する場合は、ウィンドウの名前を指定できません。1 つの SELECT リスト内で複数のウィンドウ関数呼び出しが同じウィンドウを使用する場合には、ウィンドウ句で定義した名前付きウィンドウを参照するか、インラインのウィンドウ定義を繰り返す必要があります。

ウィンドウ関数の例 ウィンドウ関数の例を次に示します。このクエリでは、データを部署別のパーティションに分け、在社年数が最も長い従業員を基点とした従業員の累積給与を計算して、結果セットを返します。この結果セットには、マサチューセッツ在住の従業員だけが含まれます。Sum_Salary カラムには、従業員の給与の累積和が含まれます。

```
SELECT DepartmentID, Surname, StartDate, Salary,
SUM(Salary) OVER (PARTITION BY DepartmentID ORDER BY
startdate rows between unbounded preceding and
current row) AS sum_salary
FROM Employees
WHERE State IN ('CA') AND DepartmentID IN (100, 200)
ORDER BY DepartmentID;
```

次の結果セットは部署別に分割されています。

DepartmentID	Surname	start_date	salary	sum_salary
200	Overbey	1987-02-19	39300.000	39300.000
200	Savarino	1989-11-07	72300.000	111600.000
200	Clark	1990-07-21	45000.000	156600.000

ランク付け関数

ランク付け関数を使用すると、データ・セットの値をランク付けされた順序のリストにまとめ、「今年度出荷された製品の中で売上合計が上位 10 位の製品名」または「15 社以上から受注した営業部員の上位 5%」といった質問に答えるクエリを 1 つの SQL 文で作成することができます。ランク付け関数には RANK()、DENSE_RANK()、PERCENT_RANK()、NTILE() などがあり、PARTITION BY 句と一緒に使用します。詳細については、「ランク付け関数」(60 ページ) を参照してください。

SQL/OLAP では、次の 4 つの関数がランク付け関数として分類されています。

```
<RANK FUNCTION TYPE> ::=
    RANK | DENSE RANK | PERCENT RANK | NTILE
```

ランク付け関数を使用すると、クエリで指定された順序に基づいて、結果セット内の各ローのランク値を計算することができます。たとえば販売マネージャが、営業成績が最高または最低の営業部員、販売成績が最高または最低の販売地域、あるいは売上が最高または最低の製品を調べたい場合があります。この情報はランク付け関数によって入手できます。

RANK() 関数

RANK 関数は、ORDER BY 句で指定されたカラムについて、ローのパーティション内での現在のローのランクを表す数値を返します。パーティション内の最初のローが 1 位となり、25 のローを含むパーティションでは、パーティション内の最後のローが 25 位となります。RANK は構文変換として指定されており、実際に RANK を同等の構文に変換することも、変換を行った場合に返すはずの値と同等の結果を返すこともできます。

次の例に出てくる ws1 は、w1 という名前のウィンドウを定義するウィンドウ指定を表しています。

```
RANK() OVER ws
```

これは次の指定に相当します。

```
( COUNT (*) OVER ( ws RANGE UNBOUNDED PRECEDING )
- COUNT (*) OVER ( ws RANGE CURRENT ROW ) + 1 )
```

この RANK 関数の変換では、論理的な集合 (RANGE) を使用しています。この結果、同位のロー (順序付けカラムに同じ値が含まれているロー) が複数ある場合は、それらに同じランクが割り当てられます。パーティション内で異なる値を持つ次のグループには、同位のローのランクよりも 1 以上大きいランクが割り当てられます。たとえば、順序付けカラムに 10、20、20、20、30 という値を含むローがある場合、1 つ目のローのランクは 1 になり、2 つ目のローのランクは 2 になります。3 つ目と 4 つ目のローのランクも 2 になりますが、5 つ目のローのランクは 5 になります。ランクが 3 または 4 のローは存在しません。このアルゴリズムは非連続型ランキング (sparse ranking) とも呼ばれます。

『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 4 章 SQL 関数」の「RANK 関数 [統計]」も参照してください。

DENSE_RANK() 関数

RANK 関数は同位のローがあるときに重複したランク値を割り当て非連続的なランキングを返しますが、DENSE_RANK 関数は抜けのないランキングを返します。同位のローに対しては同じように等しいランク値が割り当てられますが、このローのランクは、個々のローの順位ではなく、順序付けカラムに等しい値を含んでいるローの集まりの順位を表しています。RANK の例と同様に、順序付けカラムに 10、20、20、20、30 という値を含むローがある場合、1 つ目のローのランクは同じく 1 となり、2 つ目のローおよび 3 つ目、4 つ目のローのランクも同じく 2 となります。しかし、最後のローのランクは 5 ではなく 3 になります。

DENSE_RANK も、構文変換を通じて計算されます。

```
DENSE_RANK() OVER ws
```

これは次の指定に相当します。

```
COUNT ( DISTINCT ROW ( expr_1, . . ., expr_n ) )
  OVER ( ws RANGE UNBOUNDED PRECEDING )
```

この例では、*expr_1* から *expr_n* の部分が、ウィンドウ w1 のソート指定リストに含まれている値の式のリストを表しています。

『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 4 章 SQL 関数」の「DENSE_RANK 関数 [統計]」も参照してください。

PERCENT_RANK() 関数

PERCENT_RANK 関数は、個別の順位ではなく、パーセンテージでのランクを計算して、0 ～ 1 の小数値を返します。つまり、PERCENT_RANK が返すのはローの相対的なランクであり、この数値は、該当するウィンドウ・パーティション内での現在のローの相対位置を表します。たとえば、順序付けカラムの値がそれぞれ異なる 10 個のローがパーティションに含まれている場合、このパーティションの 3 つ目のローに対する PERCENT_RANK の値は 0.222... となります。パーティションの 1 つ目のローに続く 2/9 (22.222...%) のローをカバーしているためです。次の例に示すとおり、ローの PERCENT_RANK は、「ローの RANK - 1」を「パーティション内のローの数 - 1」で割ったものとして定義されています (“ANT”は、REAL や DOUBLE PRECISION などの概数値の型を表します)。

```
PERCENT_RANK() OVER ws
```

これは次の指定に相当します。

```
CASE
    WHEN COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
    PRECEDING AND UNBOUNDED FOLLOWING ) = 1
    THEN CAST (0 AS ANT)
    ELSE
      ( CAST ( RANK () OVER ( ws ) AS ANT ) -1 /
      ( COUNT (*) OVER ( ws RANGE BETWEEN UNBOUNDED
      PRECEDING AND UNBOUNDED FOLLOWING ) - 1 )
END
```

『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 4 章 SQL 関数」の「PERCENT_RANK 関数 [統計]」も参照してください。

ランク付けの例

ランク付けの例 1　次の SQL クエリでは、ユタ州在住の男性従業員と女性従業員を取得し、給与を基準として降順にランク付けしています。

```
SELECT Surname, Salary, Sex, RANK() OVER (ORDER BY
Salary DESC) AS Rank
FROM Employees WHERE State IN ('CA') AND DepartmentID
=200
ORDER BY Salary DESC;
```

このクエリの結果セットを次に示します。

Surname	salary	sex	rank
Scott	96300.000	M	1
Lull	87900.000	M	2
Pastor	74500.000	F	3
Shishov	72995.000	F	4
Samuels	37400.000	M	19

ランク付けの例 2　ランク付けの例 1 のクエリを基にして、データを性別のパーティションに分けることができます。次の例では、性別のパーティションに分けて、従業員の給与を降順にランク付けしています。

```
SELECT Surname, Salary, Sex, RANK() OVER (PARTITION
BY Sex ORDER BY Salary DESC) AS RANK
FROM Employees WHERE State IN ('CA', 'AZ') AND
DepartmentID
IN (200, 300)
ORDER BY Sex, Salary DESC;
```

このクエリの結果セットを次に示します。

Surname	salary	sex	rank
Savarino	72300.000	F	1
Clark	45000.000	F	2
Overbey	39300.000	M	3

ランク付けの例 3 この例では、カリフォルニアおよびテキサスの女性従業員のリストを、給与を基準として降順にランク付けしています。PERCENT_RANK 関数により、累積和が降順で示されます。

```
SELECT Surname, Salary, Sex, CAST(PERCENT_RANK() OVER
(ORDER BY Salary DESC) AS numeric (4, 2)) AS RANK
FROM Employees WHERE State IN ('CA', 'TX') AND Sex ='F'
ORDER BY Salary DESC;
```

このクエリの結果セットを次に示します。

Surname	salary	sex	percent
Savarino	72300.000	F	0.00
Smith	51411.000	F	0.33
Clark	45000.000	F	0.66
Garcia	39800.000	F	1.00

ランク付けの例 4 PERCENT_RANK 関数を使用して、データ・セットにおける上位または下位のパーセンタイルを調べることができます。次のクエリは、給与の額がデータ・セットの上位 5% に入る男性従業員を返します。

```
SELECT * FROM (SELECT Surname, Salary, Sex,
CAST(PERCENT_RANK() OVER (ORDER BY salary DESC) as
numeric (4, 2)) AS percent
FROM Employees WHERE State IN ('CA') AND sex ='F' ) AS
DT where percent > 0.5
ORDER BY Salary DESC;
```

このクエリの結果セットを次に示します。

Surname	salary	sex	percent
Clark	45000.000	F	1.00

ウィンドウ集合関数

ウィンドウ集合関数を使用すると、複数のレベルの集合を 1 つのクエリで計算できます。たとえば、支出が平均より少ない四半期をすべて列挙することができます。集合関数 (単純な集合関数 AVG、COUNT、MAX、MIN、SUM を含む) を使用すると、1 つの文の中でさまざまなレベルで計算した結果を 1 つのローに書き出すことができます。これにより、ジョインや相関サブクエリを使用しなくても、集合値をグループ内のディテール・ローと比較することができます。

これらの関数を使用して、非集合値と集合値を比較することも可能です。たとえば、営業部員が特定の年にある製品に対して平均以上の注文を出した顧客の一覧を作成したり、販売マネージャが従業員の給与をその部署の平均給与と比較したりすることが考えられます。

SELECT 文の中で DISTINCT が指定されている場合は、ウィンドウ演算子の後に DISTINCT 操作が適用されます。ウィンドウ演算子は、GROUP BY 句が処理された後、SELECT リストの項目やクエリの ORDER BY 句が評価される前に計算されます。

ウィンドウ集合関数の例 1　次のクエリは、平均販売数よりも多く売れた製品の一覧を年別に示す結果セットを返します。

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
CAST(AVG(Sal) OVER(PARTITION BY DepartmentID) AS
numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
OVER(PARTITION BY DepartmentID) AS numeric(10,2)) AS
STD_DEV
FROM Employees
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
Sal> (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

このクエリの結果セットを次に示します。

```
 E_name     Dept      Sal     Average   STD_DEV
--------  -----  --------  -------  --------
Lull       100   87900.00  58736.28  16829.59
Sheffield  100   87900.00  58736.28  16829.59
Scott      100   96300.00  58736.28  16829.59
```

```
Sterling    200   64900.00  48390.94  13869.59
Savarino    200   72300.00  48390.94  13869.59
Kelly       200   87500.00  48390.94  13869.59
Shea        300   138948.00 59500.00  30752.39
Blaikie     400   54900.00  43640.67  11194.02
Morris      400   61300.00  43640.67  11194.02
Evans       400   68940.00  43640.67  11194.02
Martinez    500   55500.80  33752.20   9084.49
```

2000 年の平均注文数は 1,787 であり、4 つの製品 (700、601、600、400) が平均を上回っています。2001 年の平均注文数は 1,048 であり、3 つの製品が平均を上回っています。

ウィンドウ集合関数の例 2 次のクエリは、給与の額がそれぞれの部署の平均給与よりも 1 標準偏差以上高い従業員を表す結果セットを返します。標準偏差とは、そのデータが平均からどのぐらい離れているかを示す尺度です。

```
SELECT * FROM (SELECT Surname AS E_name, DepartmentID AS
    Dept, CAST(Salary AS numeric(10,2) ) AS Sal,
    CAST(AVG(Sal) OVER(PARTITION BY dept) AS
    numeric(10, 2)) AS Average, CAST(STDDEV_POP(Sal)
    OVER(PARTITION BY dept) AS numeric(10,2)) AS
    STD_DEV
FROM Employees
GROUP BY Dept, E_name, Sal) AS derived_table WHERE
    Sal> (Average+STD_DEV )
ORDER BY Dept, Sal, E_name;
```

次の結果から、どの部署にも、給与の額が平均を大きく上回っている従業員が 1 人以上いることがわかります。

```
E_name       Dept        Sal     Average      STD_DEV
--------     ----    --------    --------     --------
Lull          100    87900.00    58736.28     16829.59
Sheffield     100    87900.00    58736.28     16829.59
Scott         100    96300.00    58736.28     16829.59
Sterling      200    64900.00    48390.94     13869.59
Savarino      200    72300.00    48390.94     13869.59
Kelly         200    87500.00    48390.94     13869.59
Shea          300   138948.00    59500.00     30752.39
Blaikie       400    54900.00    43640.67     11194.02
Morris        400    61300.00    43640.67     11194.02
```

```
Evans          400    68940.00    43640.67     11194.02
Martinez       500    55500.80    33752.20      9084.49
```

従業員 Scott の給与は 96,300.00 ドルで、所属部署の平均給与は 58,736.28 ドルです。この部署の標準偏差は 16,829.00 なので、給与の額が 75,565.88 ドル (58736.28 + 16829.60 = 75565.88) 未満ならば、平均の 1 標準偏差以内の範囲に収まります。

統計集合関数

ANSI SQL/OLAP 拡張機能には、数値データの統計的分析を行うための集合関数がこの他にも数多く用意されています。これには、分散、標準偏差、相関、直線回帰を計算するための関数も含まれます。

