SQL Self-Study Guide

IBM Red Brick Warehouse

バージョン 6.2
2002 年 9 月
Part No. 000-9047-8

注意 :
本書の情報および該当製品をご利用になる前に、付録「特記事項」の内容をお読みください。

目次

まえがき

この章について

この章では、本書の概要と表記法について説明します。

まえがき

例題に基づいて SQL を復習し、RISQL 拡張機能、マクロ関数、**Aroma** のサンプルデータベースを紹介します。

対象読者

本書では、以下のユーザを対象としています。

- データベース ユーザ
- データベース管理者
- データベース サーバ管理者
- データベース アプリケーション プログラマー
- データベース設計者
- データベース デザイナー
- データベース開発者
- バックアップ オペレータ
- パフォーマンス エンジニア

本書では、読者に以下の知識や経験があることを前提としています。

- 使用しているコンピュータ、オペレーティング システム、およびオペレーティング システムが提供するユーティリティに対する実務知識
- リレーショナル データベースの使用経験、またはデータベースの概念に関する知識
- コンピュータ プログラミングの経験
- データベース サーバ管理、オペレーティング システム管理、またはネットワーク管理の経験

ソフトウェア要件

本書では、IBM Red Brick Warehouse Version 6.2 をデータベース サーバとして使用していることを前提としています。

IBM Red Brick Warehouse には、コーヒーと紅茶を取り扱う架空の会社の販売データをおさめた Aroma というデータベースが添付されています。このデータベースでは、Aroma Coffee and Tea Company という企業の毎日の販売業務を管理しています。このデータベースのディメンジョン モデルは、1 つのファクト テーブルと、それに付属する複数のディメンジョン テーブルから成っています。

サンプル データベースの作成とデータのロードの詳細は、『Administrator's Guide』を参照してください。データベースとそのデータ内容の詳細は、第 1 章「Aroma—意思決定支援のためのデータベース」を参照してください。

サンプル データベースのインストール スクリプトは redbrick_dir/sample_input ディレクトリにあります。redbrick_dir は、使用しているシステムの IBM Red Brick Warehouse ディレクトリをさします。

本書の表記法

ここでは、このマニュアルで使用される、以下の表記規則について説明します。表記規則を覚えておくと、このマニュアル、およびほかのマニュアルの内容を理解するのに役立ちます。

以下のような表記規則があります。

- 文字の表記規則
- 構文の規則

文字の表記規則

このマニュアルは、新しい用語、画面表示、コマンド構文などを表記するのに、以下の規則を使用します。

表記規則	意味
KEYWORD	プログラミング言語の文中では、主要な要素 (キーワード) は、すべて大文字のセリフ フォントで表記されます。
`Computer`	製品の表示情報やユーザの入力情報はモノスペース フォントで表記されます。
♦	この記号は、製品やプラットフォームなどに特有の情報の終わりを表します。
→	この記号は、メニュー項目を表します。たとえば、"[ツール] →[オプション] を選択します。" は、[ツール] メニューの [オプション] を選択することを意味します。

ヒント：コマンドの入力、または実行の指示がある場合、入力後に ENTER キーを押してください。コマンド以外の文字の入力やほかのキーを押す指示がある場合、ENTER キーを押す必要はありません。

構文の規則

本書では、オペレーティング システム コマンドの構文に以下の規則を採用しています。

コマンドの構成要素	例	表記規則
値および パラメータ	<table_name>	適切な名前、値、式に置き換える項目は、山形かっこ < > で囲んで表記します。
オプション項目	[ ]	オプション項目は、角かっこで囲みます。角かっこは入力しません。
選択肢	ONE |TWO	選択肢は縦棒で区切ります。必要に応じて、いずれか 1 つを選択できます。
必須選択肢	{ONE|TWO}	必須選択肢は、中かっこで囲みます。いずれか 1 つを選択してください。中かっこは入力しません。
デフォルト値	ONE|TWO	デフォルト値は、下線を付けて表記します。ただし、図の部分では太字で表記します。
繰り返し項目	name, ...	繰り返し可能な項目は、後にカンマと省略記号を記述します。各項目は、カンマで区切ります。
記述記号	( ) , ; .	かっこ、カンマ、セミコロン、ピリオドは記述記号です。記述されているとおりに使用してください。

テクニカル サポート

IBM Red Brick Warehouse に関する技術上の質問があり、その対処法が該当するマニュアルに記述されていない場合は、以下の IBM カスタマ サポートまでご連絡ください。

電話番号　1-800-274-8184 または 1-913-492-2086
(中央標準時間、月曜日から金曜日の午前 7 時 ～ 午後 7 時)

Web サイト　http://www-3.ibm.com/software/data/informix/support/

新しいケース (問題点)

新しいケースを記録する場合は、次の情報を報告してください。

- IBM Red Brick Warehouse のバージョン
- プラットフォームおよびオペレーティング システムのバージョン
- IBM Red Brick Warehouse またはオペレーティング システムから返されたエラー メッセージ
- エラー メッセージが表示される前に実行したコマンドや操作など、問題についての詳しい説明
- エラー メッセージが表示される前に行われた IBM Red Brick Warehouse またはオペレーティング システムの構成変更のリスト

クライアント / サーバの接続に関する問題の場合は、さらに次の情報も報告してください。

- 使用しているクライアント ツールの名前とバージョン
- 該当する場合、Red Brick ODBC Driver または Red Brick JDBC Driver のバージョン
- 使用しているクライアント ネットワークまたは TCP/IP スタックの名前とバージョン
- クライアント アプリケーションから返されたエラー メッセージ
- サーバおよびクライアント ロケール指定

既存のケース

問題のケースを記録するか、最初に対応したサポート エンジニアが必ずケース番号を登録します。この番号によって、それぞれの問題を解決する間に行われた作業をすべて把握することができます。既存のケースの状況について知りたい場合は、ケース番号をお知らせください。

トラブルシューティングのヒント

ささいなことでも、再現可能な問題を具体的に報告することによって、ケース解決までの時間を短縮できる場合もあります。問題の原因をたくさん見つけることができれば、それだけ早くサポート エンジニアが問題を解決することができます。

- SQL クエリに関する問題の場合は、ステートメントで問題の原因となった部分が発見されるまで、列または関数の削除や、WHERE 句、ORDER BY 句、または GROUP BY 句の再指定を行ってください。
- Table Management Utility ロードに関する問題の場合は、ソース ファイルとターゲット テーブルとのマッピングでデータ型に互換性があるかどうか確認してください。少量のテスト データをロードして、ボリュームとデータ フォーマットのいずれに関連した問題であるかを特定してください。
- 接続上の問題の場合は、クライアントからホストに **ping** コマンドを実行して、ネットワークが立ち上がって稼動していることを確認してください。可能な場合は、別のクライアント ツールを使用しても同じ問題が発生するかどうか確認してください。

関連文献

IBM Red Brick Warehouse のマニュアルには、以下の文書が含まれています。

マニュアル	内容
『Administrator's Guide』	ウェアハウスのアーキテクチャやサポートされるスキーマなど、データベースに関連した基本概念のマニュアルです。データベースのインプリメントや保守の手順について説明しています。システム テーブルとコンフィグレーション ファイルの説明も含まれています。
『Client Installation and Connectivity Guide』	ODBC、Red Brick JDBC Driver、RISQL エントリ ツール、RISQL レポータ をクライアント システムにインストールして構成するためのガイドです。C および C++ アプリケーション用 ODBC 製品と Java アプリケーション用 JDBC 製品を使用して、IBM Red Brick Warehouse にアクセスする方法を説明しています。
『IBM Red Brick Vista User's Guide』	IBM Red Brick Vista の集約計算とマネジメントのシステムについて説明しています。集約を使用してクエリを自動的にリライトすることによって Vista クエリ パフォーマンスを向上させる方法、毎日集められるデータをもとに最高の集約セットを作るよう推奨すること、詳細テーブルが更新されるときに集約テーブルがどのように保守されるかを説明しています。
『Installation and Configuration Guide』	IBM Red Brick Warehouse のインストールと環境設定に関する説明書と、プラットフォーム別マニュアルです。UNIX および Linux ベースのシステム用と、Windows ベースのシステム用があります。

(1 / 2)

マニュアル	内容
『Messages and Codes Reference Guide』	IBM Red Brick Warehouse 製品が出力するすべての状態情報、警告、エラー メッセージとその考えられる原因、対処方法が示されています。ログ ファイルに書き込まれるイベント ログ メッセージも記述されています。
『Query Performance Guide』	クエリ パフォーマンスの決定要因と、最適なクエリ パフォーマンスを得るためのデータベースのチューニング方法について説明しています。Red Brick ツール (SET STATS、Dynamic Statistic Tables : 動的統計テーブル、EXPLAIN、および Query Performance Monitor) を使用してクエリ パフォーマンスを評価する方法についても、例を挙げて説明しています。
『リリースノート』	マニュアルの印刷後に判明した現リリースに関する情報が含まれます。
『RISQL Entry Tool and RISQL Reporter User's Guide』	SQL 文の入力に使用するコマンドライン ツールである RISQL エントリ ツール と、RISQL エントリ ツール にレポートのフォーマット設定機能を付加した RISQL レポータ の詳細なガイドです。
『SQL Reference Guide』	Red Brick SQL のインプリメントと RISQL (IBM Red Brick Warehouse データベースのための拡張機能) に関する詳細な言語リファレンスです。
本書	例題に基づいて SQL を復習し、RISQL 拡張機能、マクロ関数、**Aroma** のサンプル データベースを紹介します。
『Table Management Utility Reference Guide』	データのロード、管理、バックアップに関連した機能をまとめた、Table Management Utility について説明しています。データのコピーと **rb_cm** コピー管理ユーティリティについても説明しています。

(2 / 2)

このほか、以下の参考資料も必要に応じて参照してください。

- SQL に関する入門書
- リレーショナル データベースの入門書
- ご使用のハードウェア プラットフォームとオペレーティング システムのマニュアル

その他のマニュアル

上記以外の情報は、以下のマニュアルを参照してください。

- オンライン マニュアル
- 印刷マニュアル
- オンライン ヘルプ

オンライン マニュアル

Red Brick 製品には、各種の IBM Red Brick Warehouse マニュアルを電子フォーマットで収録した CD-ROM が同梱されています。収録されているマニュアルは、システムにインストールして使用することも、CD-ROM から直接アクセスすることも可能です。

印刷マニュアル

印刷マニュアルを注文する場合は、担当販売員までご連絡ください。

オンライン ヘルプ

IBM はグラフィカル ユーザ インタフェース (GUI) を用いたオンライン ヘルプを提供します。これにより、各インタフェースや実行する関数についての情報を参照することができます。オンライン ヘルプを表示するには、GUI のヘルプ機能を利用してください。

ご意見 / ご提案

今後のマニュアルの品質向上を図るため、修正すべき箇所や有用な情報をご提案ください。その際、以下の情報をご提出ください。

- マニュアルの正式名とバージョン
- マニュアルに関するご意見
- 氏名、住所、電話番号

連絡先電子メール アドレス :

comments@vnet.ibm.com

このメール アドレスは、マニュアル内の記述誤りや誤植をご報告いただくために用意されています。代わりに、以下の Web サイトのフォームに記述して送信していただくことも可能です。

http://www.ibm.com/software/data/rcf/

技術上の問題に関する緊急な問い合わせについては、IBM テクニカル サポートまでご連絡ください。

第 1 章

Aroma—意思決定支援のためのデータベース

この章について

本書では、標準 SQL と、拡張 SQL である RISQL を使い、ビジネスに関する一般的な問い合わせをデータベースクエリとして表現する方法を説明します。また、クエリやクエリの一部を繰り返し実行するときに、マクロを使って、クエリの記述を簡略化する方法も説明します。