標準偏差と分散

SQL/OLAP の一般的な関数の中には、次の構文の太字で表示されている部分のように、1 つの引数を取る関数があります。

```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
   <BASIC AGGREGATE FUNCTION TYPE>
   | STDDEV | STDDEV_POP | STDDEV_SAMP
   | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

- STDDEV_POP — グループまたはパーティションの各ロー (DISTINCT が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される「値の式」についての母標準偏差を計算します。これは、母分散の平方根として定義されます。
- STDDEV_SAMP — グループまたはパーティションの各ロー (DISTINCT が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される「値の式」についての母標準偏差を計算します。これは、標本分散の平方根として定義されます。
- VAR_POP — グループまたはパーティションの各ロー (DISTINCT が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される「値の式」についての母分散を計算します。これは、「値の式」と「値の式の平均」との差の 2 乗和をグループまたはパーティション内の残りのローの数で割った値として定義されます。
- VAR_SAMP — グループまたはパーティションの各ロー (DISTINCT が指定されている場合は、重複が削除された後に残る各ロー) に対して評価される「値の式」の標本分散を計算します。これは、「値の式」の差の 2 乗和を、グループまたはパーティション内の残りのローの数より 1 少ない数で割った値として定義されます。

これらの関数と STDDEV および VARIANCE 関数は、クエリの ORDER BY 句の指定に従ってローのパーティションについての値を計算できる集合関数です。MAX や MIN などのその他の基本的な集合関数と同様に、これらの関数は入力データ内の NULL 値を無視します。また、分析される式のドメインに関係なく、分散と標準偏差の計算では必ず IEEE の倍精度浮動小数点数が使用されます。分散関数または標準偏差関数への入力が空のデータ・セットである場合、これらの関数は結果として NULL を返します。VAR_SAMP 関数は 1 つのローに対して計算を行うと NULL を返しますが、VAR_POP は値 0 を返します。

相関

相関係数を計算する SQL/OLAP 関数は、次のとおりです。

- CORR — 数値のペアのセットの相関係数を返します。

CORR 関数は、ウィンドウ集合関数 (ウィンドウ名または指定に対するウィンドウ関数の種類を指定する関数) としても、OVER 句のない単純な集合関数としても使用できます。

共分散

共分散を計算する SQL/OLAP 関数は、次のとおりです。

- COVAR_POP — 数値のペアのセットの母共分散を返します。
- COVAR_SAMP — 数値のペアのセットの標本共分散を返します。

共分散関数は、式 1 または式 2 が null 値を持つ、すべてのペアを除外します。

共分散関数は、ウィンドウ集合関数 (ウィンドウ名または指定に対するウィンドウ関数の種類を指定する関数) としても、OVER 句のない単純な集合関数としても使用できます。

累積分布

ローのグループの 1 つの値の相対位置を計算する SQL/OLAP 関数は、CUME_DIST です。

ウィンドウ指定には ORDER_BY 句が含まれていることが必要です。

CUME_DIST 関数では、複合ソート・キーは許可されません。

回帰分析

回帰分析関数は、直線回帰の方程式を使用し、独立変数と従属変数との間の関係を計算します。SQL/OLAP の直線回帰関数は、次のとおりです。

- REGR_AVGX — 回帰直線の独立変数の平均を計算します。
- REGR_AVGY — 回帰直線の従属変数の平均を計算します。
- REGR_COUNT — 回帰直線の調整に使用される null 値以外のペアの数を表す整数を返します。
- REGR_INTERCEPT — 従属変数と独立変数に最適な回帰直線の y 切片を計算します。

- REGR_R2 — 回帰直線の決定の決定係数 (適合度の統計情報) を計算します。
- REGR_SLOPE — null 以外のペアに適合する直線回帰の傾きを計算します。
- REGR_SXX — 直線回帰モデルに使用される独立した式の 2 乗和の合計を返します。この関数は、回帰モデルの統計的な有効性を評価するときに使用できます。
- REGR_SXY — 従属変数と独立変数の結果の合計を返します。この関数は、回帰モデルの統計的な有効性を評価するときに使用できます。
- REGR_SYY — 回帰モデルの統計的有効性を評価できる値を返します。

回帰分析関数は、ウィンドウ集合関数 (ウィンドウ名または指定に対するウィンドウ関数の種類を指定する関数) としても、OVER 句のない単純な集合関数としても使用できます。

OLAP の加重集合関数

OLAP の加重集合関数は、加重移動平均を計算します。

- EXP_WEIGHTED_AVG — 指数関数的に加重された移動平均を計算します。加重は、平均を構成する各数量の相対的な重要性を決定します。EXP_WEIGHTED_AVG の加重は、指数関数的に減少します。指数関数的な加重は、最新の値に加重を適用し、加重を適用しながらも、古い値の加重を減少します。
- WEIGHTED_AVG — 時間経過とともに等差階級的に加重が減少した、直線的加重移動平均を計算します。加重は、最新のデータ・ポイントの最高値から減少し、最も古いデータ・ポイントでゼロになります。

ウィンドウ指定には ORDER_BY 句が含まれていることが必要です。

データベース業界の標準外の拡張機能

データベース業界で使用される ANSI 以外の SQL/OLAP 集合関数の拡張機能には、FIRST_VALUE、MEDIAN、および LAST_VALUE があります。

- FIRST_VALUE — 値のセットから最初の値を返します。
- MEDIAN — 式から中央値を返します。
- LAST_VALUE — 値のセットから最後の値を返します。

FIRST_VALUE および LAST_VALUE 関数には、ウィンドウ指定が必要です。MEDIAN 関数は、ウィンドウ集合関数 (ウィンドウ名または指定に対するウィンドウ関数の種類を指定する関数) としても、OVER 句のない単純な集合関数としても使用できます。

分散統計関数

SQL/OLAP には、順序付きセットを取り扱う関数がいくつか定義されています。PERCENTILE_CONT、PERCENTILE_DISC という 2 つの逆分散統計関数があります。これらの統計関数は、パーセンタイル値を引数として受け取り、WITHIN GROUP 句で指定されたデータのグループまたはデータ・セット全体に対して処理を行います。

これらの関数は、グループごとに 1 つの値を返します。PERCENTILE_DISC (不連続) では、結果のデータ型は WITHIN GROUP 句に指定した ORDER BY の項目のデータ型と同じになります。PERCENTILE_CONT (連続) では、結果のデータ型は、numeric (WITHIN GROUP 句の ORDER BY 項目が numeric の場合) または double (ORDER BY 項目が整数または浮動小数点の場合) となります。

逆分散統計関数では、WITHIN GROUP (ORDER BY) 句を指定する必要があります。次に例を示します。

PERCENTILE_CONT (*expression1*)
WITHIN GROUP (**ORDER BY** *expression2* [**ASC** | **DESC**])

expression1 の値には、numeric データ型の定数を、0 以上 1 以下の範囲で指定します。引数が NULL であれば、“wrong argument for percentile” エラーが返ります。引数の値が 0 よりも小さいか、1 よりも大きい場合は、“data value out of range” エラーが返されます。

必須の ORDER BY 句には、パーセンタイル関数の実行の対象となる式と、各グループ内でのローのソート順を指定します。この ORDER BY 句は、WITHIN GROUP 句の内部でのみ使用するもので、SELECT 文の ORDER BY とは異なります。

WITHIN GROUP 句は、クエリ結果を順序付けられたデータ・セットに分類します。関数はこのデータ・セットに基づいて結果を計算します。

expression2 には、カラム参照を含む 1 つの式でソートを指定します。このソート式に、複数の式やランク付け統計関数、set 関数、またはサブクエリを指定することはできません。

ASC と DESC のパラメータでは、昇順または降順の順序付けシーケンスを指定します。昇順がデフォルトです。

逆分散統計関数は、サブクエリ、HAVING 句、ビュー、union で使用することが可能です。逆分散統計関数は、統計を行わない単純な集合関数が使用されるところであれば、どこでも使用できます。逆分散統計関数は、データ・セット内の NULL 値を無視します。

PERCENTILE_CONT 例　この例では、PERCENTILE_CONT 関数を使用して、各地域の自動車販売の 10 番目のパーセンタイル値を求めます。次のようなデータ・セットを使用します。

sales	region	dealer_name
900	Northeast	Boston
800	Northeast	Worcester
800	Northeast	Providence
700	Northeast	Lowell
540	Northeast	Natick
500	Northeast	New Haven
450	Northeast	Hartford
800	Northwest	SF
600	Northwest	Seattle
500	Northwest	Portland
400	Northwest	Dublin
500	South	Houston
400	South	Austin
300	South	Dallas
200	South	Dover

次のクエリ例では、SELECT 文に PERCENTILE_CONT 関数を含めています。

```
SELECT region, PERCENTILE_CONT(0.1)
WITHIN GROUP ( ORDER BY ProductID DESC )
FROM ViewSalesOrdersSales GROUP BY region;
```

この SELECT 文の結果には、各地域の自動車販売の 10 番目のパーセンタイル値が一覧表示されます。

region	percentile_cont
Canada	601.0
Central	700.0
Eastern	700.0
South	700.0
Western	700.0

PERCENTILE_DISC 例　この例では、PERCENTILE_DISC 関数を使用して、各地域の自動車販売の 10 番目のパーセンタイル値を求めます。次のようなデータ・セットを使用します。

sales	region	dealer_name
900	Northeast	Boston
800	Northeast	Worcester
800	Northeast	Providence
700	Northeast	Lowell
540	Northeast	Natick
500	Northeast	New Haven
450	Northeast	Hartford
800	Northwest	SF
600	Northwest	Seattle
500	Northwest	Portland
400	Northwest	Dublin
500	South	Houston
400	South	Austin
300	South	Dallas
200	South	Dover

次のクエリでは、SELECT 文に PERCENTILE_DISC 関数を含めています。

```
SELECT region, PERCENTILE_DISC(0.1) WITHIN GROUP
   (ORDER BY sales DESC )
FROM carSales GROUP BY region;
```

この SELECT 文の結果には、各地域の自動車販売の 10 番目のパーセンタイル値が一覧表示されます。

region	percentile_cont
Northeast	900
Northwest	800
South	500

分散統計関数の詳細については、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 4 章 SQL 関数」の「PERCENTILE_CONT 関数 [統計]」と「PERCENTILE_DISC 関数 [統計]」を参照してください。

数値関数

Sybase IQ でサポートされる OLAP 数値関数には、CEILING (エイリアスは CEIL)、EXP (エイリアスは EXPONENTIAL)、FLOOR、LN (エイリアスは LOG)、SQRT、WIDTH_BUCKET があります。

```
<numeric value function> :: =
  <natural logarithm>
|  <exponential function>
|  <power function>
|  <square root>
|  <floor function>
|  <ceiling function>
|  <width bucket function>
```

サポートされる数値関数の構文を表 2-3 に示します。

表 2-3：数値関数の構文

数値関数	構文
自然対数	LN (*numeric-expression*)
指数関数	EXP (*numeric-expression*)
累乗関数	POWER (*numeric-expression1*, *numeric-expression2*)
平方根	SQRT (*numeric-expression*)
床関数	FLOOR (*numeric-expression*)
天井関数	CEILING (*numeric-expression*)
等幅ヒストグラム作成関数	WIDTH_BUCKET (*expression*, *min_value*, *max_value*, *num_buckets*)

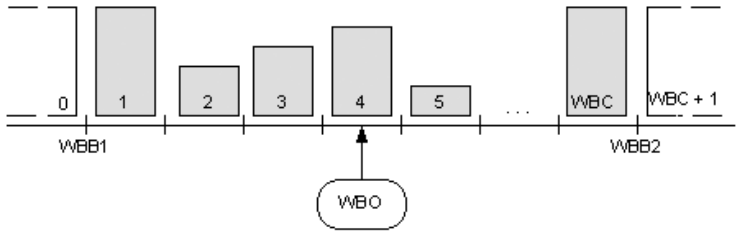
それぞれの数値関数の機能は次のとおりです。

- LN — 引数値の自然対数を返します。引数値がゼロまたは負の場合は、エラー状態が発生します。LN は LOG の同意語です。
- EXP — *e* (自然対数の底) の値を、引数値で指定された指数まで累乗した結果を返します。
- POWER — 1 つ目の引数値を、2 つ目の引数値で指定された指数まで累乗した結果を返します。両方の引数の値が 0 の場合は、1 が返されます。1 つ目の引数が 0 で、2 つ目の引数が正の値である場合は、0 が返されます。1 つ目の引数が 0 で、2 つ目の引数が負の値である場合は、例外が発生します。1 つ目の引数が負の値で、2 つ目の引数が整数でない場合は、例外が発生します。
- SQRT — 引数値の平方根を返します。これは、“POWER (*expression*, *0.5*)” の構文変換です。
- FLOOR — 引数の値以下で、正の無限大に最も近い整数値を返します。
- CEILING — 引数の値以上で、負の無限大に最も近い整数値を返します。CEIL は CEILING の同意語です。

WIDTH_BUCKET 関数

The WIDTH_BUCKET 関数は、他の数値関数よりも少し複雑です。この関数は 4 つの引数を取ります。具体的には、「目的の値」、2 つの範囲境界、そしてこの範囲を何個の等しいサイズ (または可能な限り等しいサイズ) の「バケット」に分割するかを指定します。WIDTH_BUCKET 関数は、範囲の上限から下限までの差のパーセンテージに基づき、目的の値が何番目のバケットに含まれるかを示す数値を返します。最初のバケットが、バケット番号 1 となります。

目的の値が範囲境界の外にある場合のエラーを避けるために、範囲の下限よりも小さい目的の値は、先頭の補助バケット (バケット 0) に配置されます。同様に、範囲の上限よりも大きい目的の値は、末尾の補助バケット (バケット N+1) に配置されます。



たとえば、WIDTH_BUCKET (*14, 5, 30, 5*) は 2 を返します。処理の内容は次のとおりです。

- (30-5)/5 = 5 なので、指定の範囲を 5 つのバケットに分割すると、各バケットの幅は 5 になります。
- 1 つ目のバケットは 0.00 ～ 19.999 …% の値、2 つ目のバケットは 20.00 ～ 39.999 …% の値を表し、以降同様に続き、5 つ目のバケットは 80.00 ～ 100.00% の値を表します。
- 目的の値を含むバケットは、(5*(14-5)/(30-5)) + 1 という計算によって算出されます。これは、バケットの総数に、指定範囲に対する「下限から目的の値までのオフセット」の比率を掛け、それに 1 を足すという計算です。実際の数式は (5*9/25) + 1 となり、これを計算すると 2.8 になります。これはバケット番号 2 (2.0 ～ 2.999 …) の範囲に含まれる値であるため、バケット番号 2 が返されます。

WIDTH_BUCKET 例

次の例では、サンプル・テーブル内のマサチューセッツ州の顧客の credit_limit カラムに 10 のバケット・ヒストグラムを作成し、各顧客のバケット数 (“Credit Group”) を返します。最大値を超える限度額が設定されている顧客は、オーバフロー・バケット 11 に割り当てられます。

注意　これは説明用の例であり、iqdemo データベースから生成したものではありません。

```
SELECT customer_id, cust_last_name, credit_limit,
   WIDTH_BUCKET(credit_limit, 100, 5000, 10) "Credit
   Group"
   FROM customers WHERE territory = 'MA'
   ORDER BY "Credit Group";
```

CUSTOMER_ID	CUST_LAST_NAME	CREDIT_LIMIT	Credit Group
825	Dreyfuss	500	1
826	Barkin	500	1
853	Palin	400	1
827	Siegel	500	1
843	Oates	700	2
844	Julius	700	2
835	Eastwood	1200	3
840	Elliott	1400	3
842	Stern	1400	3
841	Boyer	1400	3
837	Stanton	1200	3
836	Berenger	1200	3
848	Olmos	1800	4
847	Streep	5000	11

範囲が逆の場合、バケットはオープン・クローズ間隔になります。次に例を示します。WIDTH_BUCKET (*credit_limit*, *5000*, *0*, *5*)。この例では、バケット番号 1 は (4000, 5000]、バケット番号 2 は (3000, 4000]、およびバケット番号 5 は (0, 1000] です。オーバフロー・バケットには 0 (5000, +infinity) の番号が付き、アンダフロー・バケットには 6 (-infinity, 0] の番号が付きます。

参照

BIT_LENGTH 関数 [文字列]、EXP 関数 [数値]、FLOOR 関数 [数値]、POWER 関数 [数値]『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 4 章 SQL 関数」の「SQRT 関数 [数値]」および「WIDTH_BUCKET 関数 [数値]」を参照してください。

OLAP の規則と制限

OLAP 関数を使用できる場合

SQL クエリ内では、次の条件下で OLAP 関数を使用できます。

- SELECT リストの中
- 式の中
- スカラ関数の引数として
- 最後の ORDER BY 句の中 (クエリ内のどこかで定義されている OLAP 関数のエイリアスまたは位置参照を使用)

OLAP 関数を使用できない場合

OLAP 関数は、次の条件下では使用できません。

- サブクエリの中
- WHERE 句の検索条件の中
- SET (集合) 関数の引数として。たとえば次の式は無効です。

```
SUM(RANK() OVER(ORDER BY dollars))
```

- ウィンドウ集合を、他の集合に対する引数として使用することはできません (ただし、内側の集合がビューまたは抽出テーブル内で生成されたものである場合は例外です)。ランク付け関数についても同じことが言えます。
- ウィンドウ集合関数と RANK 関数は、HAVING 句の中では使用できません。
- ウィンドウ集合関数に DISTINCT を指定することはできません。
- ウィンドウ関数を他のウィンドウ関数の内部にネストすることはできません。
- 逆分散統計関数は、OVER 句ではサポートされていません。
- ウィンドウ定義句では外部参照を使用できません。
- OLAP 関数内での相関参照は認められていますが、相関があるカラムのエイリアスは認められていません。

OLAP 関数から参照するカラムは、その OLAP 関数と GROUP BY 句が含まれている同じクエリ・ブロック内のグループ化カラムまたは集合関数でなければなりません。OLAP の処理は、グループ化操作と集合操作の後、最後の ORDER BY 句が適用される前に行われます。そのため、中間の結果セットから OLAP 式を導出することも可能です。クエリ・ブロック内に GROUP BY 句がない場合、OLAP 関数は select リスト内の他のカラムを参照することができます。

Sybase IQ の制限事項

Sybase IQ で SQL OLAP 関数を使用するときの制限事項を次に示します。

- ウィンドウ・フレーム定義の中でユーザ定義関数を使用することはできません。
- ウィンドウ・フレーム定義で使用する定数は符号なし数値でなければならず、最大値 BIG INT 2^{63-1} を超えてはなりません。
- ウィンドウ集合関数と RANK 関数は、DELETE および UPDATE 文では使用できません。
- ウィンドウ集合関数と RANK 関数は、サブクエリ内では使用できません。
- CUME_DIST は、現時点ではサポートされていません。
- グループ化セットは、現時点ではサポートされていません。
- 相関関数と直線回帰関数は、現時点ではサポートされていません。

その他の OLAP の例

この項では、OLAP 関数を使用したその他の例を紹介します。

ウィンドウの開始ポイントと終了ポイントは、中間の結果ローが処理されるときに変化する可能性があります。たとえば、累積和を計算する場合には、ウィンドウの開始ポイントは各パーティションの最初のローに固定されますが、終了ポイントは現在のローを含めるためにパーティション内のローを移動していきます。詳細については、図 2-3 (51 ページ) を参照してください。

また、ウィンドウの開始ポイントと終了ポイントの両方が可変だが、パーティション全体のローの数は一定であるという例も考えられます。このようなウィンドウを使用すると移動平均を計算するクエリを作成でき、たとえば 3 日間の株価の移動平均を返す SQL クエリを作成できます。

例：クエリ内でのウィンドウ関数

次のクエリは、2005 年の 7 月と 8 月に出荷された全製品と、出荷日別の累積出荷数を一覧にして示します。

```
SELECT p.id, p.description, s.quantity, s.shipdate,
```

```
SUM(s.quantity) OVER (PARTITION BY productid ORDER BY
s.shipdate rows between unbounded preceding and
current row)
FROM SalesOrderItems s JOIN Products p on
(s.ProductID =
p.id) WHERE s.ShipDate BETWEEN '2001-05-01' and
'2001-08-31' AND s.quantity > 40
ORDER BY p.id;
```

このクエリの結果セットを次に示します。

ID	description	quantity	ship_date	sum quantity
302	Crew Neck	60	2001-07-02	60
400	Cotton Cap	60	2001-05-26	60
400	Cotton Cap	48	2001-07-05	108
401	Wool cap	48	2001-06-02	48
401	Wool cap	60	2001-06-30	108
401	Wool cap	48	2001-07-09	156
500	Cloth Visor	48	2001-06-21	48
501	Plastic Visor	60	2001-05-03	60
501	Plastic Visor	48	2001-05-18	108
501	Plastic Visor	48	2001-05-25	156
501	Plastic Visor	60	2001-07-07	216
601	Zipped Sweatshirt	60	2001-07-19	60
700	Cotton Shorts	72	2001-05-18	72
700	Cotton Shorts	48	2001-05-31	120

この例では、2 つのテーブルのジョインとクエリの WHERE 句を適用した後に、SUM ウィンドウ関数の計算が行われます。このクエリではインラインのウィンドウ指定を使用しており、このウィンドウ指定によって、ジョインからの入力ローが次のように処理されています。

1 prod_id 属性の値に基づいて入力ローがパーティション (グループ) に分けられます。

2 各パーティション内で、ローが ship_date 属性に基づいてソートされます。

3 パーティション内の各ローの quantity 属性について、SUM() 関数が評価されます。その際に、ソート後の各パーティションの最初のローから現在のローまでを含む移動ウィンドウ (現在のローも含む) が使用されます。図 2-3 (51 ページ) を参照してください。

このクエリを別の方法で記述するには、関数の外でウィンドウを定義し、そのウィンドウを関数呼び出しから参照します。この方法は、同じウィンドウに基づくウィンドウ関数を複数指定する場合に便利です。このウィンドウ関数を使用するクエリを、独立したウィンドウ句を使用する方法で記述すると次のようになります (cumulative というウィンドウを宣言しています)。

```
SELECT p.id, p.description, s.quantity, s.shipdate,
SUM(s.quantity) OVER(cumulative ROWS BETWEEN UNBOUNDED
PRECEDING and CURRENT ROW ) cumulative

FROM SalesOrderItems s JOIN Products p On (s.ProductID
=p.id)

WHERE s.shipdate BETWEEN '2001-07-01' and '2001-08-31'

Window cumulative as (PARTITION BY s.productid ORDER BY
s.shipdate)

ORDER BY p.id;
```

このクエリ指定では、ウィンドウ句が ORDER BY の前にあります。ウィンドウ句を使用するときには、次の制限が適用されます。

- インラインのウィンドウ指定に PARTITION BY 句を含めることはできません。
- ウィンドウ句で指定されるウィンドウにウィンドウ・フレーム句を含めることはできません。たとえば、「文法規則 32」(90 ページ) に次のように記述されています。

  ```
  <WINDOW FRAME CLAUSE> ::=
     <WINDOW FRAME UNIT>
     <WINDOW FRAME EXTENT>
  ```

- インラインのウィンドウ指定にもウィンドウ句のウィンドウ指定にもウィンドウ順序句を含めることができますが、両方に含めることはできません。たとえば、「文法規則 31」(90 ページ) に次のように記述されています。

  ```
  <WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
  ```

例：複数の関数で使用されるウィンドウ

1 つの名前付きウィンドウを定義しておき、そのウィンドウに基づいて複数の関数を計算することもできます。具体的な例を次に示します。

```
SELECT p.ID, p.Description, s.quantity, s.ShipDate,
```

```
SUM(s.Quantity) OVER ws1, MIN(s.quantity) OVER ws1
FROM SalesOrderItems s JOIN Products p ON (s.ProductID =
p.ID) WHERE s.ShipDate BETWEEN '2000-01-09' AND
'2000-01-17' AND s.Quantity > 40 window ws1 AS
(PARTITION BY productid ORDER BY shipdate rows
between unbounded preceding and current row)
ORDER BY p.id;
```

このクエリの結果セットを次に示します。

ID	description	quantity	ship_date	sum	min
400	Cotton Cap	48	2000-01-09	48	48
401	Wool cap	48	2000-01-09	48	48
500	Cloth Visor	60	2000-01-14	60	60
500	Cloth Visor	60	2000-01-15	120	60
501	Plastic Visor	60	2000-01-14	60	60

例：累積和の計算

このクエリでは、ORDER BY start_date の順序に従って、部署別の給与の累積和を計算します。

```
SELECT DepartmentID, start_date, name, salary,
   SUM(salary) OVER (PARTITION BY DepartmentID ORDER BY
   start_date ROWS BETWEEN UNBOUNDED PRECEDING AND
   CURRENT ROW)
FROM emp1
ORDER BY DepartmentID, start_date;
```

このクエリの結果セットを次に示します。

DepartmentID	start_date	name	salary	sum (salary)
100	1996-01-01	Anna	18000	18000
100	1997-01-01	Mike	28000	46000
100	1998-01-01	Scott	29000	75000
100	1998-02-01	Antonia	22000	97000
100	1998-03-12	Adam	25000	122000
100	1998-12-01	Amy	18000	140000
200	1998-01-01	Jeff	18000	18000
200	1998-01-20	Tim	29000	47000

```
200          1998-02-01    Jim       22000    69000
200          1999-01-10    Tom       28000    97000
300          1998-03-12    Sandy     55000    55000
300          1998-12-01    Lisa      38000    93000
300          1999-01-10    Peter     48000    141000
```

例：移動平均の計算

このクエリでは、連続する 3 か月間の売上の移動平均を計算します。使用するウィンドウ・フレームは 3 つのローから成り、先行する 2 つのローと現在のローが含まれます。このウィンドウは、パーティションの最初から最後までスライドしていきます。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num ROWS
   BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

このクエリの結果セットを次に示します。

prod_id	month_num	sales	avg(sales)
10	1	100	100.00
10	2	120	110.00
10	3	100	106.66
10	4	130	116.66
10	5	120	116.66
10	6	110	120.00
20	1	20	20.00
20	2	30	25.00
20	3	25	25.00
20	4	30	28.33
20	5	31	28.66
20	6	20	27.00
30	1	10	10.00
30	2	11	10.50
30	3	12	11.00
30	4	1	8.00

例：ORDER BY の結果

この例では、クエリの最上位の ORDER BY 句がウィンドウ関数の最終的な結果に適用されます。ウィンドウ句に指定されている ORDER BY は、ウィンドウ関数の入力データに適用されます。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num ROWS
   BETWEEN 2 PRECEDING AND CURRENT ROW)
FROM sale WHERE rep_id = 1
ORDER BY prod_id desc, month_num;
```

このクエリの結果セットを次に示します。

prod_id	month_num	sales	avg(sales)
30	1	10	10.00
30	2	11	10.50
30	3	12	11.00
30	4	1	8.00
20	1	20	20.00
20	2	30	25.00
20	3	25	25.00
20	4	30	28.33
20	5	31	28.66
20	6	20	27.00
10	1	100	100.00
10	2	120	110.00
10	3	100	106.66
10	4	130	116.66
10	5	120	116.66
10	6	110	120.00

例：1 つのクエリ内で複数の集合関数を使用

この例では、1 つのクエリ内で、異なるウィンドウに対して 2 種類の集合関数を実行しています。

```
SELECT prod_id, month_num, sales, AVG(sales) OVER
   (WS1 ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS
   CAvg, SUM(sales) OVER(WS1 ROWS BETWEEN UNBOUNDED
   PRECEDING AND CURRENT ROW) AS CSum
FROM sale WHERE rep_id = 1 WINDOW WS1 AS (PARTITION BY
   prod_id
ORDER BY month_num)
ORDER BY prod_id, month_num;
```

このクエリの結果セットを次に示します。

prod_id	month_num	sales	CAvg	CSum
10	1	100	110.00	100
10	2	120	106.66	220
10	3	100	116.66	320
10	4	130	116.66	450
10	5	120	120.00	570
10	6	110	115.00	680
20	1	20	25.00	20
20	2	30	25.00	50
20	3	25	28.33	75
20	4	30	28.66	105
20	5	31	27.00	136
20	6	20	25.50	156
30	1	10	10.50	10
30	2	11	11.00	21
30	3	12	8.00	33
30	4	1	6.50	34

例：ウィンドウ・フレーム指定の ROWS と RANGE の比較

このクエリでは、ROWS と RANGE を比較しています。ORDER BY 句の指定により、このデータには重複するローが含まれています。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
    (ws1 RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) AS
       Range_sum, SUM(sales) OVER
    (ws1 ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) AS
       Row_sum
FROM sale window ws1 AS (PARTITION BY prod_id ORDER BY
   month_num)
ORDER BY prod_id, month_num;
```

このクエリの結果セットを次に示します。

prod_id	month_num	sales	Range_sum	Row_sum
10	1	100	250	100
10	1	150	250	250
10	2	120	370	370
10	3	100	470	370
10	4	130	350	350
10	5	120	381	350
10	5	31	381	281

```
10              6     110         391        261
20              1      20          20         20
20              2      30          50         50
20              3      25          75         75
20              4      30          85         85
20              5      31          86         86
20              6      20          81         81
30              1      10          10         10
30              2      11          21         21
30              3      12          33         33
30              4       1          25         24
30              4       1          25         14
```

例：現在のローを除外するウィンドウ・フレーム

この例では、現在のローを除外するウィンドウ・フレームを定義しています。このクエリは、現在のローを除く 4 つのローの合計を計算します。

```
SELECT prod_id, month_num, sales, sum(sales) OVER
    (PARTITION BY prod_id ORDER BY month_num RANGE
    BETWEEN 6 PRECEDING AND 2 PRECEDING)
FROM sale
ORDER BY prod_id, month_num;
```

このクエリの結果セットを次に示します。

```
prod_id   month_num       sales     sum(sales)
-------   ---------       -----     ----------
10                1         100         (NULL)
10                1         150         (NULL)
10                2         120         (NULL)
10                3         100            250
10                4         130            370
10                5         120            470
10                5          31            470
10                6         110            600
20                1          20         (NULL)
20                2          30         (NULL)
20                3          25             20
20                4          30             50
20                5          31             75
20                6          20            105
30                1          10         (NULL)
30                2          11         (NULL)
30                3          12             10
30                4           1             21
30                4           1             21
```

例：RANGE のウィンドウ・フレーム

このクエリは、RANGE のウィンドウ・フレームを示しています。合計で使用されるローの数値は、変数です。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num RANGE
   BETWEEN 1 FOLLOWING AND 3 FOLLOWING)
FROM sale
ORDER BY prod_id, month_num;
```

このクエリの結果セットを次に示します。

prod_id	month_num	sales	sum(sales)
10	1	100	350
10	1	150	350
10	2	120	381
10	3	100	391
10	4	130	261
10	5	120	110
10	5	31	110
10	6	110	(NULL)
20	1	20	85
20	2	30	86
20	3	25	81
20	4	30	51
20	5	31	20
20	6	20	(NULL)
30	1	10	25
30	2	11	14
30	3	12	2
30	4	1	(NULL)
30	4	1	(NULL)

例：UNBOUNDED PRECEDING と UNBOUNDED FOLLOWING

この例では、パーティション内のすべてのローがウィンドウ・フレームに含まれます。このクエリは、パーティション全体 (各月に重複ローは含まれていません) における売上の最大値を計算します。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num ROWS
   BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

このクエリの結果セットを次に示します。

prod_id	month_num	sales	max(sales)
10	1	100	680
10	2	120	680
10	3	100	680
10	4	130	680
10	5	120	680
10	6	110	680
20	1	20	156
20	2	30	156
20	3	25	156
20	4	30	156
20	5	31	156
20	6	20	156
30	1	10	34
30	2	11	34
30	3	12	34
30	4	1	34

このクエリは、次のクエリと同じ意味になります。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id )
FROM sale WHERE rep_id = 1
ORDER BY prod_id, month_num;
```

例：RANGE のデフォルトのウィンドウ・フレーム

このクエリは、RANGE のデフォルトのウィンドウ・フレームの例を示しています。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
   (PARTITION BY prod_id ORDER BY month_num)
FROM sale
ORDER BY prod_id, month_num;
```

このクエリの結果セットを次に示します。

prod_id	month_num	sales	max(sales)
10	1	100	250
10	1	150	250
10	2	120	370
10	3	100	470

```
10              4          130          600
10              5          120          751
10              5           31          751
10              6          110          861
20              1           20           20
20              2           30           50
20              3           25           75
20              4           30          105
20              5           31          136
20              6           20          156
30              1           10           10
30              2           11           21
30              3           12           33
30              4            1           35
30              4            1           35
```

このクエリは、次のクエリと同じ意味になります。

```
SELECT prod_id, month_num, sales, SUM(sales) OVER
    (PARTITION BY prod_id ORDER BY month_num RANGE
    BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM sale
ORDER BY prod_id, month_num;
```

OLAP 関数の BNF 文法

次の BNF (Backus-Naur Form) 文法は、さまざまな ANSI SQL 統計関数に関する具体的な構文サポートの概要を示しています。ここに記載されている関数の多くは Sybase IQ で実装されています。

文法規則 1

```
<SELECT LIST EXPRESSION> ::=
    <EXPRESSION>
    | <GROUP BY EXPRESSION>
    | <AGGREGATE FUNCTION>
    | <GROUPING FUNCTION>
    | <TABLE COLUMN>
    | <WINDOWED TABLE FUNCTION>
```

文法規則 2

```
<QUERY SPECIFICATION> ::=
    <FROM CLAUSE>
    [ <WHERE CLAUSE> ]
    [ <GROUP BY CLAUSE> ]
    [ <HAVING CLAUSE> ]
    [ <WINDOW CLAUSE> ]
[ <ORDER BY CLAUSE> ]
```

文法規則 3
```
<ORDER BY CLAUSE> ::= <ORDER SPECIFICATION>
```

文法規則 4
```
<GROUPING FUNCTION> ::=
    GROUPING <LEFT PAREN> <GROUP BY EXPRESSION>
    <RIGHT PAREN>
```

文法規則 5
```
<WINDOWED TABLE FUNCTION> ::=
    <WINDOWED TABLE FUNCTION TYPE> OVER <WINDOW NAME OR
    SPECIFICATION>
```

文法規則 6
```
<WINDOWED TABLE FUNCTION TYPE> ::=
    <RANK FUNCTION TYPE> <LEFT PAREN> <RIGHT PAREN>
    | ROW_NUMBER <LEFT PAREN> <RIGHT PAREN>
    | <WINDOW AGGREGATE FUNCTION>
```

文法規則 7
```
<RANK FUNCTION TYPE> ::=
    RANK | DENSE RANK | PERCENT RANK | CUME_DIST
```

文法規則 8
```
<WINDOW AGGREGATE FUNCTION> ::=
    <SIMPLE WINDOW AGGREGATE FUNCTION>
    | <STATISTICAL AGGREGATE FUNCTION>
```

文法規則 9
```
<AGGREGATE FUNCTION> ::=
    <DISTINCT AGGREGATE FUNCTION>
    | <SIMPLE AGGREGATE FUNCTION>
    | <STATISTICAL AGGREGATE FUNCTION>
```