本書および IBM Red Brick Warehouse のほとんどのマニュアルでは、Aroma というデータベースを使った例題を記述します。**Aroma** は、全米の小売店で販売されているコーヒーや紅茶の販売データを格納したサンプル データベースです。各例題は、3 つの部分で構成されています。

- 日常的な言葉で表したビジネスに関する問い合わせ
- その問い合わせを SQL で表した SELECT 文
- データベースから返される結果をまとめた表

通常、**Aroma** は Red Brick ソフトウェアのインストール時にインストールされます。サンプル クエリを実行する場合は、**Aroma** データベースにアクセスする方法を各サイトのシステム管理者に確認してください。

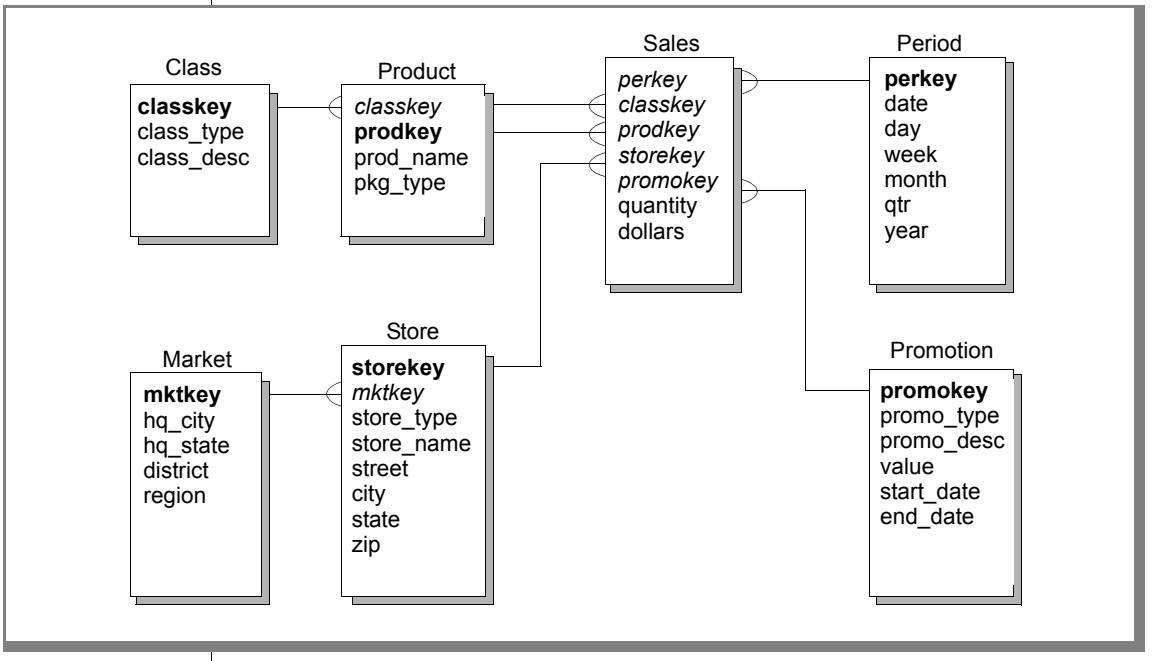
この章では、基本的な **Aroma** データベースのテーブルを紹介し、各テーブルのデータをリンクするプライマリ キーと外部キーの関係を簡単に説明します。

この章では、**Aroma** データベースや Red Brick データベースが迅速かつ効率的に応答できる質問もいくつか説明します。

Aroma データベースの販売スキーマ

本書の例はほとんどの場合、Aroma Coffee and Tea Company が所有する店舗の日別販売売上を格納する、基本 **Aroma** データベースのデータを使用しています。販売スキーマは、4 つの主要なディメンジョン テーブル (**Period**: 会計期間、**Product**: 製品、**Store**: 店舗、**Promotion**: 販売促進活動)、ファクト テーブル (**Sales**)、および 2 つのアウトボード テーブル (**Class**、**Market**) で構成されます。

次の図は、この基本スキーマを示したものです。



この図で 3 つの分岐線は、2 つのテーブル間に 1 対多の関係があることを表しています。たとえば、**Period** テーブルの **Perkey** 列のある値が、Period テーブルでは 1 回しか現れないのに、Sales テーブルでは何度も現れることがあるということを示しています。太字の列名はプライマリ キー列、イタリックの列名は外部キー列を示します。イタリックで示す列名は、プライマリ キー列であると同時に外部キー列でもあるという意味です。

この章では、各テーブルのサンプル データに基づき、プライマリ キーと外部キーの関係がどのような意味をもつかを説明します。

基本の Aroma スキーマ

意思決定支援データベースの各テーブルと列には、使い慣れたビジネス用語で名前を付けることで、スキーマをわかりやすく使いやすくします。良く設計されたスキーマは、アプリケーション開発担当者やエンド ユーザに次のような利点があります。

- ビジネスに関する質問を、SQL クエリとして表すのが簡単になる。
- クエリが短時間で実行でき、整合性のとれた答えが返される。

Aroma の販売スキーマは、どちらの条件も兼ね備えています。**Sales** テーブルには、ビジネスに関する日々の事実 (ファクト) が格納され、**Store**、**Period**、**Product**、**Promotion** の各テーブルには、ビジネスの各種ディメンジョンつまり特性が格納されます。**Class** テーブルと **Market** テーブルは、製品や店舗に関する詳細情報を格納します。

本書の例題は、これら 7 つの基本テーブルが構成する単純なスター スキーマを主に使用しますが、**Aroma** データベースには集約された Sales テーブルや、複雑な設計の仕入れスキーマも含まれています。詳細は、付録 A「Aroma データベース詳細」を参照してください。

重要： **Aroma** データベースは、事前に定義された集約テーブルを含みません。集約クエリの性能を高める Vista クエリ リライト システムの使用法の詳細については、『IBM Red Brick Vista User's Guide』を参照してください。

Period、Product、Class のディメンジョン

Period テーブル

Period テーブルの先頭の数行を示します。プライマリ キー列は、**Perkey** 列です。

Perkey	Date	Day	Week	Month	Qtr	YEAR
1	1998-01-01	TH	1	JAN	Q1_98	1998
2	1998-01-02	FR	1	JAN	Q1_98	1998
3	1998-01-03	SA	1	JAN	Q1_98	1998
4	1998-01-04	SU	2	JAN	Q1_98	1998
5	1998-01-05	MO	2	JAN	Q1_98	1998
6	1998-01-06	TU	2	JAN	Q1_98	1998
...						

Product テーブルと Class テーブル

Product テーブルの先頭の数行を示します。プライマリ キーは、**Classkey** の値と **Prodkey** の値を組み合わせたものになります。

Classkey	Prodkey	Prod_Name	Pkg_Type
1	0	Veracruzano	No pkg
1	1	Xalapa Lapa	No pkg
1	10	Colombiano	No pkg
1	11	Espresso XO	No pkg

(1 / 2)

Classkey	Prodkey	Prod_Name	Pkg_Type
1	12	La Antigua	No pkg
1	20	Lotta Latte	No pkg
...			

(2 / 2)

ディメンジョン テーブルの外部キー列が、ほかのディメンジョン テーブルを参照する場合、参照先のテーブルは「アウトボード テーブル」、または「アウトリガー テーブル」と呼ばれます。たとえば、**Product** テーブルの **Classkey** 列は、**Class** テーブルを参照する外部キーです。

Class テーブルの先頭の数行を示します。

Classkey	Class_Type	Class_Desc
1	Bulk_beans	Bulk coffee products
2	Bulk_tea	Bulk tea products
3	Bulk_spice	Bulk spices
4	Pkg_coffee	Individually packaged coffee products
5	Pkg_tea	Individually packaged tea products
6	Pkg_spice	Individually packaged spice products

Store、Market、Promotion のディメンジョン

ディメンジョン テーブルには、データベースに問い合わせをする際にデータ アナリストが使用する属性が含まれています。たとえば、**Store** テーブルには店舗の名前と所在地、**Product** テーブルには製品やパッケージ関連の情報、**Period** テーブルには月、四半期、年度の値が格納されています。どのテーブルにも、1 つまたは複数の列で構成されるプライマリ キーが含まれています。プライマリ キーの値により、テーブルの各行が一意に識別されます。

Store テーブルと Market テーブル

Store テーブルの先頭の数行を示します。1 ページに収まるように切り捨てられた列もあります。プライマリ キー列は、**Storekey** です。**Mktkey** は、**Market** テーブルを参照する外部キーです。

Storekey	Mktkey	Store_Type	Store_Name	STREET	CITY	STATE	ZIP
1	14	Small	Roasters, Los Gatos	1234 University Ave	Los Gatos	CA	95032
2	14	Large	San Jose Roasting	5678 Bascom Ave	San Jose	CA	95156
3	14	Medium	Cupertino Coffee	987 DeAnza Blvd	Cupertino	CA	97865
4	3	Medium	Moulin Rouge	898 Main Street	New Orleans	LA	70125
5	10	Small	Moon Pennies	98675 University	Detroit	MI	48209
6	9	Small	The Coffee Club	9865 Lakeshore Bl	Chicago	IL	06060
...							

Market テーブルの先頭の数行を示します。

Mktkey	Hq_city	Hq_state	District	Region
1	Atlanta	GA	Atlanta	South
2	Miami	FL	Atlanta	South
3	New Orleans	LA	New Orleans	South
4	Houston	TX	New Orleans	South
5	New York	NY	New York	North
...				

Promotion テーブル

Promotion テーブルの先頭の数行を示します。プライマリ キー列は **Promokey** です。

Promokey	Promo_Type	Promo_Desc	Value	Start_Date	End_Date
0	1	No promotion	0.00	9999-01-01	9999-01-01
1	100	Aroma catalog coupon	1.00	1998-01-01	1998-01-31
2	100	Aroma catalog coupon	1.00	1998-02-01	1998-02-28
3	100	Aroma catalog coupon	1.00	1998-03-01	1998-03-31
4	100	Aroma catalog coupon	1.00	1998-04-01	1998-04-30
5	100	Aroma catalog coupon	1.00	1998-05-01	1998-05-31
...					

Sales テーブル

Sales テーブルの先頭 20 行を示します。

Perkey	Classkey	Prodkey	Storekey	Promokey	Quantity	Dollars
2	2	0	1	116	8	34.00
2	4	12	1	116	9	60.75
2	1	11	1	116	40	270.00
2	2	30	1	116	16	36.00
2	5	22	1	116	11	30.25
2	1	30	1	116	30	187.50
2	1	10	1	116	25	143.75
2	4	10	2	0	12	87.00
2	4	11	2	0	14	115.50

(1 / 2)

Perkey	Classkey	Prodkey	Storekey	Promokey	Quantity	Dollars
2	2	22	2	0	18	58.50
2	4	0	2	0	17	136.00
2	5	0	2	0	13	74.75
2	4	30	2	0	14	101.50
2	2	10	2	0	18	63.00
2	1	22	3	0	11	99.00
2	6	46	3	0	6	36.00
2	5	12	3	0	10	40.00
2	1	11	3	0	36	279.00
2	5	1	3	0	11	132.00
2	5	10	3	0	12	48.00
...						

(2 / 2)

プライマリ キー列は、次の 5 つの列の値を組み合わせたものです。

```
perkey, classkey, prodkey, storekey, promokey
```

Sales ファクトについて

Sales テーブルは、ファクト テーブルです。つまり、Sales テーブルのデータは参照先テーブルに定義されたビジネス属性によって簡単にアクセスでき、各属性に関する大量の統計情報を格納しているテーブルです。**Sales** テーブルは、**Aroma** データベースの最大のテーブルで、「セグメント」と呼ばれる 2 つのデータベース格納領域にデータが分割されています。セグメントについては、『Administrator's Guide』を参照してください。

IBM Red Brick Warehouse では、ユーザがなじみやすいビジネス ディメンジョンに基づき、クエリの作成者がファクト テーブルの行に素早くアクセスできるようになっています。たとえば、San Jose Roasting Company の La Antigua コーヒーの 1999 年 1 月 31 日の売上を知りたければ、年月日、製品名、店舗名という 3 つのディメンジョンを指定すれば、ウェアハウス サーバが要求内容を即座に抽出します。

複合プライマリ キー

Sales テーブルには、複合プライマリ キーが格納されています。5 つの列のそれぞれが、ほかのテーブルのプライマリ キーを参照する外部キーです。

```
perkey, classkey, prodkey, storekey, promokey
```

上記の各プライマリ キーは、**Period**、**Product**、**Store**、および **Promotion** の各ディメンジョンに Sales データを結びつけます。このリンクにより、ある都市のある日におけるある製品の売上金額と販売数量を素早く簡単にデータベースから抽出できます。

一般的な質問

ビジネスに関する一般的な質問には以下のものがあります。

低難易度

- 昨年の San Jose における Lotta Latte ブランド コーヒーの週間売上は？
- 昨年の West 地区における全コーヒー製品の毎月の週間平均売上は？

中難易度

- Los Angeles および New York と比較して、San Jose における Lotta Latte の売上はどのように評価されるか？
- 過去 2 年間の全市場における Lotta Latte の月間マーケットシェアは、どのように推移しているか？
- 計り売り紅茶製品について、請求金額が最も高い仕入先は？
- California における昨年 12 月の最も効果のあった販売促進活動は？

OLAP 機能または RISQL 拡張機能を使わないと非常に難しい質問

- 昨年の各月における Lotta Latte の累積売上は？
- 同期間における Lotta Latte の総売上に対する月間売上の比率 (%) は？
- 1998 年の売上と販売数量の下位 10 都市をあげると？
- 1999 年の第 1 四半期における販売収入について、上位 25%、中位 50%、下位 25% に入る Aroma の店舗は？

データウェアハウスに対する一般的なクエリ

ビジネスに関する一般的な問合せの多くは SQL クエリとして簡単に表現できます。たとえば、ある年度におけるある製品の四半期毎販売を返すクエリは、SQL の知識があれば誰でも作成できます。

ところが、一般的な質問の中には簡単に表現できないものも多くあります。比較を要する質問は、クエリの作成者にとっても、SQL そのものにとっても難題です。たとえば、週間、月間、四半期毎、年間の比較を問い合わせる質問は、販売分析で扱う最も基本的な質問の 1 つですが、クエリとして表現することは、クエリ作成者、クエリ言語、データベース サーバにとって非常に厄介なのです。

順次処理を要するビジネス上の質問も、SQL クエリとして表すのは非常に困難です。たとえば、簡単な累積合計を算出する場合でも、データ アナリストは複数のクエリをクライアント ツールで実行し、別のツールで結果を集計しなければなりません。この場合ユーザに高度な技能が要求され、ネットワーク上のデータ量も増え、一般にデータベース サーバより低速なクライアントで処理されるという、効率の悪い方法になります。

標準 SQL OLAP 機能および SQL の RISQL 拡張機能を使用すれば、使いやすく、ネットワークの負荷が少なく、サーバ上でさまざまな演算を高速に実行することができます。

まとめ

この章では、**Aroma** データベースの販売スキーマを簡単に説明し、IBM Red Brick Warehouse データベースが答えることができるビジネスに関する一般的な質問がどのようなものかを紹介しました。

意思決定支援データベースは、クエリを処理することを目的として設計されています。わかりやすいいくつかのテーブルから構成され、無類のクエリ性能を発揮し、データの整合性を確保します。これには、IBM Red Brick Warehouse データベースのプライマリ テーブルが以下の条件を備えている必要があります。

- 数が少ないこと
- アナリストが使う用語で設計されていること
- ビジネスの様々な特性のとらえ方を反映していること

以降の章では、ビジネスに関する一般的な質問の詳しい例題を紹介します。例題のほとんどは、**Aroma** 販売スキーマを基本としています。高度な例題では、別のテーブルも随時追加して使用します。これらのテーブルについては、付録 A「Aroma データベース詳細」で説明します。

第 2 章

基本的なクエリ

この章について

この章では、簡単な例題をいくつか紹介し、標準 SQL SELECT 文を使って IBM Red Brick Warehouse データベースからデータを抽出する方法を説明します。

この章で説明する内容は、次のとおりです。

- リレーショナル データベースのテーブルから、特定の列や行を抽出する
- 抽出したデータについて、論理的な演算を実行する
- 検索条件の中で、ワイルドカード文字を使用する
- 複数のテーブルからデータを抽出する
- データを一定順序に並べ替え、数値列の小計を算出する
- 集合関数により、集約演算を実行する
- データをグループ分けする
- 抽出データについて、算術演算を実行する
- 指定した列に NULL、ゼロ、空白が含まれていた場合に、リザルト セットから行を削除する

SELECT 文を使用したデータ抽出

例題

Aroma データベースに定義されている地域、地区、市場は？

SQL 文例

```
select * from market;
```

実行結果

Mktkey	HQ_City	HQ_State	District	Region
1	Atlanta	GA	Atlanta	South
2	Miami	FL	Atlanta	South
3	New Orleans	LA	New Orleans	South
4	Houston	TX	New Orleans	South
5	New York	NY	New York	North
6	Philadelphia	PA	New York	North
7	Boston	MA	Boston	North
8	Hartford	CT	Boston	North
9	Chicago	IL	Chicago	Central
10	Detroit	MI	Chicago	Central
11	Minneapolis	MN	Minneapolis	Central
12	Milwaukee	WI	Minneapolis	Central
14	San Jose	CA	San Francisco	West
15	San Francisco	CA	San Francisco	West

(1 / 2)

Mktkey	HQ_City	HQ_State	District	Region
16	Oakland	CA	San Francisco	West
17	Los Angeles	CA	Los Angeles	West
19	Phoenix	AZ	Los Angeles	West

(2 / 2)

データの抽出 : SELECT 文

SELECT 文は、データベースのテーブルからデータ列やデータ行を抽出したり、データに対して算術演算を実行したり、データをグループ分けしたり、一定順序に並べ替えたり、グループ分けしながら並べ替えたりするのに使用します。通常、SELECT 文は、SELECT キーワードで始まる簡単なクエリ式、その後に続く任意数の句やサブ句で構成します。複雑なクエリ式は、『SQL Reference Guide』を参照してください。

最も基本的な SELECT 文は、SELECT と FROM という 2 つのキーワードで構成されます。

```
SELECT select_list
FROM table_list;
```

select_list　カンマで区切られた列名か SQL 式です。アスタリスク (*) も使用できます。

table_list　カンマで区切られたテーブル名です。参照先のテーブルは <select_list> に指定されている列名を含んでいる必要があります。

SELECT と FROM (ならびに、本書の構文で大文字で表記するほかの単語) は、予約された SQL キーワードです。SQL には大文字 / 小文字の区別がないため、キーワードは大文字で書いても小文字で書いてもかまいません。

例題について

このクエリは、**Market** テーブルの内容をすべて抽出します。アスタリスク記号 (*) は、<table_list> 内のすべての列名を表す SQL の省略記号です。アスタリスクを指定するかわりに、**Market** テーブル内のすべての列名を列挙することもできます。

IBM Red Brick Warehouse は、明示的テーブルもサポートします。これを使うと、上記のクエリは次のように簡単に表すことができます。

```
table market;
```

使用上の注意

検索項目リストに指定する名称は、FROM 句に指定したテーブル内に定義されていなければなりません (例外は、この章で後述) 。指定した順序で、データベースの列が返されます。アスタリスクを使用すると、データベース テーブルに格納された順序で各列が返されます。

各例題の最後に記載されているセミコロン (;) は、SQL 構文の要素ではありません。RISQL エントリ ツール や RISQL レポータに必要な、ステートメントの終りを示すマーカです。クエリの入力に使用する対話型 SQL ツールによって、マーカが不要な場合もあります。

SELECT リストを使用した特定列の抽出

例題

Aroma データベースに定義されている地区と地域は？

SQL 文例

```
select district, region
from market;
```

実行結果

District	Region
Atlanta	South
Atlanta	South
New Orleans	South
New Orleans	South
New York	North
New York	North
Boston	North
Boston	North
Chicago	Central
Chicago	Central
Minneapolis	Central
Minneapolis	Central
San Francisco	West
San Francisco	West
San Francisco	West
Los Angeles	West
Los Angeles	West

特定の列を抽出する

SELECT 文の検索項目リストに列名を指定することで、任意のテーブルから特定の列の組み合わせを抽出できます。列は、検索項目リストに指定した順序で返されます。

例題について

このクエリは、**Market** テーブルから地区および対応する地域を抽出します。

使用上の注意

検索項目リストに指定する列名は、FROM 句に指定する参照先テーブル内で定義されていなければなりません。検索項目リストには、ほかの式も指定することができます。式の例は、本書の後半で紹介します。

検索項目リストにテーブル内のすべての列が指定されていないと、クエリは、前の例題のように、行を重複して返すことがあります。重複を避けるには、DISTINCT キーワードを使用します。たとえば、以下のクエリは、地区名と地域名を重複なしに **Market** テーブルから抽出します。

```
select distinct district, region
from market;
```

District	Region
Atlanta	South
Boston	North
Chicago	Central
Los Angeles	West
Minneapolis	Central
New Orleans	South
New York	North
San Francisco	West

WHERE 句を使用した特定行の抽出

例題

パッケージに入れずに販売する製品は？

SQL 文例

```
select prod_name, pkg_type
from product
where pkg_type = 'No pkg';
```

実行結果

Prod_Name	Pkg_Type
Veracruzano	No pkg
Xalapa Lapa	No pkg
Colombiano	No pkg
Expresso XO	No pkg
La Antigua	No pkg
Lotta Latte	No pkg
Cafe Au Lait	No pkg
NA Lite	No pkg
Aroma Roma	No pkg
Demitasse Ms	No pkg
Darjeeling Number 1	No pkg
Darjeeling Special	No pkg
Assam Grade A	No pkg

(1 / 2)

Prod_Name	Pkg_Type
Assam Gold Blend	No pkg
Earl Grey	No pkg
English Breakfast	No pkg
Irish Breakfast	No pkg
Special Tips	No pkg
Gold Tips	No pkg
Breakfast Blend	No pkg
Ruby's Allspice	No pkg
Coffee Mug	No pkg
Travel Mug	No pkg
Aroma t-shirt	No pkg
Aroma baseball cap	No pkg

(2 / 2)

特定の行を抽出する : WHERE 句

論理条件をクエリ内に指定すると、特定の行の集合をテーブルから抽出することができます。論理条件は、WHERE 句で指定します。指定した条件を満たす行が返され、それ以外の行は破棄されます。論理条件は、検索条件、プレディケート、制約、修飾子とも呼ばれます。

WHERE 句

```
SELECT select_list
FROM table_list
[WHERE search_condition];
```

<search_condition>　　真か偽のどちらかを判定する条件です。

WHERE 句はオプションです。

例題について

このクエリは、パッケージづめされていない製品の名前を抽出し、表示します。IBM Red Brick Warehouse は、**Product** テーブルの各行に対して次の条件を判定し、条件を満たした行だけを返します。

```
pkg_type = 'No pkg'
```

使用上の注意

一重引用符で囲んだ文字列を、文字列定数と呼びます。文字列定数の中で一重引用符を使用する場合は、以下のように 2 つの一重引用符 ('') で表します。たとえば、以下のように指定します。

```
'Scarlet O''Hara'
```

文字列定数は、データベースに格納されているとおりに、大文字と小文字を区別して指定してください。たとえば、次の条件を考えてみます。

```
class_type = 'Bulk_beans'
```