文法規則 10
```
<DISTINCT AGGREGATE FUNCTION> ::=
    <BASIC AGGREGATE FUNCTION TYPE> <LEFT PAREN>
    <DISTINCT> <EXPRESSION> <RIGHT PAREN>
    | LIST <LEFT PAREN> DISTINCT <EXPRESSION>
    [ <COMMA> <DELIMITER> ]
    [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

文法規則 11
```
<BASIC AGGREGATE FUNCTION TYPE> ::=
    SUM | MAX | MIN | AVG | COUNT
```

文法規則 12
```
<SIMPLE AGGREGATE FUNCTION> ::=
    <SIMPLE AGGREGATE FUNCTION TYPE> <LEFT PAREN>
    <EXPRESSION> <RIGHT PAREN>
    | LIST <LEFT PAREN> <EXPRESSION> [ <COMMA>
    <DELIMITER> ]
    [ <ORDER SPECIFICATION> ] <RIGHT PAREN>
```

文法規則 13
```
<SIMPLE AGGREGATE FUNCTION TYPE> ::= <SIMPLE WINDOW
AGGREGATE FUNCTION TYPE>
```

文法規則 14
```
<SIMPLE WINDOW AGGREGATE FUNCTION> ::=
    <SIMPLE WINDOW AGGREGATE FUNCTION TYPE> <LEFT PAREN>
    <EXPRESSION> <RIGHT PAREN>
| GROUPING FUNCTION
```

文法規則 15
```
<SIMPLE WINDOW AGGREGATE FUNCTION TYPE> ::=
    <BASIC AGGREGATE FUNCTION TYPE>
    | STDDEV | STDDEV_POP | STDDEV_SAMP
    | VARIANCE | VARIANCE_POP | VARIANCE_SAMP
```

文法規則 16
```
<STATISTICAL AGGREGATE FUNCTION> ::=
    <STATISTICAL AGGREGATE FUNCTION TYPE> <LEFT PAREN>
    <DEPENDENT EXPRESSION> <COMMA> <INDEPENDENT
    EXPRESSION> <RIGHT PAREN>
```

文法規則 17
```
<STATISTICAL AGGREGATE FUNCTION TYPE> ::=
    CORR | COVAR_POP | COVAR_SAMP | REGR_R2 |
    REGR_INTERCEPT | REGR_COUNT | REGR_SLOPE |
    REGR_SXX | REGR_SXY | REGR_SYY | REGR_AVGY |
    REGR_AVGX
```

文法規則 18
```
<WINDOW NAME OR SPECIFICATION> ::=
    <WINDOW NAME> | <IN-LINE WINDOW SPECIFICATION>
```

文法規則 19
```
<WINDOW NAME> ::= <IDENTIFIER>
```

文法規則 20
```
<IN-LINE WINDOW SPECIFICATION> ::= <WINDOW
SPECIFICATION>
```

文法規則 21
```
<WINDOW CLAUSE> ::= <WINDOW WINDOW DEFINITION LIST>
```

文法規則 22
```
<WINDOW DEFINITION LIST> ::=
    <WINDOW DEFINITION> [ { <COMMA> <WINDOW DEFINITION>
    } . . . ]
```

文法規則 23
```
<WINDOW DEFINITION> ::=
    <NEW WINDOW NAME> AS <WINDOW SPECIFICATION>
```

文法規則 24
```
<NEW WINDOW NAME> ::= <WINDOW NAME>
```

文法規則 25
```
<WINDOW SPECIFICATION> ::=
    <LEFT PAREN> <WINDOW SPECIFICATION> <DETAILS> <RIGHT
    PAREN>
```

文法規則 26
```
<WINDOW SPECIFICATION DETAILS> ::=
    [ <EXISTING WINDOW NAME> ]
    [ <WINDOW PARTITION CLAUSE> ]
    [ <WINDOW ORDER CLAUSE> ]
    [ <WINDOW FRAME CLAUSE> ]
```

文法規則 27
```
<EXISTING WINDOW NAME> ::= <WINDOW NAME>
```

文法規則 28
```
<WINDOW PARTITION CLAUSE> ::=
    PARTITION BY <WINDOW PARTITION EXPRESSION LIST>
```

文法規則 29
```
<WINDOW PARTITION EXPRESSION LIST> ::=
    <WINDOW PARTITION EXPRESSION>
    [ { <COMMA> <WINDOW PARTITION EXPRESSION> } . . . ]
```

文法規則 30
```
<WINDOW PARTITION EXPRESSION> ::= <EXPRESSION>
```

文法規則 31
```
<WINDOW ORDER CLAUSE> ::= <ORDER SPECIFICATION>
```

文法規則 32
```
<WINDOW FRAME CLAUSE> ::=
    <WINDOW FRAME UNIT>
    <WINDOW FRAME EXTENT>
```

文法規則 33
```
<WINDOW FRAME UNIT> ::= ROWS | RANGE
```

文法規則 34
```
<WINDOW FRAME EXTENT> ::= <WINDOW FRAME START> | <WINDOW
FRAME BETWEEN>
```

文法規則 35
```
<WINDOW FRAME START> ::=
    UNBOUNDED PRECEDING
    | <WINDOW FRAME PRECEDING>
    | CURRENT ROW
```

文法規則 36
```
<WINDOW FRAME PRECEDING> ::= <UNSIGNED VALUE
SPECIFICATION> PRECEDING
```

文法規則 37
```
<WINDOW FRAME BETWEEN> ::=
    BETWEEN <WINDOW FRAME BOUND 1> AND <WINDOW FRAME
    BOUND 2>
```

文法規則 38
```
<WINDOW FRAME BOUND 1> ::= <WINDOW FRAME BOUND>
```

文法規則 39
```
<WINDOW FRAME BOUND 2> ::= <WINDOW FRAME BOUND>
```

文法規則 40
```
<WINDOW FRAME BOUND> ::=
    <WINDOW FRAME START>
    | UNBOUNDED FOLLOWING
    | <WINDOW FRAME FOLLOWING>
```

文法規則 41
```
<WINDOW FRAME FOLLOWING> ::= <UNSIGNED VALUE
SPECIFICATION> FOLLOWING
```

文法規則 42
```
<GROUP BY EXPRESSION> ::= <EXPRESSION>
```

文法規則 43
```
<SIMPLE GROUP BY TERM> ::=
    <GROUP BY EXPRESSION>
    | <LEFT PAREN> <GROUP BY EXPRESSION> <RIGHT PAREN>
    | <LEFT PAREN> <RIGHT PAREN>
```

文法規則 44
```
<SIMPLE GROUP BY TERM LIST> ::=
    <SIMPLE GROUP BY TERM> [ { <COMMA> <SIMPLE GROUP BY
    TERM> } . . . ]
```

文法規則 45
```
<COMPOSITE GROUP BY TERM> ::=
    <LEFT PAREN> <SIMPLE GROUP BY TERM>
    [ { <COMMA> <SIMPLE GROUP BY TERM> } . . . ]
    <RIGHT PAREN>
```

文法規則 46
```
<ROLLUP TERM> ::= ROLLUP <COMPOSITE GROUP BY TERM>
```

文法規則 47

```
<CUBE TERM> ::= CUBE <COMPOSITE GROUP BY TERM>
```

文法規則 48

```
<GROUP BY TERM> ::=
   <SIMPLE GROUP BY TERM>
   | <COMPOSITE GROUP BY TERM>
   | <ROLLUP TERM>
   | <CUBE TERM>
```

文法規則 49

```
<GROUP BY TERM LIST> ::=
   <GROUP BY TERM> [ { <COMMA> <GROUP BY TERM> } Åc ]
```

文法規則 50

```
<GROUP BY CLAUSE> ::= GROUP BY <GROUPING SPECIFICATION>
```

文法規則 51

```
<GROUPING SPECIFICATION> ::=
   <GROUP BY TERM LIST>
   | <SIMPLE GROUP BY TERM LIST> WITH ROLLUP
   | <SIMPLE GROUP BY TERM LIST> WITH CUBE
   | <GROUPING SETS SPECIFICATION>
```

文法規則 52

```
<GROUPING SETS SPECIFICATION> ::=
   GROUPING SETS <LEFT PAREN> <GROUP BY TERM LIST>
   <RIGHT PAREN>
```

文法規則 53

```
<ORDER SPECIFICATION> ::= ORDER BY <SORT SPECIFICATION
LIST>
   <SORT SPECIFICATION LIST> ::= <SORT SPECIFICATION>
   [ { <COMMA> <SORT SPECIFICATION> } . . . ]
   <SORT SPECIFICATION> ::= <SORT KEY>
   [ <ORDERING SPECIFICATION> ] [ <NULL ORDERING> ]
   <SORT KEY> ::= <VALUE EXPRESSION>
   <ORDERING SPECIFICATION> ::= ASC | DESC
   <NULL ORDERING> := NULLS FIRST | NULLS LAST
```

第 3 章

データ・サーバとしての Sybase IQ

この章について

Sybase IQ は、ODBC または JDBC を介したクライアント・アプリケーション接続をサポートしています。この章では、Sybase IQ をクライアント・アプリケーションのデータ・サーバとして使用する方法を説明します。

Sybase IQ は、特定のクライアント・アプリケーションに対しては、限定的ながら Open Server としても機能します。この章では、そのようなアプリケーションの作成時と実行時の制限についても簡単に説明します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — プログラミング > SQL Anywhere データ・アクセス API > Sybase Open Client API』の「Open Client アーキテクチャ」を参照してください。

この章で説明する機能は、Windows システムや Sun Solaris システムを使用する IQ ユーザに対してリモート・データ・アクセスを提供するものではありません。リモート・データ・アクセスは、OmniConnect™ の相互運用性機能の核であるコンポーネント統合サービス (CIS) によって提供されます。リモート・データ・アクセスとプロキシ・データベースの詳細については、「第 4 章 リモート・データへのアクセス」と「第 5 章 リモート・データ・アクセス用のサーバ・クラス」を参照してください。

内容

Sybase IQ とのクライアント/サーバ・インタフェース

単純化するために、Sybase アプリケーションまたはサード・パーティ・アプリケーションのクライアントを Sybase IQ と共に使用してください。接続用インタフェースやネットワーク・プロトコルの詳細を理解する必要はありません。しかし、これら 1 つ 1 つがどのように組み合わさっているかを理解すれば、データベースの構成やアプリケーションの設定をする上で役立ちます。このセクションでは、これらの要素の位置付けについて説明します。サード・パーティ製クライアント・アプリケーションの詳細については、『インストールおよび設定ガイド』を参照してください。

SQL Anywhere のマニュアルSQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用』『の「Open Client、Open Server、TDS」を参照してください。

iqdsedit による IQ Server の設定

Sybase IQ は、ネットワーク上の別の Adaptive Server、Open Server アプリケーション、クライアント・ソフトウェアと通信できます。クライアントは 1 つ以上のサーバと通信でき、サーバはリモート・プロシージャ・コールを通して他のサーバと通信できます。製品が互いに通信するには、それぞれの製品がネットワーク上のどこにあるかを互いに知っている必要があります。このネットワーク・サービス情報は interfaces ファイルに格納されています。

注意 Sybase IQ には、INSERT...LOCATION を使用可能にするために、次のような機能を提供する Open Client ユーティリティの一種が付いています。

- iqisql
- iqdsedit
- iqdscp (UNIX のみ)
- iqocscfg (Windows のみ)

---

interfaces ファイル

Open Client™ プログラムを使用してデータベース・サーバに接続すると、プログラムは interfaces ファイルでサーバの名前を検索し、そのアドレスを使用してサーバに接続します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「interfaces ファイル」を参照してください。

iqdsedit ユーティリティの使用

iqdsedit ユーティリティは interfaces ファイル (interfaces または SQL.ini) の設定に使用します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「DSEdit ユーティリティの使用」を参照してください。

iqdsedit の起動

Windows では、iqdsedit 実行可能ファイルは *SYBASE¥IQ-15_1¥bin32* または *SYBASE¥IQ-15_1¥bin64* ディレクトリにあります。これはインストール時に自動的に追加されます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「DSEdit の起動」を参照してください。

ディレクトリ・サービスのセッションを開く

[ディレクトリ・サービスの選択] ウィンドウでは、Sybase IQ サーバなどのサーバについて、エントリの追加、修正、削除ができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「ディレクトリ・サービスのセッションを開く」を参照してください。

サーバ・エントリの追加

サーバ・エントリが [サーバ] フィールドに表示されます。サーバの属性を指定するには、エントリを修正します。

ここで入力するサーバ名は、Sybase IQ のコマンド・ラインに表示される名前に一致させる必要はありません。サーバの識別や場所の確認には、サーバ名ではなく、「アドレス」が使用されます。

このサーバ名フィールドは、Open Client が識別の手がかりにするためのものです。Sybase IQ では、サーバにデータベースが 2 つ以上ロードされている場合、使用するデータベースを IQDSEDIT サーバ名エントリによって識別します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「サーバ・エントリの追加」を参照してください。

サーバ・アドレスの追加または変更

サーバ名を入力した後は、サーバ・アドレスを変更して interfaces ファイル入力を完了する必要があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「サーバ・アドレスの追加または変更」を参照してください。

ポート番号

入力するポート番号は、「Open Server としてのデータベース・サーバの起動」(99 ページ) で説明したように、Sybase IQ データベース・サーバのコマンド・ラインで指定されているポートに一致させます。Sybase IQ サーバのデフォルトのポート番号は、2638 です。

次に示すのは有効なサーバ・アドレス・エントリの例です。

```
elora,2638
123.85.234.029,2638
```

サーバ・アドレスの確認

Windows では、[サーバ・オブジェクト] メニューから [サーバに ping を実行] コマンドを使用してネットワーク接続を確認できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「サーバ・アドレスの確認」を参照してください。

サーバ・エントリの名前の変更

[*dsedit*] セッション・ウィンドウからサーバ・エントリの名前を変更できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「サーバ・エントリ名の変更」を参照してください。

サーバ・エントリの削除

[*dsedit*] セッション・ウィンドウからサーバ・エントリの名前を削除できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > Open Server の設定』の「サーバ・エントリの削除」を参照してください。

Sybase アプリケーションと Sybase IQ

Sybase IQ は Open Server としての役割も果たすので、OmniConnect などの Sybase アプリケーションを Sybase IQ で使用できます。Open Client ライブラリを使用する場合、クライアント・アプリケーションではサポートされているシステム・テーブル、ビュー、ストアド・プロシージャしか使用できません。

OmniConnect サポート

Sybase OmniConnect は組織内に存在する各種の異質なデータを統一して表示し、データの内容や保管場所がわからなくても複数のデータ・ソースにアクセスできるようにします。OmniConnect はさらに、企業全体のデータの異機種間ジョインを実行し、DB2、Sybase Adaptive Server Enterprise™、SQL Anywhere、Oracle、VSAM など、ターゲットのプラットホームを問わないテーブル・ジョインを可能にします。

Open Server インタフェースを使用すれば、Sybase IQ を OmniConnect のデータ・ソースとして利用できます。

Open Client アプリケーションと Sybase IQ

PowerSoft Power++™ のような C や C++ プログラミング環境から直接 Open Client ライブラリを使って、Sybase IQ ベースのテーブルにあるデータにアクセスする Open Client アプリケーションを作成できます。そうしたアプリケーションでカタログ・テーブル、ビュー、システム・ストアド・プロシージャを参照している場合は、これらのオブジェクトは Adaptive Server Enterprise (Transact-SQL™ 構文) と Sybase IQ の両方でサポートされている必要があります。

『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「付録 A 他の Sybase データベースとの互換性」を参照してください。

Open Client の設定

Open Client を使用して Sybase IQ に接続する場合、または INSERT...LOCATION 構文を使用する場合、Open Client 実行時設定 (*.cfg*) ファイルを使用して、各種の Open Client 設定パラメータを設定できます。たとえば、CS_MAX_CONNECT オプションの値で制御されている最大接続数のデフォルトを変更できます。

INSERT...LOCATION のアプリケーション名は `Sybase IQ` です (単語間のスペースは必須です)。このアプリケーション名は、Open Client のコンテキスト・レベルではなく、Open Client の接続レベルで設定されます。Open Client 実行時設定ファイルおよび使用可能なオプションの使用については、Open Client の『Client-Library/C リファレンス・マニュアル』を参照してください。

.cfg を有効にするには、Sybase IQ サーバを停止し、再起動します。また、INSERT...LOCATION コマンド・ラインで指定できる設定パラメータもあります。INSERT...LOCATION で設定されたパラメータは、設定ファイルで設定されたパラメータに置き換えられます。

Sybase IQ を Open Server として設定

ここでは、Open Client アプリケーションからの接続を受けるように Sybase IQ サーバを設定する方法について説明します。

システムの稼働条件

Sybase IQ を Open Server として使用するためには、クライアント側とサーバ側でそれぞれ稼働条件があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > SQL Anywhere を Open Server として設定する』の「システムの稼働条件」を参照してください。

注意 OmniConnect を使用してローカル SQL Anywhere Enterprise サーバからリモート Sybase IQ に接続する場合は、次のサーバ・クラスを使用してください。

- Sybase IQ 12 以降に接続するには、サーバ・クラス asaodbc と sajdbc を使用します。
- Sybase IQ 11.x に接続するには、サーバ・クラス asiq を使用します。

Open Server としてのデータベース・サーバの起動

Sybase IQ を Open Server として使用する場合は、必ず TCP/IP プロトコルを使用して起動してください。

SQL Anywhere マニュアルの『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > SQL Anywhere を Open Server として設定する』の「データベース・サーバを Open Server として起動する」を参照してください。

ネットワーク・パケットが正しいアプリケーションに届くようにするために、同じマシン上で TCP/IP を使用するすべてのアプリケーションは、それぞれ別の TCP/IP「ポート」を使用します。Sybase IQ のデフォルトのポートは 2638 です。これは共有メモリ通信に使用されます。次のように、別のポート番号を指定することもできます。