この条件は、参照先の列に次の文字列が含まれていると偽になります。

```
'BULK_beans'
```

WHERE 句で集合関数は使用できません。集合関数の詳細は、2-31 ページを参照してください。

AND、NOT、および OR 接続詞を使用した複合条件の指定

例題

South 地区か West 地区に所在する都市と地区は？

SQL 文例

```
select hq_city, district, region
from market
where region = 'South' or region = 'West';
```

実行結果

HQ_City	District	Region
Atlanta	Atlanta	South
Miami	Atlanta	South
New Orleans	New Orleans	South
Houston	New Orleans	South
San Jose	San Francisco	West
San Francisco	San Francisco	West
Oakland	San Francisco	West
Los Angeles	Los Angeles	West
Phoenix	Los Angeles	West

複合条件の指定：AND、NOT、OR

意思決定支援分析に使用するクエリには、複合条件を指定するものが多くあります。複合条件とは、個々の条件を論理接続詞で結合したものです。SQL には、以下の論理接続詞があります。

接続詞	名前	優先順位
( )	かっこ (演算優先順位の明示)	1
AND	論理積	3
NOT	否定	2
OR	論理和	4

サーバは、複合条件を次の順に判定します。1. すべての NOT 演算子、2. すべての AND 接続詞、3. すべての OR 接続詞。この演算の評価される順番は、一般に優先順位と呼ばれます。

複合条件をかっこで囲んでグループ分けすると、演算優先順位を指定することができます。かっこがネストされている場合、IBM Red Brick Warehouse によって 1 番内側のかっこから順に評価されます。複合条件の論理が明確でなければ、かっこを使って明示してください。

例題について

このクエリは、South 地区か West 地区に所在するすべての都市と地区を抽出します。**Region** 列が South または West であるすべての行が、複合条件を満たしている行として結果テーブルに返されます。

使用上の注意

演算優先順位が複雑な場合は、かっこで条件をグループ分けして明示してください。

AND 接続詞を使用した複合条件の指定

例題

Los Angeles または San Jose に所在する小規模または大規模な Aroma 店舗は？

SQL 文例

```
select store_type, store_name, city
from store
where (store_type = 'Large' or store_type = 'Small')
    and (city = 'Los Angeles' or city = 'San Jose');
```

実行結果

Store_Type	Store_Name	City
Large	San Jose Roasting Company	San Jose
Large	Beaches Brew	Los Angeles
Small	Instant Coffee	San Jose

複合条件の指定

検索条件、特に意思決定支援分析のための検索条件は複雑なものになることがあります。論理接続詞 AND、OR、および NOT を使用して単純な条件を組み合わせても、条件が複雑化すると理解しにくくなることがあります。SQL は自由形式ですから、タブ、空白、改行を使って論理的関係を見やすく表記し、複合条件の論理的構造を明示できます。

例題について

このクエリは、Los Angeles または San Jose に所在するという条件と、小規模または大規模であるという条件の両方を満たす Aroma 店舗の名前を抽出し、表示します。

AND 接続詞は OR 接続詞より優先順位が高いため、かっこが必須です。かっこを使わないと、結果テーブルの内容が違ってしまいます。

Store_Type	Store_Name	City
Large	San Jose Roasting Company	San Jose
Large	Beaches Brew	Los Angeles
Small	Instant Coffee	San Jose
Large	Miami Espresso	Miami
Large	Olympic Coffee Company	Atlanta

使用上の注意

検索条件によって明示的にデータを限定せずに抽出するクエリや、選択性の低い条件だけを少数使って指定したクエリからは、膨大な数の行が返されることがあります。

複合条件をサーバがどのように判定するかが不明確な場合は、かっこを使って条件を明示的にグループ分けし、演算優先順位を明示してください。

> 演算子と <= 演算子の使用

例題

Mktkey の値が 4 より大きく、12 以下である都市と地区は？

SQL 文例

```
select mktkey, hq_city, hq_state, district
from market
where mktkey > 4
    and mktkey <= 12;
```

実行結果

Mktkey	HQ_City	HQ_State	District
5	New York	NY	New York
6	Philadelphia	PA	New York
7	Boston	MA	Boston
8	Hartford	CT	Boston
9	Chicago	IL	Chicago
10	Detroit	MI	Chicago
11	Minneapolis	MN	Minneapolis
12	Milwaukee	WI	Minneapolis

比較演算子の使用

真か偽のどちらかとして判定される条件は、比較演算子または比較プレディケートを使って表すことができます。比較プレディケートについて、次の 2 ページに渡って説明します。

SQL では、以下の比較演算子を使用します。

演算子	名称
=	等号 (等しい)
<	不等号 (より小さい)
>	不等号 (より大きい)
<>	不等号 (等しくない)
>=	不等号 (以上)
<=	不等号 (以下)

例題について

このクエリは、**Mktkey** の値が 4 より大きく、12 以下である都市と地区を抽出し、表示します。

Mktkey 列には整数値が格納されているため、数値と比較することができます。整数と文字を比較するとエラー メッセージが返されます。

```
select mktkey, hq_city, hq_state, district
from market
where mktkey > '4';

** ERROR ** (19) Operands of comparison must have comparable
datatypes.
```

使用上の注意

条件で比較される値は、互いに比較可能なデータ型である必要があります。異なるデータ型を比較すると、エラー メッセージまたは誤った結果が返されます。次の例が示すように、比較演算子で文字列どうしを比較することもできます。

```
(city > 'L')
```

比較可能なデータ型の詳細については、『SQL Reference Guide』を参照してください。

IN 比較プレディケートの使用

例題

Chicago、New York、New Orleans の各地区にある都市は？

SQL 文例

```
select hq_city, hq_state, district
from market
where district in
    ('Chicago', 'New York', 'New Orleans');
```

実行結果

HQ_City	HQ_State	District
New Orleans	LA	New Orleans
Houston	TX	New Orleans
New York	NY	New York
Philadelphia	PA	New York
Chicago	IL	Chicago
Detroit	MI	Chicago

比較プレディケートの使用

以下の SQL 比較プレディケートを使い、単純な条件を表すことができます。

プレディケート
BETWEEN <expression1> AND <expression2>
LIKE パターン
IN (< リスト >)
IS NULL
IS NOT NULL
ALL
SOME または ANY
EXISTS

ALL、SOME または ANY、および EXISTS の例は、第 4 章「比較クエリ」で紹介します。

全プレディケートの構文、使用例、ならびに単純な式と複雑な式の詳しい定義方法は、『SQL Reference Guide』を参照してください。

例題について

このクエリは、Chicago、New York、New Orleans の各地区にあるすべての都市を表示します。以下のように、等号比較演算子 (=) と複数の OR 条件を使って表すこともできます。

```
where district = 'Chicago'
    or district = 'New York'
    or district = 'New Orleans'
```

使用上の注意

なるべく簡単で理解しやすく、管理しやすい論理的条件を使用してください。空白を使って複合条件の論理的構造を明確にし、インデントを使って論理ブロックを明確にし、かっこで優先順位を明示してください。

% ワイルドカードの使用

例題

Min という文字で始まる地区にある都市は？

SQL 文例

```
select district, hq_city
from market
where district like 'Min%';
```

実行結果

District	HQ_City
Minneapolis	Minneapolis
Minneapolis	Milwaukee

ワイルドカード文字を使う

これまで紹介したクエリでは、文字列全体が一致している条件を扱ってきました。LIKE プレディケートと 2 種類のワイルドカード (%、_) を使うと、文字列の一部 (部分文字列) だけを照合させることができます。

パーセント記号 (%) のワイルドカードは、任意の文字列に置き換えることができます。たとえば、以下のように指定します。

- `like 'TOT%'` と指定すると、`'TOT'` で始まる文字列はすべて真になります。
- `like '%ZERO%'` と指定すると、`'ZERO'` を含む文字列はすべて真になります。
- `like '%FRESH'` と指定すると、`'FRESH'` で終わり後続空白文字のない文字列はすべて真になります。文字データ中にある後続空白文字は LIKE 制約の適用時に有効と見なされます。

パーセント記号は、Null (文字数が 0) の検索にも使用できます。

下線記号のワイルドカード (_) は、その位置の 1 字に置き換えることができます。たとえば、以下のように指定します。

- `like '_EE_'` と指定した場合、中間の 2 文字が `'EE'` である 4 字の文字列はすべて真になります。
- `like '%LE_N%'` と指定すると、`'LE_N'` というパターンを含む文字列はすべて真になります。`'CLEAN'`、`'KLEEN'`、`'VERY KLEEN'` は、どれもこのパターンに一致します。

例題について

このクエリは、Min という文字で始まる地区名をすべて抽出し、各地区にある都市を表示します。パーセント記号のワイルドカード (%) は、Min の n より後は空白も含めてどの文字でもよいのですが、n より前は指定された文字パターンでなければならないということを意味します。

使用上の注意

LIKE を使った条件は、列内の部分文字列と指定パターンが一致すれば TRUE (真) になります。パターンにワイルドカードが含まれていなければ、パターン全体と列の内容全体が一致しなければなりません。たとえば、次の条件は、列の内容が文字列 APRIL だけを含み、ほかに何も含まない場合に限って真になります。

```
month like 'APRIL'
```

つまり、以下の条件と同等になります。

```
month = 'APRIL'
```

LIKE プレディケートは、文字列を含む列だけに使用できます。

シンプル ジョインの使用

例題

1999 年のタイプが 900 の販売促進活動で、週末に売れた Easter 製品の日別売上の合計を、店舗ごとに表示すると？

SQL 文例 1

```
select prod_name, store_name, day, dollars
from promotion, product, period, store, sales
where promotion.promokey = sales.promokey
    and product.prodkey = sales.prodkey
    and product.classkey = sales.classkey
    and period.perkey = sales.perkey
    and store.storekey = sales.storekey
    and prod_name like 'Easter%'
    and day in ('SA', 'SU')
    and promo_type = 900
    and year = 1999;
```

SQL 文例 2

```
select prod_name, store_name, day, dollars
from promotion natural join sales
    natural join product
    natural join period
    natural join store
where prod_name like 'Easter%'
    and day in ('SA', 'SU')
    and promo_type = 900
    and year = 1999;
```

実行結果 (2 つのクエリは同一の結果となる)

Prod_Name	Store_Name	Day	Dollars
Easter Sampler Basket	Olympic Coffee Company	SA	150.00

ディメンジョンとファクトのジョイン

この章で紹介した前述の SQL 文では、1 つのテーブルからデータを抽出しましたが、複数テーブルの情報をジョインするクエリの方が一般的です。通常、ディメンジョン テーブルをファクト テーブルとジョインし、目的のファクトを抽出します。たとえば、**Sales** ファクト テーブルを **Store** ディメンジョンと **Product** ディメンジョンにジョインし、製品別、店舗別の売上を抽出したり、**Period** ディメンジョンと **Product** ディメンジョンにジョインし、製品別の週間売上を抽出します。

例題について

このクエリは、**Aroma** 販売スキーマの 5 つのテーブルをジョインする必要があります。つまり、**Sales** ファクト テーブルと、**Product**、**Period**、**Store**、**Promotion** の各ディメンジョンのジョインです。

クエリ内でテーブルをジョインするには、ジョインの実行方法をデータベース サーバに明示しなければなりません。SQL 文 1 では、WHERE 句で 5 つの単純な条件によるジョインを指定し、5 つのプライマリ キー列に基づいて **Sales** テーブルと各ディメンジョンをジョインします。**Product** テーブルには 2 つの部分からなるプライマリ キーがあるので、Product テーブルは **Prodkey** と **Classkey** の 2 つの列を基準にして **Sales** テーブルにジョインされます。

どの条件にも明示的なジョイン列が含まれるため、FROM 句にナチュラル ジョインを指定した SQL 文 2 の形で表すこともできます。このようなテーブル ジョインは、**Aroma** データベースでは正常に機能します。主要テーブルが単純なスター スキーマを形成し、すべての外部キー列が参照先のプライマリ キー列と同じ名前を使用しているからです。

ナチュラル ジョインは、各テーブルが共有する同一名の列の組み合わせに対して動作します。SQL 文 2 では、**Sales** テーブルと **Product** テーブルが **Classkey** 列と **Prodkey** 列に基づいてジョインされます。

FROM 句で上記のテーブルをジョインする方法は、ほかに 2 種類あります。詳細は第 5 章「ジョインとユニオン」を参照してください。各種のジョイン クエリと例題が示されています。

使用上の注意

2 つのテーブルは、比較可能なデータ型の列を基準としてジョインすることができます。ジョインは、この例のプライマリ キーや外部キーの関係に依存しません。

ORDER BY 句の使用

例題

1999 年 11 月の Instant Coffee 店における Assam Gold Blend と Earl Grey の売上は？各製品について、売上の多い順に並べ替えてください。

SQL 文例

```
select prod_name, store_name, dollars
from store natural join sales
     natural join product
     natural join period
where (prod_name like 'Assam Gold%'
    or prod_name like 'Earl%')
    and store_name like 'Instant%'
    and month = 'NOV'
    and year = 1999
order by prod_name, dollars desc;
```

実行結果

Prod_Name	Store_Name	Dollars
Assam Gold Blend	Instant Coffee	96.00
Assam Gold Blend	Instant Coffee	78.00
Assam Gold Blend	Instant Coffee	66.00
Assam Gold Blend	Instant Coffee	58.50
Assam Gold Blend	Instant Coffee	58.50
Assam Gold Blend	Instant Coffee	39.00
Assam Gold Blend	Instant Coffee	39.00
Assam Gold Blend	Instant Coffee	32.50
Earl Grey	Instant Coffee	48.00
Earl Grey	Instant Coffee	45.50
Earl Grey	Instant Coffee	42.00
Earl Grey	Instant Coffee	32.00
Earl Grey	Instant Coffee	24.00
Earl Grey	Instant Coffee	20.00

リザルト テーブルの順序指定 : ORDER BY 句

ORDER BY 句は、指定した任意数の列の値に基づいてクエリのリザルト テーブルをソートするのに使用します。デフォルトのソート順は、昇順 (ASC) です。指定列の値を降順にソートするには、次に示すように DESC キーワードを使用します。

```
order by prod_name, 3 desc
```

検索項目リストに式 (集合関数など) を指定して結果をソートするには、式に使用する列のエリアスを指定し、そのエリアスを ORDER BY で指定します。列エリアスの詳細は、2-33 ページを参照してください。

GROUP BY 句の構文

```
SELECT select_list
FROM table_list
[WHERE <search_condition>]
[ORDER BY <order_list>] ;
```

<order_list>　データの並べ替えの順序を指定する列のリスト。<order_list> に指定する列は、<select_list> になくてもかまいませんが、FROM 句の参照先テーブルに存在するものでなければなりません。

例題について

このクエリは、1999 年 11 月の Instant Coffee 店における Assam Gold Blend と Earl Grey の売上を抽出し、製品名と日別総売上に基づいて結果をソートします。

使用上の注意

ORDER BY 句は、SELECT 文のほかの句 (SUPPRESS BY 句を除く) の後に指定し、ソートする列のリストを指定しなければなりません。列は、検索項目リストに指定した列名、列エリアス、位置 (序数) のいずれかに基づいて参照できます。たとえば、前出 SQL 例題の ORDER BY 句は、次のように記述できます。

```
order by prod_name, 3 desc
```

<order_list> にない列を <select_list> に指定すると、リザルト テーブルに表示されない列に基づいてデータをソートできます。

小計の算出

例題

1999 年 11 月の Instant Coffee 店における Assam Gold Blend、Darjeeling Special、Earl Grey の日別売上と月間売上の小計は？ 3 製品合わせた月間小計は？

SQL 文例

```
select prod_name, store_name, dollars
from store natural join sales
    natural join product
    natural join period
where prod_name in ('Assam Gold Blend', 'Earl Grey',
    'Darjeeling Special')
    and store_name like 'Instant%'
    and month = 'NOV'
    and year = 1999
order by prod_name, dollars desc
    break by prod_name summing 3;
```

実行結果

Prod_Name	Store_Name	Dollars
Assam Gold Blend	Instant Coffee	96.00
Assam Gold Blend	Instant Coffee	78.00
Assam Gold Blend	Instant Coffee	66.00
Assam Gold Blend	Instant Coffee	58.50
Assam Gold Blend	Instant Coffee	58.50
Assam Gold Blend	Instant Coffee	39.00

(1 / 2)

実行結果

Prod_Name	Store_Name	Dollars
Assam Gold Blend	Instant Coffee	39.00
Assam Gold Blend	Instant Coffee	32.50
Assam Gold Blend	NULL	467.50
Darjeeling Special	Instant Coffee	207.00
Darjeeling Special	Instant Coffee	168.00
Darjeeling Special	Instant Coffee	149.50
Darjeeling Special	Instant Coffee	144.00
Darjeeling Special	Instant Coffee	138.00
Darjeeling Special	Instant Coffee	132.00
Darjeeling Special	Instant Coffee	96.00
Darjeeling Special	Instant Coffee	69.00
Darjeeling Special	Instant Coffee	60.00
Darjeeling Special	Instant Coffee	60.00
Darjeeling Special	Instant Coffee	48.00
Darjeeling Special	NULL	1271.50
Earl Grey	Instant Coffee	48.00
Earl Grey	Instant Coffee	45.50
Earl Grey	Instant Coffee	42.00
Earl Grey	Instant Coffee	32.00
Earl Grey	Instant Coffee	24.00
Earl Grey	Instant Coffee	20.00
Earl Grey	NULL	211.50
NULL	NULL	1950.50

(2 / 2)

小計の算出 : BREAK BY 句

ORDER BY 句を使うクエリでは、リザルト セットの集計区分を BREAK BY 句で指定して、数値列の小計を算出することができます。BREAK BY 句は、小計を合計し、レポートの最終行に総計を表示する機能も持っています。この句は、ANSI SQL-92 標準に準拠した RISQL 拡張機能です。

BREAK BY 句の構文

```
SELECT select_list
FROM table_list
[WHERE <search_condition>]
[ORDER BY <order_list>]
    [BREAK BY <order_reference> SUMMING
<select_reference_list>];
```

<order_reference>	<order_list> で使用される列
<select_reference_list>	<select_list> で使用される数式

例題について

このクエリは、1999 年 11 月の Instant Coffee 店における 3 つの紅茶製品の日別売上を表示します。製品ごとの売上を小計し、3 製品すべての総計をレポートの最後に表示します。order_reference は **Prod_Name** で、<select_reference_list> には 1 つの列 (**Dollars** 列に対応する 3) を指定しています。

使用上の注意

BREAK BY 句は、一定順序の行集合について値を集計するだけでなく、大量のリザルト セットをわかりやすくします。

クエリで RISQL 表示関数が使用されている場合は、RESET BY サブ句というもう 1 つの RISQL 拡張機能を ORDER BY 句で指定することができます。詳細は、3-10 ページを参照してください。

BREAK BY 句は、INSERT INTO...SELECT 形式の文のクエリ式には使用できません。

SUM、AVG、MAX、MIN、COUNT 集合関数の使用

例題

Los Angeles における Lotta Latte の 1999 年の総売上は？同年度の日別売上の平均値、最大値、最小値、ならびに各数値の集計に使用された日数は？

SQL 文例

```
select sum(dollars), avg(dollars), max(dollars),
min(dollars),
    count(*)
from store natural join sales
    natural join period
    natural join product
where prod_name like 'Lotta Latte%'
    and year = 1999
    and city like 'Los Ang%';
```

実行結果

Sum	Avg	Max	Min	Count
13706.50	171.33125000	376.00	39.00	80

集合関数の使用

集合関数は、値の集合に対する操作です。たとえば、SUM(dollars) はリザルト テーブルの金額を合計し、AVG(dollars) は平均値を算出します。以下に示す SQL 集合関数は、検索項目リスト内で何度でも使用できます。

機能	説明
SUM(<expression>)	<expression> 内のすべての値を合計する
SUM(DISTINCT <expression>)	<expression> 内の一意値を合計する
AVG(<expression>)	<expression> 内のすべての値を平均する
AVG(DISTINCT <expression>)	<expression> の一意値を平均する
MAX(<expression>)	<expression> の最大値を判定する
MIN(<expression>)	<expression> 内の最小値を判定する
COUNT(*)	返された行数をカウントする
COUNT(<expression>)	expression 内の非 Null 値の数をカウントする
COUNT(DISTINCT<expression>)	expression 内の特定の非 Null 値の数をカウントする

<expression> は、列名や数値表現式に置き換えます。COUNT(*) を除き、どの関数も NULL 値を無視して集計値を算出します。

例題について

このクエリは、Los Angeles における Lotta Latte の 1999 年の総売上を抽出し、リザルト セットは同年度の売上について、平均値、最大値、最小値、ならびに各数値の集計に使用された日数を返します。

使用上の注意

個別の値と集計値の両方を含むリザルト セットを得る場合は、クエリに GROUP BY 句を指定しなければなりません。ORDER BY 句の詳細は、2-35 ページを参照してください。

集合関数は RISQL 表示関数の引数として使用できますが、表示関数は集合関数の引数として使用できません。表示関数については、第 3 章「データの解析」で説明します。

列エリアスの使用

例題

Los Angeles における Lotta Latte の 1999 年の年間売上は？同期間の売上の平均値、最大値、最小値、ならびに各数値の集約に使用された日数は？集約結果の各列に表記名を付けてください。

SQL 文例

```
select sum(dollars) as dol_sales, avg(dollars) as avg_sales,
    max(dollars) as max_dol, min(dollars) as min_dol,
    count(*) as num_items
from store natural join sales
    natural join period
    natural join product
where prod_name like 'Lotta Latte%'
    and year = 1999
    and city like 'Los Ang%';
```

実行結果

Dol_Sales	Avg_Sales	Max_Dol	Min_Dol	Num_Items
13706.50	171.33125000	376.00	39.00	80

列エリアスの使用：AS

SELECT コマンドは、デフォルトでは集合関数で算出した値を返すだけで、戻り値に表記名を付けません。AS キーワードの後に文字列が続く列には、列のラベル、つまり列エリアスを指定できます。このエリアスは、それ以降のクエリの句の中で参照することができます。

たとえば、次の AS 句は、**Dollars** 列に **Dol_Sales** というエリアスを割り付けます。

```
dollars as dol_sales
```

2-35 ページに示すように、列エリアスはそれ以降に作成した句の項目リストで列を参照する場合に非常に便利です。

例題について

このクエリは、前のクエリと同じリザルト セットを返しますが、列エリアスの割り付けによって集約結果に表記名が付けられます。

使用上の注意

リザルト テーブルを読みやすくするには、クエリの検索項目リストに指定する集合関数に列エリアスを割り当てます。

エリアスは、データベースの識別子です。文字で始まり、128 文字以内でなければなりません。先頭文字の後は、ゼロ、文字、数値、下線も使用できます。キーワードは、データベース識別子として使用することはできません。詳細は、『SQL Reference Guide』を参照してください。

列エリアスは、参照先の列を指定する SELECT 文の任意の箇所 (WHERE、ORDER BY、GROUP BY、HAVING などの句) で使用できます。

重要： 列エリアスは、参照する列の値が集合関数の結果である場合は WHERE 句に指定できません。HAVING 句では使用できます。

GROUP BY 句を使用した列のグループ分け

例題

各地区における 1998 年のコーヒー マグの年間総売上は？同期間における売上の平均値、最大値、最小値は？地区ごとに、結果を表示してください。

SQL 文例

```
select district as district_city, sum(dollars) as dol_sales,
    avg(dollars) as avg_sales, max(dollars) as max_sales,
    min(dollars) as min_sales
from market natural join store
     natural join sales
     natural join period
     natural join product
where prod_name like '%Mug%'
and year = 1998
group by district_city
order by dol_sales desc;
```