```
start_iq -x tcpip{port=2629} -n myserver iqdemo.db
```

Open Client で使用するためのデータベースの設定

データベースは Sybase IQ 12.0 以降でなければなりません。

Sybase IQ を Adaptive Server Enterprise と共に使用する場合は、データベースを Adaptive Server Enterprise との互換性ができるだけ高くなるように作成してください。

Open Server としての Sybase IQ に接続すると、多くの場合アプリケーション側では、Adaptive Server Enterprise の場合と同じサービスが提供されるものと期待します。これらのサービスは常に存在するわけではありません。

『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「付録 A 他の Sybase データベースとの互換性」を参照してください。

Open Client および jConnect 接続の特性

Sybase IQ は TDS を通してアプリケーションからの要求を処理するとき、関連したデータベース・オプションを自動的に SQL Anywhere サーバのデフォルトの動作と互換性のある値に設定します。それらのオプションは、接続されている間のみ一時的に設定されます。クライアント・アプリケーションはこれらのオプションをいつでも独自に設定して変更できます。

注意 Sybase IQ は `ANSI_BLANKS`、`AFLOAT_AS_DOUBLE`、`ATSQL_HEX_CONSTANT` の各オプションをサポートしません。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > レプリケーション > Open Server としての SQL Anywhere の使用 > SQL Anywhere を Open Server として設定する』の「Open Client と jConnect 接続の特性」を参照してください。

デフォルトでは、sp_iq_process_login システム・プロシージャが Sybase IQ ユーザの受け入れを行い、その後 sp_login_environment を呼び出します。それによって TDS 接続のための sp_tsql_environment が呼び出されます。

注意 Interactive SQL アプリケーションなどの ODBC アプリケーションは、ODBC 仕様で要求されるデータベース・オプションの値を自動的に設定します。これにより、LOGIN_PROCEDURE データベース・オプションによる設定が上書きされます。詳細と対処方法については、『リファレンス：文とオプション』の「LOGIN_PROCEDURE オプション」を参照してください。

複数のデータベースがあるサーバ

サーバに複数のデータベースがある場合、Open Client Library を使用して、接続するデータベースを指定できます。

- *interfaces* ファイル内にサーバのエントリを設定します。
- start_iq コマンドで -n パラメータを指定し、データベース名のショートカットを設定します。
- isql コマンドで -S *database_name* パラメータをデータベース名と共に指定します。このパラメータは接続時に常に必要です。

プログラム自体を変更しなくても、ショートカット名をプログラムに記述し、ショートカット定義を変更するだけで、同じプログラムを複数のデータベースに対して実行できます。

たとえば、次の live_sales と test_sales の 2 つのサーバ定義は *interfaces* ファイルから抜粋したものです。

```
live_sales
     query tcp ether myhostname 5555
    master tcp ether myhostname 5555
test_sales
     query tcp ether myhostname 7777
    master tcp ether myhostname 7777
```

サーバを起動して、所定のデータベースのエイリアスを設定します。次のコマンドは、live_sales を salesbase.db と等価に設定します。

```
start_iq -n sales_live <other parameters> -x ¥
'tcpip{port=5555}' salesbase.db -n live_sales
```

live_sales サーバに接続するには

```
isql -Udba -Psql -Slive_sales
```

サーバ名は *interfaces* ファイル内に一度しか記述しません。これは、Sybase IQ への接続がデータベース名に基づくようにしたため、データベース名はユニークでなければならないからです。全スクリプトが *salesbase* データベースを利用するように設定されている場合、live_sales や test_sales を利用するようにスクリプトを変更する必要はありません。

第 4 章

リモート・データへのアクセス

この章について

Sybase IQ では、他のサーバに置かれているデータに対しても、ローカル・データにアクセスするかのような感覚でアクセスできます。これは、アクセス先が Sybase のサーバでも Sybase 以外のサーバでも同じです。

内容

Sybase IQ とリモート・データ

SQL Anywhere のリモート・データ・アクセスを使用すると、他のデータ・ソースのデータにアクセスできます。この機能を使用して、データを SQL Anywhere データベースに移行できます。また、複数のデータベース内のデータを問い合わせることができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション』の「Open Client と jConnect 接続の特性」を参照してください。

リモート・データにアクセスするための要件

ここでは、リモート・データへのアクセスに必要な基本要素について説明します。

リモート・テーブルのマッピング

Sybase IQ は、テーブル内の全データがアプリケーションの接続先データベースに格納されているかのように見えるよう、クライアント・アプリケーションにテーブルを提示します。内部的には、Sybase IQ はリモート・テーブルが関与するクエリを実行して記憶領域の場所を特定し、リモート・ロケーションにアクセスしてデータを取り出します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス』の「リモート・テーブルのマッピング」を参照してください。

サーバ・クラス

サーバ・クラスは、各リモート・サーバに割り当てられています。サーバ・クラスは、サーバとの対話に使用するアクセス方法を示すものであり、リモート・サーバの種類によって、必要とされるアクセス方法の種類も異なります。サーバ・クラスは Sybase IQ に詳細なサーバ機能情報を提供します。Sybase IQ はこの機能に基づいてリモート・サーバとの対話を調節します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス』の「サーバ・クラス」を参照してください。

注意 OMNI JDBC は、IPv6 ではサポートされていません。

リモート・サーバの使用

リモート・プロキシ・テーブルをローカル・プロキシ・テーブルにマッピングするには、リモート・オブジェクトを配置するリモート・サーバを定義する必要があります。これによって、リモート・サーバの ISYSSERVER システム・テーブルにエントリが追加されます。

リモート・サーバの作成

CREATE SERVER 文を使用してリモート・サーバの定義を設定します。

Sybase IQ や SQL Anywhere を含む一部のシステムでは、各データ・ソースが 1 つのデータベースを表すため、データベースごとにリモート・サーバの定義が必要になります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・サーバの使用』の「CREATE SERVER 文を使用して、リモート・サーバを作成します。」を参照してください。

ネイティブ・クラスなしでのリモート・データのロード

次の場所のリモート・データ・ソースにアクセスするには、DirectConnect を使用する必要があります。

- 64 ビット UNIX プラットフォーム
- ODBC ドライバを使用できない 32 ビット・プラットフォーム (Microsoft SQL Server など)

これ以降は、DirectConnect によるデータのロードとクエリの例を示します。

非 Sybase リモート・データの例

この例では次の状況を想定します。

- *mssql* という名前の Enterprise Connect Data Access (ECDA) サーバが UNIX ホスト *myhostname*、ポート 12530 上にあります。
- データは、ホスト *myhostname*、ポート 1433 上のMS SQL サーバ (名前は *2000*) から取り出されます。

❖ **UNIX 上の IQ サーバに MS SQL Server データをロード**

1. DirectConnect のマニュアルに従って、DirectConnect をデータ・ソースに合わせて設定します。
2. ECDA サーバ (*mssql*) が IQ インタフェース・ファイルにリストされているかどうかを確認します。

```
mssql
master tcp ether myhostname 12530
query tcp ether myhostname 12530
```

3. *mssql* サーバのユーザ ID とパスワードを使用して、新しいユーザを追加します。

```
isql -Udba -Psql -Stst_iqdemo
grant connect to chill identified by chill
grant dba to chill
```

4. 新しいユーザとしてログインし、IQ 上にローカル・テーブルを作成します。

```
isql -Uchill -Pchill -Stst_iqdemo
create table billing(status char(1), name
varchar(20), telno int)
```

5. データを挿入します。

```
insert into billing location 'mssql.pubs' { select *
from billing }
```

ネイティブ・クラスなしでデータのクエリを実行

64 ビット・システム上の非 Sybase データにアクセスするには、次に示す間接的な方法を取るのが現時点では最善のアプローチです。

1. DirectConnect 経由で接続するよう、ASE/CIS、リモート・サーバ、プロキシを設定します。たとえば、Oracle サーバには DirectConnect for Oracle を使用します。

2　ASEJDBC クラスを ASE サーバに使用して、IQ と リモート・サーバを設定します (ASE 用の 64 ビット Unix ODBC ドライバが存在しないため、ASEODBC クラスは利用できません)。

3　CREATE EXISTING TABLE 文を使用してプロキシ・テーブルを作成し、そのプロキシ・テーブルが指す ASE 内のプロキシ・テーブルを介して最終的に Oracle を指すようにします。

UNIX 上で DirectConnect とプロキシ・テーブルを使用して、リモート・データのクエリを実行

この例は、MS SQL Server データへのアクセス方法を示します。この例では次の状況を想定します。

- ホスト *myhostname*、ポート 7594 の上に Sybase IQ サーバが存在しています。
- ホスト *myhostname*、ポート 4101 の上に Adaptive Server Enterprise サーバが存在しています。
- ホスト *myhostname*、ポート 12530 の上に、*mssql* という名前の Enterprise Connect Data Access (ECDA) サーバが存在しています。
- データは、ホスト *myhostname*、ポート 1433 上のMS SQL サーバ (名前は *2000*) から取り出されます。

❖ **MS SQL Server のクエリを実行できるよう Adaptive Server Enterprise を設定**

1　DirectConnect を介して MS SQL Server にアクセスできるよう、Adaptive Server とコンポーネント統合サービス (CIS) を設定します。たとえば、サーバ名が *jones_1207* であるとします。

2　*mssql* に接続するためのエントリを ASE インタフェース・ファイルに追加します。

```
mssql

master tcp ether hostname 12530

query tcp ether hostname 12530
```

3　ASE サーバでの CIS とリモート・プロシージャ・コールの処理を有効にします。たとえば、CIS がデフォルトで有効になっているとします。

```
sp_configure 'enable cis'

Parameter Name Default Memory Used Config Value Run Value

enable cis       1      0            1            1

(1 row affected)
(return status=0)
```

```
                    sp_configure 'cis rpc handling', 1
Parameter Name Default Memory Used Config Value Run Value
enable cis       0      0            0            1
(1 row affected)
Configuation option changed. The SQL Server need not be restarted since
the option is dynamic.
```

この場合、Sybase IQ 12.5 などの古いバージョンによる CIS リモート・プロシージャ・コールの処理を有効にした後で、Adaptive Server Enterprise サーバを再起動しなければならないこともあります。

4 ASE サーバの SYSSERVERS システム・テーブルに DirectConnect サーバを追加します。

```
sp_addserver mssql, direct_connect, mssql
Adding server 'mssql', physical name 'mssql'
Server added.
(Return status=0)
```

5 ASE に接続するために Sybase IQ で使用するユーザを Adaptive Server Enterprise に作成します。

```
sp_addlogin tst, tsttst
Password correctly set.
Account unlocked. New login created.
(return status = 0)
grant role sa_role to tst
use tst_db
sp_adduser tst
New user added.
(return status = 0)
```

6 マスタ・データベースから外部ログインを追加します。

```
use master
sp_addexternlogin mssql, tst, chill, chill
User 'tst' will be known as 'chill' in remote server
'mssql'.
(return status = 0)
```

7 目的のデータベースから追加されたユーザとして、ASE プロキシ・テーブルを作成します。

```
isql -Utst -Ttsttst
```

```
use test_db
create proxy_table billing_tst at
'mssql.pubs..billing'
select * from billing_tst
```

```
status    name          telno
------    -----------   -----
D         BOTANICALLY   1
B         BOTANICALL    2
(2 rows affected)
```

❖ **ASE サーバに接続できるよう Sybase IQ を設定**

1 IQ インタフェース・ファイルにエントリを追加します。

```
jones_1207
master tcp ether jones 4101
query tcp ether jones 4101
```

2 ASE への接続に使用するユーザを作成します。

```
grant connect to tst identified by tsttst
grant dba to tst
```

3 追加されたユーザとしてログインし、'asejdbc' サーバ・クラスを作成して外部ログインを追加します。

```
isql -Utst -Ptsttst -Stst_iqdemo
create SERVER jones_1207 CLASEE 'asejdbc' USING
'jones:4101/tst_db'
create existing table billing_iq at
'jones_1207.tst_db..billing_txt'
select * from billing_iq
```

```
status    name          telno
------    -----------   -----
D         BOTANICALLY   1
B         BOTANICALL    2
(2 rows affected)
```

リモート・サーバの削除

Sybase Central または DROP SERVER 文を使用して、ISYSSERVER システム・テーブルからリモート・サーバを削除できます。このアクションを正しく実行するには、そのサーバ上で定義されているすべてのリモート・テーブルが削除済みであることが必要です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・サーバの使用』の「リモート・サーバの削除」を参照してください。

リモート・サーバの変更

サーバの属性を変更するには、ALTER SERVER 文を使用します。これらの変更は、次にリモート・サーバに接続するまで有効になりません。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・サーバの使用』の「リモート・サーバの変更」を参照してください。

サーバ上のリモート・テーブルをリスト

Sybase IQ を設定するとき、特定のサーバで利用できるリモート・テーブルのリストがあると便利です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・サーバの使用』の「サーバ上のリモート・テーブルのリスト」を参照してください。

『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「sp_remote_tables システム・プロシージャ」も参照してください。

リモート・サーバ機能のリスト

sp_servercaps プロシージャは、リモート・サーバの機能に関する情報を表示します。Sybase IQ はこの機能情報に基づいて、1 つのリモート・サーバに渡すことのできる SQL 文の数を判定します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・サーバの使用』の「リモート・サーバの機能のリスト」を参照してください。

『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「sp_servercaps システム・プロシージャ」も参照してください。

外部ログインの使用

Sybase IQ は、クライアントに代わってリモート・サーバに接続するときに、それらのクライアントの名前とパスワードを使用します。ただし、外部ログインを作成すると、この動作を無効にすることができます。外部ログインとは、リモート・サーバとの通信時に使用される代替ログイン名とパスワードのことです。

Sybase IQ がリモート・サーバに接続するときに、CREATE EXTERNLOGIN でリモート・ログインが作成されており、CREATE SERVER 文でリモート・サーバが定義されている場合は、INSERT...LOCATION は現在の接続のユーザ ID にリモート・ログインを使用します。リモート・サーバが定義されていないか、現在の接続のユーザ ID に対するリモート・ログインが作成されていない場合、IQ は現在の接続のユーザ ID とパスワードを使用して接続します。リモート・ログインを使用する INSERT...LOCATION の詳細と例については、『リファレンス：文とオプション』の「INSERT 文」を参照してください。

統合化ログインを使用する場合、IQ クライアントの IQ 名とパスワードは、IQ のユーザ ID が syslogins にマッピングしたデータベースのログイン ID およびパスワードと同じです。

外部ログインの作成

外部ログインを追加または変更できるのは、そのログイン名と DBA アカウントだけです。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・サーバの使用』の「外部ログインの作成」を参照してください。

詳細については、『リファレンス：文とオプション』の「CREATE EXTERNLOGIN 文」を参照してください。

外部ログインの削除

Sybase IQ のシステム・テーブルから外部ログインを削除するには、DROP EXTERNLOGIN 文を使用します。

詳細については、『リファレンス：文とオプション』の「DROP EXTERNLOGIN 文」を参照してください。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・サーバの使用』の「外部ログインの削除」を参照してください。

プロキシ・テーブルの使用

リモート・オブジェクトをマッピングするローカル・プロキシ・テーブルを作成すると、実際にどこにあるかは意識せずにリモート・データを扱えるようになります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス』の「プロキシ・テーブルの操作」を参照してください。

プロキシ・テーブルのロケーションの指定

CREATE TABLE と CREATE EXISTING TABLE のどちらの文でも、既存オブジェクトのロケーションを定義するには AT キーワードを使用します。このロケーション文字列には 4 つの要素があり、それぞれをピリオドまたはセミコロンで区切ります。セミコロンを使用すると、database と owner の各フィールドにファイル名と拡張子を指定できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > プロキシ・テーブルの使用』の「プロキシ・テーブルのロケーションの指定」を参照してください。

例

次の例は、ロケーション文字列の使用方法を示します。

- Sybase IQ:

  ```
  'testiq..DBA.employee'
  ```

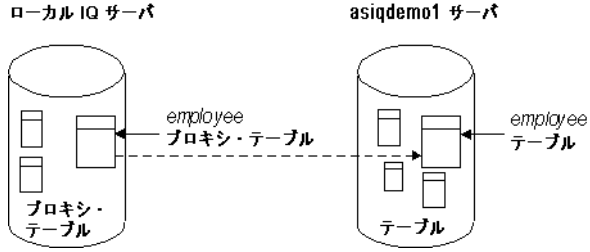
プロキシ・テーブルの作成

CREATE EXISTING TABLE 文は、リモート・サーバ上の既存テーブルをマッピングするプロキシ・テーブルを作成します。Sybase IQ は、リモート・ロケーションのオブジェクトからカラム属性とインデックス情報を導出します。

例

iqdemo1 サーバ上のリモート・テーブル employee をマッピングするプロキシ・テーブル p_employee を現在のサーバ上に作成するには、次の構文を使用します。

```
CREATE EXISTING TABLE p_employee
AT 'iqdemo1..DBA.employee'
```



詳細については、『リファレンス：文とオプション』の「CREATE EXISTING TABLE 文」を参照してください。

CREATE TABLE 文の使用

AT オプションを指定した場合、CREATE TABLE 文は、リモート・サーバ上に新しいテーブルを作成し、さらにそのテーブルに対するプロキシ・テーブルを定義します。カラムは、Sybase IQ データ型を使用して定義されます。データは Sybase IQ により、リモート・サーバのネイティブ形式に自動的に変換されます。

CREATE TABLE 文を使用してローカル・テーブルとリモート・テーブルの両方を作成し、続いて DROP TABLE 文を使用してプロキシ・テーブルを削除すると、リモート・テーブルも削除されます。ただし、CREATE EXISTING TABLE 文を使用して作成したプロキシ・テーブルを DROP TABLE 文で削除できますが、この場合は、リモート・テーブルは削除されません。

例

次に示す文は、リモート・サーバ iqdemo1 上に employee テーブルを作成し、リモート・ロケーションにマッピングするプロキシ・テーブル members を作成します。

```
CREATE TABLE members
( membership_id INTEGER NOT NULL,
member_name CHAR(30) NOT NULL,
office_held CHAR( 20 ) NULL)
AT 'iqdemo1..DBA.Employees'
```

詳細については、『リファレンス：文とオプション』の「INSERT 文」を参照してください。

リモート・テーブル上のカラムをリスト

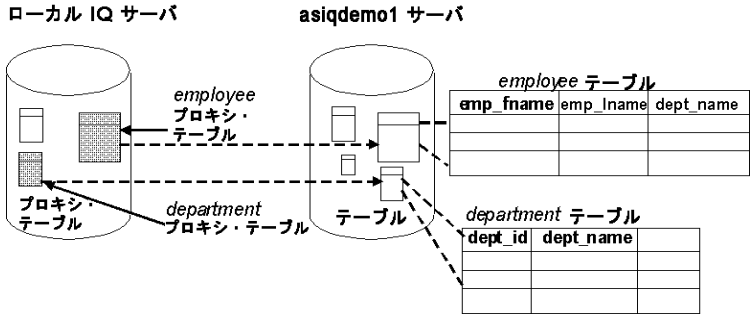
CREATE EXISTING TABLE 文を入力してカラムのリストを指定すると、リモート・テーブルで利用可能なカラムのリストを得るのに便利です。sp_remote_columns システム・プロシージャは、リモート・テーブルのカラムのリストとそれらのデータ型についての説明を生成します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > プロキシ・テーブルの使用』の「リモート・テーブルのカラムのリスト」を参照してください。

詳細については、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「sp_remote_columns システム・プロシージャ」を参照してください。

例：2 つのリモート・テーブルのジョイン

次の図は、サンプル・データベースのリモート Sybase IQ テーブルの employee と department をローカル・サーバ testiq にマッピングしたようすを示します。



SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス』の「リモート・テーブルのジョイン」を参照してください。

複数のローカル・データベースへのアクセス

Sybase IQ サーバでは、同時に複数のローカル・データベースを稼働させることができます。他のローカル Sybase IQ データベースのテーブルをリモート・テーブルとして定義することにより、テーブル間のジョインを実行できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス』の「複数のローカル・データベースのテーブルのジョイン」を参照してください。

リモート・サーバにネイティブ文を送信

FORWARD TO 文を使用して、1 つまたは複数の文をネイティブ構文でリモート・サーバに送信します。この文の使用方法は、次の 2 とおりです。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス』の「ネイティブ文のリモート・サーバへの送信」を参照してください。

リモート・プロシージャ・コール (RPC) の使用

Sybase IQ ユーザは、機能をサポートしているリモート・サーバへのプロシージャ・コールを発行することができます。

Sybase IQ、SQL Anywhere、Adaptive Server Enterprise、Oracle、DB2 はこの機能をサポートしています。リモート・プロシージャ・コールを発行するのは、ローカル・プロシージャ・コールを使用するのと同様です。

リモート・プロシージャの作成

次のプロシージャのいずれかを使用して、リモート・プロシージャ・コールを発行します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・プロシージャ・コール (RPC) の使用』の「リモート・プロシージャの作成」を参照してください。

トランザクション管理とリモート・データ

トランザクションでは、複数の SQL 文をグループ化して 1 つの単位として扱うことができます。つまり、すべての文の実行結果がデータベースにコミットされるか、実行結果が何もコミットされないかの、どちらかになります。

リモート・テーブルとローカル IQ テーブルとでは、トランザクション管理の扱いに若干の違いがあります。リモート・テーブルのトランザクション管理は、SQL Anywhere の場合とほとんど同様に扱われますが、一部に違いがあります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > データベースの作成』の「トランザクションと独立性レベルの使用」を参照してください。

Sybase IQ でのトランザクションの概要については、『システム管理ガイド 第 1 巻』の「第 10 章 トランザクションとバージョン管理」を参照してください。

リモート・トランザクション管理の概要

リモート・サーバが関与するトランザクションの管理には、「**2 フェーズ・コミット**」プロトコルを使用します。Sybase IQ には、ほとんどのシナリオでトランザクションの一貫性を維持することのできる戦略が実装されています。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データへのアクセス > トランザクションの管理とリモート・データ』の「リモート・トランザクション管理の概要」を参照してください。

トランザクション管理の制限

トランザクション管理には次の制約があります。

- セーブポイントはリモート・サーバには伝達されません。
- リモート・サーバに関わるトランザクションに、BEGIN TRANSACTION 文と COMMIT TRANSACTION 文の組がネストされている場合は、一番外側の組だけが処理されます。一番内側の BEGIN TRANSACTION 文と COMMIT TRANSACTION 文の組は、リモート・サーバには送信されません。

内部操作

ここでは、リモート・サーバ上で SQL Anywhere がクライアント・アプリケーションに代わって実行している基本手順について説明します。

クエリの解析

クライアントから文が届くと、文はデータベース・サーバによって解析されます。その文が有効な SQL Anywhere SQL 文でなかった場合は、データベース・サーバでエラーが発生します。

クエリの正規化

次の手順は、クエリの正規化と呼ばれるものです。この手順では、参照されているオブジェクトを検証し、一部のデータ型の互換性をチェックします。

次のクエリ例を見てみましょう。

```
SELECT *
FROM t1
WHERE c1 = 10
```

このクエリの正規化では、c1 カラムを持つテーブル t1 がシステム・カタログに存在することを検証します。また、c1 カラムのデータ型が値 10 と合っているかを確認します。たとえば、このカラムのデータ型が datetime だったとすると、この文は拒否されます。

クエリの前処理

クエリの前処理は、クエリの最適化の準備をします。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > 内部オペレーション』の「クエリの前処理」を参照してください。

サーバ機能

前述の手順は、すべてのクエリ (ローカルとリモートの両方) に使用します。

次の手順は、SQL 文の種類とリモート・サーバの機能によって異なります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > 内部オペレーション』の「サーバの機能」を参照してください。

文の完全なパススルー

文を処理する最も効率的な方法は、元の文をなるべくそのままの形でリモート・サーバに渡すことです。デフォルトでは、Sybase IQ はこの方針に従って文を渡します。多くの場合、Sybase IQ に渡された文は、そのままの完全な形でリモート・サーバに渡されます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > 内部オペレーション』の「文の完全なパススルー」を参照してください。

文の部分的なパススルー

1 つの文の中で複数のサーバが参照されている場合、または、リモート・サーバがサポートしていない SQL 機能が指定されている場合、クエリはよりシンプルな要素へと分解されます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > 内部オペレーション』の「文の部分的なパススルー」を参照してください。

リモート・データ・アクセスのトラブルシューティング

ここでは、リモート・サーバへのアクセスのトラブルシューティングのヒントについて説明します。

リモート・データには使用できない機能

次の機能は、リモート・データではサポートしていません。Sybase IQ がまったくサポートしていない機能もあれば、ローカル・データだけをサポートしている機能もあります。Sybase IQ では、SQL Anywhere のリストに次が追加されています。

- Java データ型がサポートされていない。
- 特定の地域でコンポーネント統合サービス (CIS) を使用している場合、接続要求を行うと No Suitable Driver エラーが返る。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・データ・アクセスのトラブルシューティング』の「リモート・データに対してサポートされない機能」を参照してください。

大文字と小文字の区別

IQ データベースの大文字と小文字の扱いは、アクセス先のリモート・サーバの設定と合わせる必要があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・データ・アクセスのトラブルシューティング』の「大文字と小文字の区別」を参照してください。

接続の問題

リモート・サーバに接続できることを確認するには、簡単なパススルー文を実行して、接続とリモート・ログインの設定をチェックします。次に例を示します。

```
FORWARD TO testiq {select @@version}
```

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・データ・アクセスのトラブルシューティング』の「接続のテスト」を参照してください。

クエリ関連の一般的問題

Sybase IQ でリモート・テーブルに対するクエリを処理していて何か問題が発生するという場合は、Sybase IQ がクエリを実行する過程を理解することで解決につながる可能性があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・データ・アクセスのトラブルシューティング』の「クエリに関する一般的な問題」を参照してください。

自動的にブロックされるクエリ

1 つの Sybase IQ サーバまたは SQL Anywhere サーバ上の複数のデータベースにアクセスする場合は、コマンドラインに -gx スイッチを指定して、そのデータベース・サーバで使用するスレッドの数を増やすことをおすすめします。デフォルトでは、このスイッチは、マシン上の CPU の数に 1 を加えた数に設定されています。

クエリによって実行される個々のタスクをサポートできるよう、スレッドの数には余裕を持たせておく必要があります。必要とされるだけのタスクを実行できないと、クエリは自動的にブロックされます。

注意 -gx スイッチは、通常は IQ データベースに設定する必要がないため、『ユーティリティ・ガイド』には説明されていません。ここに説明した以外の目的でスレッドの数を増やすには、IQ ストア操作用のスレッドの数を制御する -iqmt スイッチを設定してください。

リモート・データ・アクセス接続の管理

ODBC 経由でリモート・データベースにアクセスするとき、リモート・サーバへの接続には名前が与えられます。この名前は、リモート要求をキャンセルする方法の 1 つとして接続を切断するときにも使用します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データへのアクセス > リモート・データ・アクセスのトラブルシューティング』の「ODBC を使用したリモート・データ・アクセスの接続の管理」を参照してください。

第 5 章

リモート・データ・アクセス用のサーバ・クラス

この章について

この章では、Sybase IQ と各種サーバ・クラスとのやり取りがどのように行われるかについて説明します。

内容

サーバ・クラスの概要

リモート接続の動作は、CREATE SERVER 文内のサーバ・クラスによって決定されます。サーバ・クラスは、Sybase IQ に詳細なサーバ機能情報を提供します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション』の「リモート・データ・アクセスのサーバ・クラス」を参照してください。

JDBC ベースのサーバ・クラス

JDBC ベースのサーバ・クラスが使用されるのは、Sybase IQ がリモート・サーバに接続するために Java 仮想マシンと jConnect™ for JDBC™ 5.5 を内部的に使用したときです。JDBC ベースのサーバ・クラスには次の種類があります。

- **sajdbc** Sybase IQ と SQL Anywhere
- **asejdbc** Sybase SQL Anywhere と Adaptive Server Enterprise (バージョン 10 以降)

JDBC クラスの設定上の注意事項

JDBC ベースのクラスで定義されたリモート・サーバにアクセスするときは、このトピック内の情報を考慮してください。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > JDBC ベースのサーバ・クラス』の「JDBC クラスの設定上の注意」を参照してください。

サーバ・クラス sajdbc

Sybase IQ や SQL Anywhere のデータ・ソースの設定に関しては、特別な要件はありません。

CREATE SERVER 文の USING パラメータ値

CREATE SERVER 文の USING パラメータは、*hostname:portnumber [/databasename]* という書式で指定します。

- **hostname** リモート・サーバを実行しているマシンです。
- **portnumber** リモート・サーバが受信している TCP/IP のポート番号です。Sybase IQ が受信するデフォルトのポート番号は 2638 です。
- **databasename** その接続で使用される Sybase IQ データベースです。これは、サーバ起動時に -n スイッチに指定された名前、または DBN (DatabaseName) 接続パラメータに指定された名前です。

Sybase IQ の例

apple という名前のマシン上にありポート番号 2638 を受信している、testiq という Sybase IQ サーバを設定するには、次の文を実行します。

```
CREATE SERVER testiq
CLASS 'sajdbc'
USING 'apple:2638'
```

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > JDBC ベースのサーバ・クラス > サーバ・クラス sajdbc』の「CREATE SERVER 文の USING パラメータ」を参照してください。

サーバ・クラス asejdbc

サーバ・クラス asejdbc を持つサーバは、次のいずれかです。

- Adaptive Server Enterprise
- SQL Anywhere バージョン 10 以降

Adaptive Server Enterprise のデータ・ソースの設定に関しては、特別な要件はありません。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > JDBC ベースのサーバ・クラス』の「サーバ・クラス asejdbc」を参照してください。

データ型変換

プロキシ・テーブルを作成するために CREATE TABLE 文を発行すると、Sybase IQ はデータ型を対応する Adaptive Server Enterprise のデータ型に自動的に変換します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > JDBC ベースのサーバ・クラス』の「サーバ・クラス asejdbc」を参照してください。

ODBC ベースのサーバ・クラス

Sybase IQ は各種の ODBC ベースのサーバ・クラスをサポートします。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス』の「ODBC ベースのサーバ・クラス」を参照してください。

ODBC 外部サーバの定義

ODBC ベースのサーバを定義する最も一般的な方法は、ODBC データ・ソースを利用する方法です。これを実行するには、ODBC アドミニストレータでデータ・ソース名 (DSN) を作成する必要があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > ODBC ベースのサーバ・クラス』の「ODBC 外部サーバの定義」を参照してください。

Sybase IQ の例

Sybase IQ への接続は次のようになります。

```
CREATE SERVER testiq
CLASS 'asaodbc'
USING 'driver=adaptive server IQ 12.0;
eng=testasaiq;dbn=iqdemo;links=tcpip{}'
```

Sybase IQ に ODBC データ・ソースを作成する方法の詳細については、『システム管理ガイド 第 1 巻』の「第 3 章 Sybase IQ の接続」の「ODBC データ・ソースの作成と編集」を参照してください。

サーバ・クラス saodbc

サーバ・クラス saodbc を持つサーバは、次のいずれかです。

- Sybase IQ バージョン 12 以降
- SQL Anywhere

Sybase IQ や SQL Anywhere のデータ・ソースの設定に関しては、特別な要件はありません。

複数のデータベースをサポートしている SQL Anywhere データベースサーバにアクセスするには、各データベースへの接続を定義する ODBC データ・ソース名を作成します。これらの ODBC データ・ソース名ごとに、CREATE SERVER 文を発行します。

サーバ・クラス aseodbc

サーバ・クラス aseodbc を持つサーバは、次のいずれかです。

- Adaptive Server Enterprise
- SQL Anywhere (バージョン 10 以降)

Sybase IQ からaseodbc クラスを持つリモート Adaptive Server に接続するには、ローカルにインストールされている Adaptive Server Enterprise ODBC ドライバと Open Client コネクティビティ・ライブラリが必要です。しかし、パフォーマンスは、asejdbc クラスを使った場合に比べて高くなります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > ODBC ベースのサーバ・クラス』の「サーバ・クラス aseodbc」を参照してください。

サーバ・クラス db2odbc

サーバ・クラス db2odbc を持つサーバは IBM DB2 です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > ODBC ベースのサーバ・クラス』の「サーバ・クラス db2odbc」を参照してください。

サーバ・クラス oraodbc

サーバ・クラス oraodbc を持つサーバは、Oracle バージョン 8.0 以降です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > ODBC ベースのサーバ・クラス』の「サーバ・クラス oraodbc」を参照してください。

Oracle データにアクセスするには、Windows 上で稼働している Sybase IQ サーバのみを使用してください。64 ビット UNIX システム上で稼働している Sybase IQ サーバに対応した 64 ビット・ドライバ・マネージャはありません。

- **64 ビット UNIX 上の Sybase IQ サーバを使用した Oracle データのクエリ**

  この状況下で Oracle データのクエリを実行するために必要な 2 つのプロキシ・テーブルを作成する手順は次のとおりです。

  1. Oracle に接続できるよう、DirectConnect for Oracle を設定します。
  2. DirectConnect for Oracle のプロキシ・テーブルを設定します。
  3. サーバの ASEJDBC クラスと DirectConnect for Oracle のポート番号を使用して、Sybase IQ にリモート・サーバを作成します。
  4. CREATE EXISTING TABLE 文を使用して、ASE プロキシ・テーブルを指すプロキシ・テーブルを DirectConnect for Oracle に作成します。

❖ **64 ビット UNIX 上の Sybase IQ サーバを使用した Oracle データのロード**

大量のデータをロードするときは、最適なパフォーマンスが得られるよう、クエリの場合とは異なる次の方法でリモート・データベースにアクセスします。

1 DirectConnect for Oracle にプロキシ・テーブルを作成します。

2 INSERT .. LOCATION 文をプロキシ・テーブルに対して使用します。

INSERT .. LOCATION の詳細については、『システム管理ガイド 第 1 巻』の「第 7 章 データベースへのデータの入出力」の「別のデータベースからの挿入」を参照してください。

サーバ・クラス mssodbc

サーバ・クラス mssodbc を持つサーバは、Microsoft SQL Anywhere バージョン 6.5、Service Pack 4 です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > ODBC ベースのサーバ・クラス』の「サーバ・クラス mssodbc」を参照してください。

サーバ・クラス odbc

独自のサーバ・クラスを持たない ODBC データ・ソースは、odbc というサーバ・クラスを使用します。ODBC ドライバはどれでも使用できます。

Microsoft ODBC ドライバの最新バージョンは、Microsoft のダウンロード センタで配布されている Microsoft Data Access Components (MDAC) を通じて入手することができます。以下に示す Microsoft ドライバのバージョンは、MDAC 2.0 のものです。

Microsoft Excel (Microsoft 3.51.171300)

Excel の各ワークブックは複数のテーブルを持つデータベースと考えることができます。ワークブックのシートがテーブルに相当します。ODBC データ・ソース名を ODBC ドライバ・マネージャで設定するときは、そのデータ・ソースに関連するデフォルトのワークブック名を指定します。しかし、CREATE TABLE 文を発行するときは、デフォルトを無効にしてロケーション文字列でワークブック名を指定することができます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > ODBC ベースのサーバ・クラス > サーバ・クラス odbc』の「Microsoft Excel (Microsoft 3.51.171300)」を参照してください。

Microsoft Foxpro (Microsoft 3.51.171300)

複数の Foxpro テーブルを 1 つの Foxpro データベース ファイル (*.dbc*) にまとめて格納することも、各テーブルを独自の *.dbf* ファイルに個別に格納することもできます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > ODBC ベースのサーバ・クラス > サーバ・クラス odbc』の「Microsoft FoxPro (Microsoft 3.51.171300)」を参照してください。

Lotus Notes SQL 2.0 (2.04.0203)

このドライバは Lotus の Web サイトで入手できます。Notes データがどのようにリレーショナル・テーブルにマッピングされるかについては、ドライバに付属のマニュアルを参照してください。IQ テーブルは簡単に Notes 形式にマッピングできます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > リモート・データとバルク・オペレーション > リモート・データ・アクセスのサーバ・クラス > ODBC ベースのサーバ・クラス > サーバ・クラス odbc』の「Lotus Notes SQL 2.0」を参照してください。

Address サンプル・ファイルにアクセスできるように Sybase IQ を設定する手順は次のとおりです。

❖ Address サンプル・ファイルにアクセスできるように IQ を設定するには

1. NotesSQL ドライバを使用して、ODBC データ・ソースを作成します。

   データベースはサンプルの names ファイル *c:¥notes¥data¥names.nsf* になります。[特殊文字のマッピング] オプションをオンにしてください。この例では、データ・ソース名は *my_notes_dsn* です。

2. IQ サーバを作成します。

   ```
   CREATE SERVER names
   CLASS 'odbc'
   USING 'my_notes_dsn'
   ```

3 Person フォームを IQ テーブルにマッピングします。