実行結果

District_City	Dol_Sales	Avg_Sales	Max_Sales	Min_Sales
Atlanta	1378.30	35.34102564	98.55	4.00
Los Angeles	711.60	30.93913043	98.55	9.95
San Francisco	410.45	25.65312500	54.75	5.00

行のグループ分け：GROUP BY **句**

集合関数は、リザルト テーブルの全行か、GROUP BY 句で指定した行のグループに対する操作です。たとえば、市場単位で売上をグループ分けし、各市場の総売上、最大値、最小値を算出することができます。

GROUP BY **句の構文**

```
SELECT select_list
FROM table_list
[WHERE <search_condition>]
[GROUP BY <group_list>]
[ORDER BY <order_list>]
    [BREAK BY <order_reference>SUMMING
<select_reference_list>];
```

<group_list>　列名 (<select_list> の列名または FROM 句に指定したテーブルの列名) または <select_list> に指定した列エリアスのリストです。<select_list> に指定するすべての非集約列は <group_list> で指定されている必要があります。

例題について

このクエリは、3 地区で販売されているコーヒー マグの 1998 年の年間総売上を抽出し、売上の多い順に並べ替えます。サーバがこのクエリを処理する手順は、以下のようになります。

1. FROM 句に指定されたテーブルからすべてのデータ行を抽出し、各テーブルの行をジョインし、中間リザルト テーブルを出力する。
2. WHERE 句に指定された検索条件を満たす行を、中間リザルト テーブルから抽出する。
3. リザルト テーブルを、GROUP BY 句に指定されたグループに分割する。
4. リザルト テーブル全体の指定グループに対し、集合関数を処理する。
5. ORDER BY 句に従い、結果を一定順序に並べ替える。
6. 検索項目リストに指定された列だけを返す。

使用上の注意

Vista のクエリ リライト システムを使用して、集合関数または GROUP BY 句を使用する集約クエリの性能を向上させることができます。詳細は、『IBM Red Brick Vista User's Guide』を参照してください。

ORDER BY 句は、検索項目リストの項目を列名、列エリアス、位置のいずれかに基づいて参照します。order_list の項目が集合関数の場合は、列名がないため、エリアス (**Dol_Sales**) または位置番号に基づいて参照します。列エリアスの詳細は、2-33 ページを参照してください。

GROUP BY 句を使用した複数グループの生成

例題

各都市における 1998 年と 1999 年の総売上は？各年度について、各地域と各地区の都市毎に表示してください。

SQL 文例

```
select year, region, district, city, sum(dollars) as sales
from market natural join store
    natural join sales
    natural join product
    natural join period
where year in (1998, 1999)
group by year, region, district, city
order by year, region, district, city;
```

実行結果

Year	Region	District	City	Sales
1998	Central	Chicago	Chicago	133462.75
1998	Central	Chicago	Detroit	135023.50
1998	Central	Minneapolis	Milwaukee	172321.50
1998	North	Boston	Boston	184647.50
1998	North	Boston	Hartford	69196.25
1998	North	New York	New York	181735.00
1998	North	New York	Philadelphia	172395.75
1998	South	Atlanta	Atlanta	230346.45
1998	South	Atlanta	Miami	220519.75
1998	South	New Orleans	Houston	183853.75
1998	South	New Orleans	New Orleans	193052.25
1998	West	Los Angeles	Los Angeles	219397.20
1998	West	Los Angeles	Phoenix	192605.25
1998	West	San Francisco	Cupertino	180088.75
1998	West	San Francisco	Los Gatos	176992.75
1998	West	San Francisco	San Jose	395330.25
1999	Central	Chicago	Chicago	131263.00
1999	Central	Chicago	Detroit	136903.25
1999	Central	Minneapolis	Milwaukee	173844.25
1999	Central	Minneapolis	Minneapolis	132125.75
1999	North	Boston	Boston	189761.00
1999	North	Boston	Hartford	135879.50

(1 / 2)

Year	Region	District	City	Sales
1999	North	New York	New York	171749.75
1999	North	New York	Philadelphia	171759.50
1999	South	Atlanta	Atlanta	229615.05
1999	South	Atlanta	Miami	234458.90
1999	South	New Orleans	Houston	186394.25
1999	South	New Orleans	New Orleans	190441.75
1999	West	Los Angeles	Los Angeles	228433.00
1999	West	Los Angeles	Phoenix	197044.50
1999	West	San Francisco	Cupertino	196439.75
1999	West	San Francisco	Los Gatos	175048.75
1999	West	San Francisco	San Jose	398829.10

(2 / 2)

グループ分けされた結果のネスト：GROUP BY 句

GROUP BY 句に複数の列名を指定した場合は、リザルト テーブルのグループが小グループに細分されます。たとえば、年度、地域、地区の列名を GROUP BY 句に指定すると、結果が年度別に分類され、各年度の結果が地域別に分類され、各地域の結果が地区別に分類されます。

GROUP BY **句の構文**

```
SELECT select_list
FROM table_list
[WHERE <search_condition>]
[GROUP BY <group_list>]
[ORDER BY <order_list>]
    [BREAK BY <order_reference> SUMMING
select_reference_list];
```

<group_list>　<select_list> の列名または <table_list> 内のテーブルに指定されている列名か、<select_list> 内に指定されている列エリアスのリストです。<group_list> には、<select_list> に指定した集合関数の対象にならない列 (非集約列) を指定します。

例題について

このクエリは、各都市における 1998 年と 1999 年の全製品の年間売上を抽出します。売上は、年度別、地域別、地区別、都市別にグループ分けされ、ソートされます。

重要：このクエリで参照される都市は、**Store** テーブルで定義された各店舗が所在する都市であり、**Market** テーブルで **hq_cities** として定義された都市ではありません。

使用上の注意

検索項目リストに集約関数を指定し、GROUP BY 句をクエリに指定しない場合は、すべての参照先列が集約関数でなければなりません。

除算演算子 (/) の使用

例題

1998 年の製品別平均単価は？総売上を総販売数量で除算し、平均値を算出してください。

SQL 文例

```
select prod_name, sum(dollars) as total_sales,
    sum(quantity) as total_qty,
    string(sum(dollars)/sum(quantity), 7, 2) as price
from product natural join sales
    natural join period
where year = 1998
group by prod_name
order by price;
```

実行結果

Prod_Name	Total_Sales	Total_Qty	Price
Gold Tips	38913.75	11563	3.36
Special Tips	38596.00	11390	3.38
Earl Grey	41137.00	11364	3.61
Assam Grade A	39205.00	10767	3.64
Breakfast Blend	42295.50	10880	3.88
English Breakfast	44381.00	10737	4.13
Irish Breakfast	48759.00	11094	4.39
Coffee Mug	1054.00	213	4.94
Darjeeling Number 1	62283.25	11539	5.39
Ruby's Allspice	133188.50	23444	5.68

(1 / 2)

Prod_Name	Total_Sales	Total_Qty	Price
Assam Gold Blend	71419.00	11636	6.13
Colombiano	188474.50	27548	6.84
Aroma Roma	203544.00	28344	7.18
La Antigua	197069.50	26826	7.34
Veracruzano	201230.00	26469	7.60
Expresso XO	224020.00	28558	7.84
Aroma baseball cap	15395.35	1953	7.88
Lotta Latte	217994.50	26994	8.07
Cafe Au Lait	213510.00	26340	8.10
Aroma Sounds Cassette	5206.00	620	8.39
Xalapa Lapa	251590.00	29293	8.58
NA Lite	231845.00	25884	8.95
Demitasse Ms	282385.25	28743	9.82
Aroma t-shirt	20278.50	1870	10.84
Travel Mug	1446.35	133	10.87
Darjeeling Special	127207.00	10931	11.63
Spice Sampler	6060.00	505	12.00
Aroma Sounds CD	7125.00	550	12.95
French Press, 2-Cup	3329.80	224	14.86
Spice Jar	4229.00	235	17.99
French Press, 4-Cup	3323.65	167	19.90
Tea Sampler	13695.00	550	24.90
...			

(2 / 2)

算術演算子の使用：( )、+、-、*、/

算術演算子は、検索条件リストか検索条件の中で使用することができます。次の表は、すべての算術演算子を表示したものです。演算優先順位は、表の上から下、同一行内では左から右です。

演算子	Name
( )	演算優先順の明示
+、-	正と負
*、/	乗算と除算
+、-	加算と減算

式の演算優先順位を指定する場合は、かっこを使って式をグループ分けしてください。たとえば、(4 + 3 * 2) の評価結果は 10 になりますが、式をグループ分けして ((4 + 3) * 2) にすると 14 になります。

使用上の注意

このクエリは、long 型の数値を **Price** 列に返します。STRING スカラ関数を使うと、**Price** の値が小数第 2 位まで算出されます。

```
string(sum(dollars)/sum(quantity), 7, 2) as price
```

STRING などのスカラ関数は、『SQL Reference Guide』を参照してください。

HAVING 句を使用したグループの除外

例題

1999 年の総売上が 25,000 ドル未満の製品は？製品別の販売総数は？

SQL 文例

```
select prod_name, sum(dollars) as total_sales,
    sum(quantity) as total_units
from product natural join sales
    natural join period
where year = 1999
group by prod_name
having total_sales < 25000
order by total_sales desc;
```

実行結果

Prod_Name	Total_Sales	Total_Units
Aroma t-shirt	21397.65	1967
Espresso Machine Royale	18119.80	324
Espresso Machine Italiano	17679.15	177
Coffee Sampler	16634.00	557
Tea Sampler	14907.00	597
Aroma baseball cap	13437.20	1696
Aroma Sheffield Steel Teapot	8082.00	270
Spice Sampler	7788.00	649
Aroma Sounds CD	5937.00	459
Aroma Sounds Cassette	5323.00	630
French Press, 4-Cup	4570.50	230

(1 / 2)

Prod_Name	Total_Sales	Total_Units
Spice Jar	4073.00	227
French Press, 2-Cup	3042.75	205
Travel Mug	1581.75	145
Easter Sampler Basket	1500.00	50
Coffee Mug	1258.00	256
Christmas Sampler	1230.00	41

(2 / 2)

グループに対する条件 : HAVING 句

データをグループ分けすると、返される情報量が少なくなりますが、それでもクエリが必要以上の情報を返すことがあります。HAVING 句を使うと、指定した条件を満たさないグループを除外できます。たとえば、金額が所定の値より少ないもの、あるいは多いものという条件を指定できます。

このクエリは、1999 年の製品別総売上を算出し、総売上が 25,000 ドル未満の製品だけを抽出します。

HAVING 句の構文

```
SELECT select_list
FROM table_list
[WHERE search_condition]
[GROUP BY <group_list>]
[HAVING <condition>]
[ORDER BY <order_list>]
    [BREAK BY <order_reference>SUMMING
<select_reference_list>];
```

<condition>　集合関数を使用できる SQL 条件

HAVING 句と WHERE 句では、以下の相違点があります。

WHERE 句	HAVING 句
グループ分けする前のデータ行を対象とする。	グループ分けした後のリザルト セットを対象とする。
SUM や AVG などの集合関数を使って条件を表現することはできないが、集約をしない場合には列エリアスを使用できる。	条件の表現には、集合関数や列エリアスも使用できる。

使用上の注意

HAVING 句の条件には、任意の集合関数を使用できます。検索項目リストに集合関数だけを指定した場合を除き、HAVING 句を使ったクエリには GROUP BY 句も指定しなければなりません。たとえば、以下のように指定します。

```
select min(prodkey), max(classkey)
from product
having min(prodkey) = 0;
```