```
CREATE EXISTING TABLE Person
AT 'names...Person'
```

4 テーブルをクエリします。

```
SELECT * FROM Person
```

第 6 章

スケジューリングとイベント処理によるタスクの自動化

この章について

この章では、Sybase IQ のスケジューリング機能とイベント処理機能を使用して、データベース管理やその他のタスクを自動化する方法について説明します。

内容

スケジューリングとイベント処理の概要

多くのデータベース管理タスクは体系的に実行するのが最も望ましい方法です。たとえば、データベースの適切な管理では、定期バックアップの実行が重要な位置を占めています。

SQL Anywhere のマニュアル『SQL Anywhere > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化』の「スケジュールとイベント処理の概要」を参照してください。

スケジューリングの概要

アクティビティをスケジューリングすると、あらかじめ設定しておいた時刻に確実にアクションが実行されるようにできます。スケジューリング情報とイベント・ハンドラは、どちらもデータベース自体の中に格納されます。

SQL Anywhere の製品マニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化』の「スケジュールの概要」を参照してください。

Sybase IQ の例

注意 例を見るには、Sybase IQ デモ・データベース iqdemo.db を使用できます。インストール情報については、「デモ・データベース」(xvi ページ) を参照してください。

```
Create table OrderSummary(c1 date, c2 int);
create event Summarize
schedule
start time '6:00 pm'
on ('Mon', 'Tue', 'Wed', 'Thu', 'Fri')
handler
begin
   insert into DBA.OrderSummary
   select max(OrderDate), count(*)
   from GROUPO.SalesOrders where OrderDate = current
date
end
```

スケジュールの定義

柔軟性を持たせるために、スケジュールの定義は複数のコンポーネントで構成されています。

SQL Anywhere の製品マニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > スケジュールの概要』の「スケジュールの定義」を参照してください。

イベントについて

データベース・サーバは数種類のシステム・イベントを追跡します。システム・イベントをチェックし、特定の**トリガ条件**が満たされていることがわかると、データベース・サーバはそれに対応するイベント・ハンドラをトリガします。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化』の「システム・イベントの概要」を参照してください。

「システム・イベントの選択」(133 ページ) と「イベントのトリガ条件の定義」(134 ページ) も参照してください。

システム・イベントの選択

Sybase IQ は複数のシステム・イベントを追跡します。各システム・イベントにはアクションを割り当てるためのフックがあります。データベース・サーバはイベントを追跡し、必要に応じて (イベント・ハンドラに定義されているとおりに) アクションを実行します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化』の「システム・イベントの概要」を参照してください。

イベントのトリガ条件の定義

各イベントの定義には、システム・イベントが関連付けられています。さらに、1 つまたは複数のトリガ条件も定義されています。システム・イベントのトリガ条件が満たされると、イベント・ハンドラがトリガされます。

注意 Sybase IQ のイベントに関連付けられているトリガ条件は、SQL Anywhere や Adaptive Server Enterprise のトリガ (ユーザが指定のテーブルで指定のデータ操作文を実行しようとすると自動的に実行される) と同じではありません。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > システム・イベントの概要』の「イベントのトリガ条件の定義」を参照してください。

Sybase IQ の例

注意 例を見るには、Sybase IQ デモ・データベース iqdemo.db を使用できます。インストール情報については、「デモ・データベース」(xvi ページ) を参照してください。

```
create event SecurityCheck
type ConnectFailed
handler
begin
declare num_failures int;
declare mins int;

insert into FailedConnections( log_time )
values ( current timestamp );

select count( * ) into num_failures
from FailedConnections
where log_time >= dateadd( minute, -5,
    current timestamp );
if( num_failures >= 3 ) then
    select datediff( minute, last_notification,
        current timestamp ) into mins
    from Notification;
```

```
    if( mins > 30 ) then
        update Notification
        set last_notification = current timestamp;
        call xp_sendmail( recipient='DBAdmin',
               subject='Security Check',"message"=
        'over 3 failed connections in last 5 minutes' )
    end if
end if
end
```

イベント・ハンドラについて

イベント・ハンドラは、イベントをトリガしたアクションとは別の接続を使って動作するので、クライアント・アプリケーションとの対話は行いません。イベント・ハンドラの実行にはイベント作成元のパーミッションが使用されます。

イベント・ハンドラの開発

イベント・ハンドラは (スケジューリングしたイベント用かシステム・イベントの処理用かにかかわらず)、複合文を含み、多くの点でストアド・プロシージャに似ています。ループや条件付き実行などを追加できます。また、Sybase IQ デバッガを使用してイベント・ハンドラをデバッグすることもできます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > システム・イベントの概要』の「イベント・ハンドラの開発」を参照してください。

詳細については、『リファレンス：ビルディング・ブロック、テーブル、およびプロシージャ』の「第 4 章 SQL 関数」の「EVENT_PARAMETER 関数 [システム]」を参照してください。

イベント処理の使用例については、『システム管理ガイド 第 1 巻』の「ユーザ ID とパーミッションの管理」の「IQ のユーザ・アカウントと接続の管理」を参照してください。

スケジュールとイベントの内容

ここでは、データベース・サーバがスケジュールとイベントの定義を処理する仕組みについて説明します。

データベース・サーバがシステム・イベントをチェックする仕組み

イベントは、CREATE EVENT 文に直接指定された**イベント・タイプ**に従って分類されます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > スケジュールとイベントの内部』の「データベース・サーバによるシステム・イベントのチェック」を参照してください。

予定時刻をデータベース・サーバがチェックする仕組み

予定イベント時刻の計算は、データベース・サーバ起動時と、スケジューリングしたイベント・ハンドラが完了するたびに行われます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > スケジュールとイベントの内部』の「データベース・サーバによるシステム・イベントのチェック」を参照してください。

イベント・ハンドラが実行される仕組み

イベント・ハンドラがトリガされると、内部接続が一時的に確立され、その接続でイベント・ハンドラが実行されます。イベント・ハンドラは、それ自体のトリガ元となった接続の上では実行されません。したがって、クライアント・アプリケーションとの対話を伴う MESSAGE ... TO CLIENT などの文をイベント・ハンドラ内に書いても無意味です。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > スケジュールとイベントの内部』の「イベント・ハンドラの実行」を参照してください。

スケジューリングとイベント処理のタスク

ここでは、スケジュールとイベントの自動化に関連するタスクについて説明します。

スケジュールやイベントのデータベースへの追加

スケジュールやイベントは、Sybase Central で SQL を使って追加できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > イベント処理タスク』の「データベースへのイベントの追加」を参照してください。

詳細については、『リファレンス：文とオプション』の「第 1 章 SQL 文」の「ALTER EVENT 文」を参照してください。

手動トリガ・イベントのデータベースへの追加

トリガ元となるスケジュールやシステム・イベントを持たないイベント・ハンドラは、手動でトリガしない限り実行されません。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > イベント処理タスク』の「データベースへの手動トリガ・イベントの追加」を参照してください。

イベントの変更方法については、『リファレンス：文とオプション』の「第 1 章 SQL 文」の「ALTER EVENT 文」を参照してください。

イベント・ハンドラのトリガ

どのイベント・ハンドラも、スケジュールやシステム・イベントによっても実行されますが、手動でもトリガできます。イベントの手動トリガは、イベント・ハンドラの開発時に役立つだけでなく、イベントによっては運用環境においても役立つ場合があります。たとえば、月次売り上げ報告の作成がスケジューリングされているとします。しかし、売り上げ報告が必要なのは月末だけではありません。別の目的で中間報告が必要な場合もあるでしょう。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > イベント処理タスク』の「イベント・ハンドラのトリガ」を参照してください。

詳細については、『リファレンス：文とオプション』の「第 1 章 SQL 文」の「TRIGGER EVENT 文」を参照してください。

イベント・ハンドラのデバッグ

デバッグはソフトウェア開発にはつきものの作業です。イベント・ハンドラは開発過程の中でデバッグできます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — データベース管理 > データベースの保守 > スケジュールとイベントの使用によるタスクの自動化 > イベント処理タスク』の「イベント・ハンドラのデバッグ」を参照してください。

イベントやスケジュールに関する情報の取得

Sybase IQ は、イベント、システム・イベント、スケジュールに関する情報を、SYSEVENT、SYSEVENTTYPE、SYSSCHEDULE の各システム・テーブルに保存しています。ALTER EVENT 文を使用してイベントを変更するとき、イベント名のほかに、オプションでスケジュール名を指定します。TRIGGER EVENT 文を使用してイベントをトリガするには、イベント名を指定します。

システム・テーブル SYSEVENT をクエリして、イベント名の一覧を表示できます。次に例を示します。

```
SELECT event_id, event_name FROM SYSEVENT
```

スケジュール名のリストは、システム・テーブル SYSSCHEDULE に対するクエリによって作成できます。次に例を示します。

```
SELECT event_id, sched_name FROM SYSSCHEDULE
```

イベントには固有のイベント ID が付いています。イベントと関連するスケジュールの対応付けには、SYSEVENT と SYSSCHEDULE の event_id カラムを使用します。

付 録　A

データベースでのロジックのデバッグ

この付録について

この付録では、SQL のストアド・プロシージャとイベント・ハンドラ、および Java のストアド・プロシージャの開発効率の向上に役立つ Sybase デバッガの使用法について説明します。

内容

データベースでのデバッグの概要

次のオブジェクトの開発過程ではデバッガを使用できます。

- SQL ストアド・プロシージャ、イベント・ハンドラ、ユーザ定義の関数
- データベース内の Java ストアド・プロシージャ

デバッガの機能

SQL ストアド・プロシージャ、トリガ、イベント・ハンドラ、ユーザ定義の関数の開発過程でデバッガを使用できます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、関数、トリガ、イベントのデバッグ』の「SQL Anywhere のデバッガの概要」を参照してください。

デバッガを使用するための要件

デバッガを使用するために必要なことを以下に示します。

- **パーミッション** DBA 権限を持っているか、SA_DEBUG グループでパーミッションを与えられている必要があります。このグループは、データベースの作成時に自動的にすべてのデータベースに追加されます。
- **Java クラスのソース・コード** アプリケーションのソース・コードをデバッガから参照できる必要があります。Java クラスの場合、ソース・コードはハード・ディスク上のディレクトリに格納されています。ストアド・プロシージャの場合、ソース・コードはデータベース内に格納されています。
- **コンパイル・オプション** Java クラスをデバッグするには、その内にデバッグ情報が含まれるように、コンパイルする必要があります。たとえば、Sun Microsystems の JDK コンパイラ *javac.exe* を使用する場合は、コマンド・ライン・オプションの -g を使用してコンパイルする必要があります。

注意 Sybase IQ のデモ・データベース *iqdemo.db* のインストール情報については、「デモ・データベース」(xvi ページ) を参照してください。

チュートリアル 1 ：デバッガの作業の開始

このチュートリアルでは、デバッガを起動してデータベースに接続する手順と、Java クラスをデバッグする方法について説明します。

レッスン 1 ：デバッガの起動とデータベースへの接続

このチュートリアルでは、デバッガを起動してデータベースに接続し、デバッグ用の接続に接続する手順を示します。そのために Sybase IQ サンプル・データベースを使用します。

デバッガの起動

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、関数、トリガ、イベントのデバッグ > チュートリアル：デバッガの使用開始』の「レッスン 1：データベースへの接続とデバッガの起動」を参照してください。

チュートリアル 2 ：ストアド・プロシージャのデバッグ

このチュートリアルでは、ストアド・プロシージャをデバッグする場合の作業例を示します。これは「チュートリアル 1 ：デバッガの作業の開始」(140 ページ) の続きです。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、関数、トリガ、イベントのデバッグ > チュートリアル：デバッガの使用開始』の「レッスン 2：ストアド・プロシージャのデバッグ」を参照してください。

チュートリアル 3 ：Java クラスのデバッグ

このチュートリアルでは、Interactive SQL (DBISQL) から JDBCExamples.Query() を呼び出して、デバッガでその実行を中断し、このメソッドのソース・コードをトレースします。

JDBCExamples.Query() メソッドは、サンプル・データベースに対して次のクエリを実行します。

```
SELECT ID, UnitPrice
FROM Products
```

そして、ループ処理によってクエリの結果セットのローの値を順次取得し、単価が最も高いローの値を返します。

クラスをデバッグするには、*javac* の -g オプションを指定してクラスをコンパイルする必要があります。サンプル・クラスは、デバッグが可能なようにコンパイルされています。

注意 Java の例を実行する場合は、Java サンプル・クラスをサンプル・データベースにインストールする必要があります。詳細については、「データベースの準備」(142 ページ) を参照してください。

データベースの準備

Java の例を実行する場合は、Java サンプル・クラスをサンプル・データベースにインストールする必要があります。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ— プログラミング > SQL Anywhere データ・アクセス API > SQL Anywhere JDBC ドライバ > JDBC を使用したデータへのアクセス』の「サンプルの準備」を参照してください。

デバッガでの Java ソース・コードの表示

❖ **デバッガで Java ソース・コードを表示するには**

デバッガはソース・コード・ファイル (拡張子が *.java* のファイル) が格納されている場所でファイルを調べます。

1 Sybase Central で、[モード] - [デバッグ] を選択します。

2 デバッグするユーザを選択する画面が表示されたら、「*」つまり、すべてのユーザを指定して [OK] をクリックします。

3 デバッガのインタフェースで、[デバッグ] - [Java ソース・コード・パスの設定] を選択します。

4 Sybase IQ インストール・ディレクトリの下にある *java* サブディレクトリのパスを入力します。*%IQDIR15%* に Sybase IQ をインストールした場合は次のように入力します。