NULL、ゼロ、空白を含む行の除外

例題

Aroma の仕入先から受けた注文の平均割引額は？

SQL 文例 1

```
select name as supplier,
    dec(sum(discount)/count(order_no),7,2) as avg_deal
from supplier natural join orders
    natural join deal
group by supplier
order by avg_deal desc;
```

実行結果

Supplier	Avg_Deal
Espresso Express	500.00
Western Emporium	66.66
Aroma West Mfg.	50.00
CB Imports	47.50
Leaves of London	40.00
Tea Makers, Inc.	20.00
Colo Coffee	0.00
Aroma East Mfg.	0.00
Crashing By Design	0.00

SQL 文例 2

```
select name as supplier,
    dec(sum(discount)/count(order_no),7,2) as avg_deal
from supplier natural join orders
    natural join deal
group by supplier
order by avg_deal desc
suppress by 2;
```

実行結果

Supplier	Avg_Deal
Espresso Express	500.00
Western Emporium	66.66
Aroma West Mfg.	50.00

(1 / 2)

Supplier	Avg_Deal
CB Imports	47.50
Leaves of London	40.00
Tea Makers, Inc.	20.00

(2 / 2)

空白行を削除する：SUPPRESS BY 句

クエリで抽出したデータに NULL、空白、ゼロを含む列がある場合は、SUPPRESS BY 句を使って最終的なリザルト セットから除外できます。この句は、ANSI SQL-92 標準に準拠した RISQL 拡張機能です。

SUPPRESS BY 句の構文

```
SELECT select_list
FROM table_list
[WHERE search_condition]
[GROUP BY <group_list>]
[HAVING condition]
[ORDER BY <order_list>]
    [BREAK BY <order_reference>SUMMING
<select_reference_list>]
[SUPPRESS BY <column_list>];
```

<column_list> <select_list> 内の列名またはエリアスのリスト、またはそれらの列を指定する位置番号のリスト

例題について

最初のクエリは、Aroma の仕入先が注文の割引を行ったかどうかにかかわらず、全仕入先のリストを抽出します。そのため、リザルト セット ファイルには平均取引額が 0 ドルの仕入先が 3 社含まれています。

2 番目のクエリは、列 2 **(Avg_Deal)** の値が 0.00 である行を除外し、上記の 3 社をリザルト セットから削除します。

使用上の注意

DEC スカラ関数は、**Avg_Deal** 列で桁数の多い数字を切り捨てます。2-42 ページで説明した STRING 関数と異なり、DEC 関数は平均値を小数点以下までの正確な数値 (文字列ではない) に変換します。

SUPPRESS BY 句は、RISQL 表示関数、または SQL OLAP 関数が計算される前に適用されます。したがって、これらの関数のいずれかを含む列を参照することによって行を除外することはできません。OLAP 関数および表示関数を含むクエリ例は、第 3 章「データの解析」を参照してください。

まとめ

SELECT 文

```
SELECT select_list
FROM table_list
[WHERE search_condition]
[GROUP BY group_list]
[HAVING search_condition]
[ORDER BY order_list]
    [BREAK BY order_reference SUMMING select_reference_list]
[SUPPRESS BY column_list];
```

論理接続詞

( )　かっこ (演算優先順位の明示)

NOT　否定

AND　論理積

OR　論理和

比較演算子

= 等号 (等しい)

< 不等号 (より小さい)

> 不等号 (より大きい)

<> 不等号 (等しくない)

>= 不等号 (以上)

<= 不等号 (以下)

比較プレディケート

BETWEEN	<expression1> AND <expression2>
LIKE	パターン
IN	(<list>)
IS	NULL
IS	NOT NULL

この章では、ビジネスに関する一般的な例題を SELECT 文として表現し、任意数のテーブルからデータを抽出し、データのグループ分けとソートをする方法を説明しました。合計値、平均値、最小値、最大値などの集約値を算出する方法、BREAK BY 句による小計の算出方法、ゼロ、NULL、空白を含む行を削除する SUPPRESS BY 句の使用方法も説明しました。

この章で扱った例題のほとんどは、標準の SELECT 文として簡単に表現でき、ユーザや SQL にとっても難題ではありません。この後の章では、さらに難しい例題を説明します。順次処理、集約値の比較、複雑なジョイン指定などを含んだ長い SELECT 文の説明です。

次章では、順次処理を必要とするような、ビジネスに関する問い合わせに答える OLAP 関数および RISQL 拡張機能の使い方を説明します。

第 3 章

データの解析

この章について

この章では、何らかのデータ解析を必要とするクエリについて説明します。多くのクエリには順次演算、つまり一定順序にソートされた行を端から処理していくような演算が含まれており、ビジネス分析によく使用されます。たとえば、以下のように指定します。

- 月間の累積合計 (総額) は？
- 週間の移動平均は？
- 月間売上の順位付けは？
- 年間売上に対する今月の売上の比率は？

IBM Red Brick Warehouse では、この種の質問に対応するため 2 種類の有効な方法を使用できます。

- 標準 SQL OLAP 関数
- RISQL 表示関数

この章では、OLAP および RISQL 構文について、それぞれ一連の例を挙げて説明します。これらのクエリを実行する前に、『SQL Reference Guide』の OLAP および RISQL 関数の説明をお読みください。

この章では、DATETIME 列から日付時間に関する情報を抽出し、演算を行うスカラ関数の使い方も説明します。

この章で紹介する例の多くは、集約された売上合計を使用します。**Sales** テーブルには日別合計しか格納されないため、クエリへの応答のために集約テーブルを作成すると便利です。集約テーブルの作成と使用については、『IBM Red Brick Vista User's Guide』を参照してください。

分析関数

以下に、SQL OLAP 関数および対応する RISQL 関数を示します。

SQL OLAP 関数	同等の RISQL 表示関数
RANK	RANK
DENSE_RANK、DENSERANK	
NTILE	NTILE、TERTILE
RATIO_TO_REPORT、RATIOTOREPORT	RATIOTOREPORT
ROW_NUMBER、ROWNUMBER	CUME(1)
AVG	MOVINGAVG
COUNT	
MIN	
MAX	
SUM	CUME、MOVINGSUM

構文の詳細、および使用上の注意点については、『SQL Reference Guide』を参照してください。

累積合計

例題

2000 年 1 月の Aroma Roma コーヒーの日別売上は？同月の売上と販売数量について、累計小計も算出してください。

OLAP クエリ

```
select date, sum(dollars) as total_dollars,
    sum(total_dollars) over(order by date rows unbounded
preceding)  as run_dollars,
    sum(quantity) as total_qty,
    sum(total_qty) over(order by date rows unbounded
preceding)
        as run_qty
from period natural join sales natural join product
where year = 2000
    and month = 'JAN'
    and prod_name = 'Aroma Roma'
group by date
order by date;
```

RISQL クエリ

```
select date, sum(dollars) as total_dollars,
    cume(sum(dollars)) as run_dollars,
    sum(quantity) as total_qty,
    cume(sum(quantity)) as run_qty
from period natural join sales
    natural join product
where year = 2000
    and month = 'JAN'
    and prod_name = 'Aroma Roma'
group by date
order by date;
```

実行結果

Date	Total_Dollars	Run_Dollars	Total_Qty	Run_Qty
2000-01-02	855.50	855.50	118	118
2000-01-03	536.50	1392.00	74	192
2000-01-04	181.25	1573.25	25	217
2000-01-05	362.50	1935.75	50	267
2000-01-06	667.00	2602.75	92	359
2000-01-07	659.75	3262.50	91	450
2000-01-08	309.50	3572.00	54	504
2000-01-09	195.75	3767.75	27	531
2000-01-10	420.50	4188.25	58	589
2000-01-11	547.50	4735.75	78	667
2000-01-12	536.50	5272.25	74	741
2000-01-13	638.00	5910.25	88	829
2000-01-14	1057.50	6967.75	150	979
2000-01-15	884.50	7852.25	122	1101
2000-01-16	761.25	8613.50	105	1206
2000-01-17	455.50	9069.00	66	1272
2000-01-18	768.50	9837.50	106	1378
2000-01-19	746.75	10584.25	103	1481
2000-01-20	261.00	10845.25	36	1517
2000-01-21	630.75	11476.00	87	1604
2000-01-22	813.75	12289.75	115	1719
...				

OLAP SUM 関数

OVER() 句を含めると、単純な SET 関数 (SUM、MIN、MAX、COUNT、AVG) と OLAP 集約関数は区別されます。

OLAP SUM 関数は、以下のウィンドウ フレームが指定されると、累積合計を算出します。

```
rows unbounded preceding
```

ウィンドウ フレームの詳細は、『SQL Reference Guide』を参照してください。

OLAP ORDER BY 句は重要です。ORDER BY 句を指定すると、**OLAP SUM** 関数の入力行が適切にソートされます (この場合は、**Date** でソートされます)。最後の ORDER BY 句は、独立して扱われ、OLAP 関数の ORDER BY 句とは区別されます。

RISQL CUME 関数

CUME 関数もまた、累積合計を算出し、表示します。累積合計を算出するには、合計する数値列ごとに CUME 関数を検索項目リストに指定します。

```
CUME(expression)
```

expression は、列数または数値表現式です。ORDER BY 句は、CUME 関数が一定順序にソートされた行を端から処理するようになるために必要です。CUME 関数を使用して正しい累積合計を算出するには、必ず ORDER BY 句を指定する必要があります。

CUME 関数では、列の参照は必須ではありません。たとえば、以下のように指定します。

```
select cume(1) as row_num, order_no, price from orders;

ROW_NUM     ORDER_NO    PRICE
      1     3600        1200.46
      2     3601        1535.94
      3     3602        780.00
      4     3603        956.45
...
```

OLAP ROW_NUMBER を使用しても同じ結果が得られます。

```
select row_number() over() as row_num, order_no, price
from orders;
```

累積合計のリセット

例題

2000 年 1 月の各週の Aroma Roma コーヒーの累積売上は？

OLAP クエリ

```
select week, date, sum(dollars) as total_dollars,
    sum(total_dollars) over(partition by week order by date
        rows unbounded preceding) as run_dollars,
    sum(quantity) as total_qty,
    sum(total_qty) over(partition by week order by date
        rows unbounded preceding) as run_qty
from period natural join sales
    natural join product
where year = 2000
    and month = 'JAN'
    and prod_name = 'Aroma Roma'
group by week, date
order by week, date;
```

RISQL クエリ

```
select week, date, sum(dollars) as total_dollars,
    cume(sum(dollars)) as run_dollars,
    sum(quantity) as total_qty,
    cume(sum(quantity)) as run_qty
from period natural join sales
    natural join product
where year = 2000
    and month = 'JAN'
    and prod_name = 'Aroma Roma'
group by week, date
order by week, date
    reset by week;
```

実行結果

Week	Date	Total_Dollars	Run_Dollars	Total_Qty	Run_Qty
2	2000-01-02	855.50	855.50	118	118
2	2000-01-03	536.50	1392.00	74	192
2	2000-01-04	181.25	1573.25	25	217
2	2000-01-05	362.50	1935.75	50	267
2	2000-01-06	667.00	2602.75	92	359
2	2000-01-07	659.75	3262.50	91	450
2	2000-01-08	309.50	3572.00	54	504
3	2000-01-09	195.75	195.75	27	27
3	2000-01-10	420.50	616.25	58	85
3	2000-01-11	547.50	1163.75	78	163
3	2000-01-12	536.50	1700.25	74	237
3	2000-01-13	638.00	2338.25	88	325
3	2000-01-14	1057.50	3395.75	150	475
3	2000-01-15	884.50	4280.25	122	597
4	2000-01-16	761.25	761.25	105	105
4	2000-01-17	455.50	1216.75	66	171
4	2000-01-18	768.50	1985.25	106	277
4	2000-01-19	746.75	2732.00	103	380
4	2000-01-20	261.00	2993.00	36	416
4	2000-01-21	630.75	3623.75	87	503
4	2000-01-22	813.75	4437.50	115	618
...					