```
%IQDIR15%¥java
```

5 [OK] をクリックして、ウィンドウを閉じます。

Java ソース・コードの検索

[Java ソース・コード・パス] ウィンドウには、デバッガが Java ソース・コードを検索するディレクトリのリストが表示されます。ソース・コードの検索には、パッケージを検索する場合の Java の規則が適用されます。デバッガは、現在 CLASSPATH に指定されているディレクトリでもソース・コードを検索します。

たとえば、Windows では、*%IQDIR15%¥java* と *c:¥Java¥src* をソース・パスとして追加した場合、デバッガが *iqdemo.Product* という製品を検索するとき、*%IQDIR15%¥asa¥java¥asademo¥Product.Java* と *c:¥Java¥src¥my¥iqdemo¥Product.Java* でソース・コードの検索が行われます。

ブレークポイントの設定

❖ **Java クラス内でのブレークポイントの設定**

Query() メソッドの最初の行にブレークポイントを設定できます。メソッドが呼び出されると、このブレークポイントで実行が停止します。

1. [ソース・コード] ウィンドウで、Query() メソッドの始めが見えるところまでスクロールします。これはクラスの終わりの近くにあり、次のような行で始まっています。

   ```
   public static int Query() {
   ```

2. メソッドの最初の行の先頭に表示されている緑色のインジケータをクリックします。クリックすると、インジケータの色が赤に変わります。メソッドの最初の行は、次のようになっています。

   ```
   int max_price = 0;
   ```

   クリックを繰り返すと、インジケータの状態が交互に切り替わります。ブレークポイントの設定の後には、Java クラスをコンパイルし直す必要はありません。

メソッドの実行

Interactive SQL (DBISQL) から Query() メソッドを呼び出して、その実行がブレークポイントで中断されることを実際に確認できます。

❖ **Interactive SQL からメソッドを呼び出す**

1. Interactive SQL を起動します。ユーザ ID として DBA、パスワードとして SQL を指定して、サンプル・データベースに接続します。

この操作で確立された接続が、デバッガの [接続] ウィンドウのリスト内に表示されます。

2 メソッドを呼び出すには、次のコマンドを Interactive SQL で入力します。

```
SELECT JDBCExamples.Query()
```

このクエリは完了しません。実行は、デバッガ内のブレークポイントで停止します。Interactive SQL では、[停止] ボタンがアクティブになっています。デバッガの [ソース] ウィンドウに表示される赤い矢印は、現在の行を示しています。

ここからは、デバッガでソース・コードを 1 ステップずつ実行しながら、デバッグ作業を進めることができます。

ソース・コードのステップ実行

説明に従って前項までの作業を行った場合、デバッガによる実行は JDBCExamples.Query() メソッドの最初の文で停止しているはずです。

例 次に、ソース・コードを 1 ステップずつ実行する場合の手順例を示します。

1 **次の行に進む** [実行] - [ステップ] を選択するか、または [F7] キーを押して、現在のメソッドの次の行に進みます。この手順を 2 ～ 3 回実行してみてください。

2 **選択した行まで実行する** 次に示した行をマウスで選択してから、[実行] - [選択した行まで実行する] を選択するか、または [F6] キーを押します。これにより、実行が選択行まで進み、そこで停止します。

```
max_price = price;
```

赤い矢印が選択行に移動します。

3 **ブレークポイントを設定して、そこまで実行する** 次に示した行 (292 行目) を選択してから [F9] キーを押して、その行にブレークポイントを設定します。

```
return max_price;
```

ブレークポイントが設定されていることを示すアスタリスクが、左側のカラムに表示されます。[F5] キーを押して、そのブレークポイントまで実行します。

4 **テスト**　いろいろな方法でソース・コードを 1 ステップずつ実行してみてください。実行を完了するには、[F5] キーを押します。

実行が完了すると、Interactive SQL のデータ・ウィンドウに、24 という値が表示されます。

5 **次のブレークポイントまで進む**　次のブレークポイントまで移動するには、[F5] キーを押します。

実行が完了すると、Interactive SQL のデータ・ウィンドウに、24 という値が表示されます。

[実行] メニューには、ソース・コードをステップ実行するための各種オプションがすべて表示されます。詳細については、デバッガのオンライン・ヘルプを参照してください。

変数の点検と修正

メソッド内で宣言されるローカル変数とデバッガ内のクラス静的変数の、両方の値を点検できます。

クラス・レベルの変数 (静的変数) を [デバッガ] ウィンドウに表示して、その値を点検できます。詳細については、デバッガのオンライン・ヘルプを参照してください。

コードを 1 ステップずつ実行しながらメソッド内のローカル変数の値を点検できるので、各ステップがどのように実行されているかがよくわかります。

注意　Java の例を実行する場合は、Java サンプル・クラスをサンプル・データベースにインストールする必要があります。詳細については、「データベースの準備」(142 ページ) を参照してください。

❖ ローカル変数の値を点検、修正する

1 JDBCExamples.Query メソッドの最初の行にブレークポイントを設定します。この行は次のようになっています。

```
int max_price = 0
```

2 Interactive SQL でこのメソッドを再度実行します。

```
SELECT JDBCExamples.Query()
```

クエリはブレークポイントまで実行されされます。

3 [F7] キーを押して、次の行に進みます。これで max_price 変数が宣言され、0 に初期化されます。

4 [ローカル] ウィンドウが表示されていない場合は、[ウィンドウ] - [ローカル変数] を選択して、そのウィンドウを表示します。

[ローカル] ウィンドウに複数のローカル変数が表示されますが、max_price の値は 0 になっています。その他の変数については、すべて `variable not in scope` と表示されます。これは、それらの変数がまだ初期化されていないことを意味します。

5 [ローカル] ウィンドウの [値] カラムで max_price に対するエントリをダブルクリックして、max_price の値を 45 に変更します。

45 という値は、他のどの製品よりも高い価格です。したがって、クエリが最高価格として返す値は 24 ではなく、45 になるはずです。

6 [ソース] ウィンドウで [F7] キーを繰り返し押して、コードの実行を進めます。このとき、各変数の値が [ローカル] ウィンドウに表示されます。stmt 変数と result 変数に値が表示されるまで、ステップ実行を続けてください。

7 result オブジェクトの横のアイコンをクリックして、そのオブジェクトの表示を拡張します。そのオブジェクトの行にカーソルを置いて [Enter] キーを押す方法でも表示を拡張できます。これで、そのオブジェクトの各フィールドの値が表示されます。

8 変数を点検、修正する手順を十分にテストしたら、[F5] キーを押してクエリの実行を完了させ、チュートリアルを終了します。

ブレークポイントの使用

ブレークポイントはデバッガがソース・コードの実行をいつ停止するかを制御します。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、関数、トリガ、イベントのデバッグ』の「ブレークポイントの活用」を参照してください。

変数の使用

デバッガでは、コードのステップ実行時に変数の動作を表示したり編集したりできます。デバッガには [デバッガの詳細] ペインがあり、そこにストアド・プロシージャ内で使用されている各種の変数が表示されます。[デバッガの詳細] ペインは Sybase Central がデバッグ・モードで実行されているとき、画面下部に表示されます。

SQL Anywhere のマニュアル『SQL Anywhere 11.0.1 > SQL Anywhere サーバ — SQL の使用法 > ストアド・プロシージャとトリガ > プロシージャ、関数、トリガ、イベントのデバッグ』の「変数の活用」を参照してください。

デバッガの基本操作

作業内容	操作手順
Java ソース・ファイルの検索パスにディレクトリを追加する。	[デバッグ] - [Java ソース・コード・パスの設定]
デバッガによる接続の取得を有効にする。	[接続] - [取得を有効化]
デバッガを終了する。	[ファイル] - [終了]
選択されているウィンドウ内で文字列を検索する。	[検索] - [検索]
検索対象ワードの大文字と小文字の区別を無視する。	[検索] - [大文字と小文字を区別しない]
プロシージャ・デバッガの設定をファイルからロードする。	[設定] - [ファイルからロード]
新しい接続としてデータベースにログインする。	[接続] - [ログイン]
データベースからログアウトする。	[接続] - [ログアウト]
デバッグ・スクリプトを実行する。	[ファイル] - [スクリプトの実行] 詳細については、「デバッガ・スクリプトの作成」(148 ページ) を参照してください。
プログラムを 1 行ずつ実行する (プロシージャ内の処理も 1 行ずつ実行する)。	[実行] - [ステップ・イン]

作業内容	操作手順
プログラムを 1 行ずつ実行する (プロシージャ内の処理は行ごとに扱わず、一括して実行する)。プロシージャ内に設定されているブレークポイントは無視される。	[実行] - [ステップ]
プログラムを実行する。プログラムの実行がブレークポイントで停止している場合は、そのブレークポイントから実行を再開する。	[実行] - [実行]
現在のプロシージャ/メソッドから戻るまでプログラムを実行する。	[実行] - [ステップ・アウト]
プロシージャ・デバッガの設定を保存する。	[設定] - [保存]
プロシージャ・デバッガの設定をファイルに保存する。	[設定] - [ファイルに保存]
Java ソース・ファイルが格納されているパスを指定する。	[ファイル] - [ソース・パスを編集]
プロシージャまたはクラスのソース・コードを表示する。	[ファイル] - [開く]

デバッガ・スクリプトの作成

デバッガでは、Java プログラミング言語を使用してスクリプトを作成できます。スクリプトは sybase.asa.procdebug.DebugScript クラスを拡張した Java クラスです。

デバッガでスクリプトを実行すると、そのクラスがロードされ、その run メソッドが呼び出されます。run メソッドの第 1 パラメータは、そのクラスのインスタンスを示すポインタです。このインタフェースを使用すると、デバッガに対する対話的操作と制御を実行できます。

デバッガ・ウィンドウは、「sybase.asa.procdebug.IDebugWindow interface」(153 ページ) によって表されます。

スクリプトをコンパイルするには、次のようなコマンドを実行します。

```
javac -classpath
%asany%/procdebug/ProcDebug.jar;%classpath%
myScript.Java.
```

sybase.asa.procdebug.DebugScript クラス

DebugScript クラスの定義は、次のとおりです。

```
// すべてのデバッグ・クラスがこのクラスを継承すること

package sybase.asa.procdebug;

abstract public class DebugScript
{
   abstract public void run( IDebugAPI db, String args[]
);
    /*
        run メソッドはデバッガによって呼び出される
        - args にはコマンドライン引数が入る
    */

   public void OnEvent( int event ) throws DebugError {}
    /*
    - デバッグ・イベントを処理するために次のメソッドをオーバーライド
する。
    - 注：このメソッドの呼び出しには次のメソッドを呼び出す必要あり。
       DebugAPI.AddEventHandler( this );
    */

}
```

sybase.asa.procdebug.IDebugAPI インタフェース

IDebugAPI インタフェースの定義は、次のとおりです。

```
package sybase.asa.procdebug;
import java.util.*;
public interface IDebugAPI

{
    // メニュー項目をシミュレートする

    IDebugWindow MenuOpenSourceWindow() throws
DebugError;
    IDebugWindow MenuOpenCallsWindow() throws
DebugError;
    IDebugWindow MenuOpenClassesWindow() throws
DebugError;
    IDebugWindow MenuOpenClassListWindow() throws
DebugError;
```

```
    IDebugWindow MenuOpenMethodsWindow() throws
DebugError;
    IDebugWindow MenuOpenStaticsWindow() throws
DebugError;
    IDebugWindow MenuOpenCatchWindow() throws
DebugError;
   IDebugWindow MenuOpenProcWindow() throws DebugError;
    IDebugWindow MenuOpenOutputWindow() throws
DebugError;
    IDebugWindow MenuOpenBreakWindow() throws
DebugError;
    IDebugWindow MenuOpenLocalsWindow() throws
DebugError;
    IDebugWindow MenuOpenInspectWindow() throws
DebugError;
    IDebugWindow MenuOpenRowVarWindow() throws
DebugError;
    IDebugWindow MenuOpenQueryWindow() throws
DebugError;
    IDebugWindow MenuOpenEvaluateWindow() throws
DebugError;
    IDebugWindow MenuOpenGlobalsWindow() throws
DebugError;
    IDebugWindow MenuOpenConnectionWindow() throws
DebugError;
    IDebugWindow MenuOpenThreadsWindow() throws
DebugError;
    IDebugWindow GetWindow( String name ) throws
DebugError;

    void MenuRunRestart() throws DebugError;
    void MenuRunHome() throws DebugError;
    void MenuRunGo() throws DebugError;
    void MenuRunToCursor() throws DebugError;
    void MenuRunInterrupt() throws DebugError;
    void MenuRunOver() throws DebugError;
    void MenuRunInto() throws DebugError;
    void MenuRunIntoSpecial() throws DebugError;
    void MenuRunOut() throws DebugError;
    void MenuStackUp() throws DebugError;
    void MenuStackDown() throws DebugError;
    void MenuStackBottom() throws DebugError;
    void MenuFileExit() throws DebugError;
   void MenuFileOpen( String name ) throws DebugError;
    void MenuFileAddSourcePath( String what ) throws
DebugError;
```

```
    void MenuSettingsLoadState( String file ) throws
DebugError;
    void MenuSettingsSaveState( String file ) throws
DebugError;
    void MenuWindowTile() throws DebugError;
    void MenuWindowCascade() throws DebugError;
    void MenuWindowRefresh() throws DebugError;
    void MenuHelpWindow() throws DebugError;
    void MenuHelpContents() throws DebugError;
    void MenuHelpIndex() throws DebugError;
    void MenuHelpAbout() throws DebugError;
    void MenuBreakAtCursor() throws DebugError;
    void MenuBreakClearAll() throws DebugError;
    void MenuBreakEnableAll() throws DebugError;
    void MenuBreakDisableAll() throws DebugError;
    void MenuSearchFind( IDebugWindow w, String what )
throws DebugError;
    void MenuSearchNext( IDebugWindow w ) throws
DebugError;
    void MenuSearchPrev( IDebugWindow w ) throws
DebugError;
    void MenuConnectionLogin() throws DebugError;
    void MenuConnectionReleaseSelected() throws
DebugError;

    // 出力ウィンドウ
    void OutputClear();
    void OutputLine( String line );
    void OutputLineNoUpdate( String line );
    void OutputUpdate();



    // Java ソースの検索パス

   void SetSourcePath( String path ) throws DebugError;
    String GetSourcePath() throws DebugError;

    // java 例外を捕らえる
    Vector GetCatching();
    void Catch( boolean on, String name ) throws
DebugError;

    // データベース接続
    int  ConnectionCount();
    void ConnectionRelease( int index );
    void ConnectionAttach( int index );
```

```
    String ConnectionName( int index );
    void ConnectionSelect( int index );

    // データベースにログインする
    boolean LoggedIn();
    void Login( String url, String userId, String
password, String userToDebug ) throws DebugError;
    void Logout();

    // キーボードとマウスの動作をシミュレートする
   void DeleteItemAt( IDebugWindow w, int row ) throws
DebugError;
  void DoubleClickOn( IDebugWindow w, int row ) throws
DebugError;

    // ブレークポイント
    Object BreakSet( String where ) throws DebugError;
    void  BreakClear( Object b ) throws DebugError;
  void BreakEnable( Object b, boolean enabled ) throws
DebugError;
    void BreakSetCount( Object b, int count ) throws
DebugError;
    int  BreakGetCount( Object b ) throws DebugError;
   void BreakSetCondition( Object b, String condition
) throws DebugError;
    String BreakGetCondition( Object b ) throws
DebugError;
    Vector GetBreaks() throws DebugError;

    // スクリプトの作成
    void RunScript( String args[] ) throws DebugError;
    void AddEventHandler( DebugScript s );
    void RemoveEventHandler( DebugScript s );

    // その他
    void EvalRun( String expr ) throws DebugError;
    void QueryRun( String query ) throws DebugError;
    void QueryMoreRows() throws DebugError;
    Vector GetClassNames();
    Vector GetProcedureNames();
  Vector WindowContents( IDebugWindow window ) throws
DebugError;
    boolean AtBreak();
    boolean IsRunning();
    boolean AtStackTop();
    boolean AtStackBottom();
```

```
    void SetStatusText( String msg );
    String GetStatusText();
    void WaitCursor();
    void OldCursor();
    void Error( Exception x );
    void Error( String msg );
    void Warning( String msg );
    String Ask( String title );
    boolean MenuIsChecked( String cmd );
    void MenuSetChecked( String cmd, boolean on );
    void AddInspectItem( String s ) throws DebugError;

    // DebugScript.OnEvent パラメータ用の定数
    public static final int EventBreak = 0;
    public static final int EventTerminate = 1;
    public static final int EventStep = 2;
    public static final int EventInterrupt = 3;
    public static final int EventException = 4;
    public static final int EventConnect = 5;
};
```

sybase.asa.procdebug.IDebugWindow interface

IDebugWindow インタフェースの定義は、次のとおりです。

```
// このインタフェースはデバッグ・ウィンドウを表す。
package sybase.asa.procdebug;
public interface IDebugWindow
{
    public int GetSelected();
    /*
        現在選択されているロー、選択なしの場合は -1
    */

    public boolean SetSelected( int i );
    /*
        現在選択されているローを設定。 i < 0 または i > ロー数の
場合は無視される。
    */

    public String StringAt( int row );
    /*
        そのウィンドウの N 番目のローを表す文字列を取得。 row >
ロー数の場合は null を返す。
    */
```

```
    public java.awt.Rectangle GetPosition();
    public void SetPosition( java.awt.Rectangle r );
    /*
        フレーム内のウィンドウの位置を取得/設定する
    */

    public void Close();
    /*
        ウィンドウを閉じる （破棄する）
    */
}
```

索引

G

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

た

て

と

は

ひ

ふ

ほ

ま

む

め

ゆ

よ

ら

り

れ

ろ