リザルト セットの各パーティション間に空白行を挿入するには、RISQL レポータ コマンドの SET COLUMN column_name SKIP LINE を使用します。column_name は PARTITION BY (または RESET BY) 列を示します。

```
(C) Copyright IBM Corp. 1991-2002. All rights reserved.
RISQL Reporter Version 06.20.0000(0)TST
RISQL> set column week skip line;
```

OLAP ウィンドウ パーティション

OVER() 句内に OLAP PARTITION BY 句を使用すると、パーティションで区切られた列の値が変更されたときに計算をリセットできます。OLAP 計算のパーティションは、1 つ以上の列に対して設定できます。

RISQL を使用した別の方法 : CUME と RESET BY の併用

最後の ORDER BY 句内に RESET BY 句を指定すると、CUME で累積小計をリセットできます。RESET BY サブ句は、ORDER BY 句の中に指定しなければなりません。

```
SELECT select_list
FROM table_list
[WHERE search_condition]
[GROUP BY <group_list>]
[HAVING <search_condition>]
[ORDER BY <order_list>
    [RESET BY <reset_list>]]
    [BREAK BY <order_reference>SUMMING
<select_reference_list>]
[SUPPRESS BY <column_list>];
```

RESET BY サブ句は、**Week** の値が変化すると、累積合計を 0 にリセットします。これを実行するためには、クエリを **Week** 列に基づいて並び替えるか、グループ分けする必要があります。したがって、GROUP BY、ORDER BY、および RESET BY の 3 つの句すべてに **Week** 列を指定します。

RESET BY 句で参照する列は、検索項目リストおよび ORDER BY 句に指定してください。ORDER BY 句と RESET BY 句では位置番号に基づく項目指定が使用できますが、GROUP BY 句では使用できません。

OLAP を使用した累積合計の比較

例題

2000 年 3 月における West 地区と South 地区の日別売上の累積合計を比較すると ?

OLAP クエリ

```
select t1.date, sales_cume_west, sales_cume_south
from
(select date, sum(dollars) as total_sales,
    sum(total_sales) over(order by date rows unbounded
preceding)
        as sales_cume_west
from market natural join store
    natural join sales
    natural join period
where year = 2000
    and month = 'MAR'
    and region = 'West'
group by date) as t1
join
(select date, sum(dollars) as total_sales,
    sum(total_sales) over(order by date rows unbounded
preceding)
        as sales_cume_south
from market natural join store
    natural join sales
    natural join period
where year = 2000
    and month = 'MAR'
    and region = 'South'
group by date) as t2
using(date)
order by date;
```

実行結果

Date	Sales_Cume_West	Sales_Cume_South
2000-03-01	2529.25	2056.75
2000-03-02	6809.00	4146.75
2000-03-03	9068.75	6366.55
2000-03-04	12679.35	8831.30
2000-03-05	16228.60	11100.55
2000-03-06	19653.30	12665.65
2000-03-07	23515.55	14882.90
2000-03-08	27207.80	17494.15
2000-03-09	31725.95	19745.40
2000-03-10	34836.20	21323.40
2000-03-11	38257.00	23256.15
2000-03-12	42498.20	25091.35
2000-03-13	46691.95	27362.85
2000-03-14	49797.60	29729.50
2000-03-15	53692.85	31614.10
2000-03-16	57925.15	33974.60
2000-03-17	60357.15	37369.75
2000-03-18	62478.05	39420.50
2000-03-19	65415.55	41737.00
2000-03-20	68076.55	43879.95
2000-03-21	71173.10	45899.70
2000-03-22	74768.70	48129.80
2000-03-23	78084.20	50528.80

Date	Sales_Cume_West	Sales_Cume_South
2000-03-24	82288.80	52381.45
2000-03-25	86317.65	55180.50
2000-03-26	90676.55	57325.75
2000-03-27	94865.55	59063.50
2000-03-28	97510.50	60975.35
2000-03-29	100513.85	62891.35
2000-03-30	104267.40	65378.75
2000-03-31	107222.15	68100.75

OLAP ORDER BY 句

順次演算に OLAP を使用する場合は、サブクエリ内にそれらを指定できるという利点があります。RISQL CUME 関数は、その結果が ORDER BY 句で定義された順序によって異なるため、サブクエリ内に指定することはできません (サブクエリに ORDER BY 句を含めることはできません)。

OLAP 関数にはこのような制限はありません。これは、OLAP の順序付け機能が関数に含まれており、各 OLAP 関数が、クエリの最後の ORDER BY 句に依存しない独自の ORDER BY 句を持っているためです。

このクエリは、アウター クエリの FROM 句にジョインされた 2 つのサブクエリを含みます。クエリ構造の詳細については、第 4 章「比較クエリ」を参照してください。

移動平均

売上は、時間とともに変化します。急激な変化があると、基本となる長期的動向がわからなくなります。移動平均は、こうした変動の影響を平滑化するのに使用します。たとえば、3 週間の移動平均を算出するには、連続した過去 3 週間の合計を 3 で割ります。

例題

San Jose と Miami にある店舗の、1999 年第 3 四半期の製品売上の 3 週間の移動平均は？

OLAP クエリ

```
select city, week, sum(dollars) as sales,
    string(avg(sales) over(partition by city order by city,
week
        rows 2 preceding),7,2) as mov_avg,
    sum(sales) over(partition by city order by week rows
unbounded
        preceding) as run_sales
from store natural join sales natural join period
where qtr = 'Q3_99' and city in ('San Jose', 'Miami')
group by city, week;
```

RISQL クエリ

```
select city, week, sum(dollars) as sales,
    string(movingavg(sum(dollars), 3), 7, 2) as mov_avg,
    cume(sum(dollars)) as run_sales
from store natural join sales natural join period
where qtr = 'Q3_99' and city in ('San Jose', 'Miami')
group by city, week
order by city, week
    reset by city;
```

以下のリザルト セットは、RISQL のリザルト セットを示しています。各パーティションの先頭 2 行では、MOVINGAVG 関数で NULL が返されています。OLAP を使用した場合は、これらの NULL は先頭行と先頭 2 行で算出された平均とそれぞれ置き換えられます。

実行結果

City	Week	Sales	Mov_avg	Run_sales
Miami	27	1838.55	NULL	1838.55
Miami	28	4482.15	NULL	6320.70
Miami	29	4616.70	3645.80	10937.40
Miami	30	4570.35	4556.40	15507.75
Miami	31	4681.95	4623.00	20189.70
Miami	32	3004.50	4085.60	23194.20
Miami	33	3915.90	3867.45	27110.10
Miami	34	4119.35	3679.91	31229.45
Miami	35	2558.90	3531.38	33788.35
Miami	36	4556.25	3744.83	38344.60
Miami	37	5648.50	4254.55	43993.10
Miami	38	5500.25	5235.00	49493.35
Miami	39	4891.40	5346.71	54384.75
Miami	40	3693.80	4695.15	58078.55
San Jose	27	3177.55	NULL	3177.55
San Jose	28	5825.80	NULL	9003.35
San Jose	29	8474.80	5826.05	17478.15
San Jose	30	7976.60	7425.73	25454.75
San Jose	31	7328.65	7926.68	32783.40
San Jose	32	6809.75	7371.66	39593.15
San Jose	33	7116.35	7084.91	46709.50
...				

STRING スカラ関数は、**Mov_Avg** 列に返される long 型数値を切り捨てるのに使用します。この関数の詳細は、『SQL Reference Guide』を参照してください。使用するスカラ関数にかかわらず、RISQL と OLAP 関数の結果に丸め誤差が生じる場合があります。

MOVINGAVG 関数

RISQL 移動平均を算出するには、平均する各数値列について MOVINGAVG 関数を検索項目リストに指定します。MOVINGAVG 関数の参照先は、平均する数値列または数値表現式、および平均する行数 (平滑化係数) を示す整数です。

```
MOVINGAVG(<n_expression>, n)
```

この例では、**City** 列に対する RESET BY サブ句によって、コントロール ブレークが設定されます。3 週間の移動平均を算出するには、3 週間という期間が経過していなければなりません。このため、コントロール ブレークに続く最初の 2 行は、NULL になります。

リザルト テーブルは一定順序にソートし、都市名が変わるたびに移動平均をリセットしなければなりません。各行が発生順にソートされていないと、予期しない結果が返されます。このため ORDER BY 句には、**City** 列と **Week** 列の両方を指定します。

順序に依存する RISQL 表示関数を含むクエリには、ORDER BY 句を使用することが推奨されます。

OLAP AVG 関数

以下のウィンドウ フレームで OLAP AVG 関数を使用すると、RISQL MOVINGAVG 関数と同じように機能します。

```
rows n preceding
```

n は、必須の平滑化係数を示す数値です。FOLLOWING キーワードを使用することもできますが、PRECEDING 構文では RISQL MOVINGAVG 関数と同じ結果が返されます。

OLAP ORDER BY を指定することによって、AVG 関数が行の適切な移動シーケンスに適用されるようになります。PARTITION BY 句は、RISQL RESET BY 句と同じように機能します。

移動合計

例題

2000 年 3 月に販売された Demitasse Ms コーヒーの売上の 7 日間の移動合計は？

OLAP クエリ

```
select date, sum(quantity) as day_qty,
    string(sum(day_qty) over(order by date rows 6
preceding),7,2)
        as mov_sum
from store natural join sales natural join period
    natural join product
where year = 2000 and month = 'MAR' and prod_name = 'Demitasse
Ms'
group by date
order by date;
```

RISQL クエリ

```
select date, sum(quantity) as day_qty,
    string(movingsum(sum(quantity), 7),7,2) as mov_sum
from store natural join sales natural join period
    natural join product
where year = 2000 and month = 'MAR' and prod_name = 'Demitasse
Ms'
group by date
order by date;
```

以下のリザルト セットは、OLAP のリザルト セットを示しています。すべての行に対して、移動合計が算出されています。RISQL を使用した場合は、最初の 6 行が NULL に設定されます。これは、7 つの行があってはじめて最初の移動合計が算出されるという RISQL のロジックのためです。

実行結果

Date	Day_Qty	Mov_Sum
2000-03-01	65	65.00
2000-03-02	19	84.00
2000-03-03	92	176.00
2000-03-04	91	267.00
2000-03-05	106	373.00
2000-03-06	92	465.00
2000-03-07	102	567.00
2000-03-08	21	523.00
2000-03-09	74	578.00
2000-03-10	81	567.00
2000-03-11	77	553.00
2000-03-12	127	574.00
2000-03-13	169	651.00
2000-03-14	31	580.00
2000-03-15	56	615.00
2000-03-16	40	581.00
2000-03-17	84	584.00
2000-03-18	34	541.00
2000-03-19	128	542.00
2000-03-20	97	470.00
2000-03-21	50	489.00
2000-03-22	147	580.00
...		

移動合計関数は移動平均関数と同様に、長期的な傾向を分析するために短期的変動を取り除く目的で使用します。たとえば、7 日間の移動合計を算出するには、連続した 7 日間の値を合計して算出します。

STRING スカラ関数は、**Mov_Sum** 列に返される long 型数値を切り捨てるのに使用します。この関数の詳細は、『SQL Reference Guide』を参照してください。使用するスカラ関数にかかわらず、RISQL と OLAP 関数の結果に丸め誤差が生じる場合があります。

RISQL MOVINGSUM **関数**

MOVINGSUM 関数では、合計する列を示す列名または数値表現式 (<n_expression>) 、ならびに合計する行数 (平滑化係数) を示す整数 (<n>) を指定しなければなりません。

```
MOVINGSUM(<n_expression>, n)
```

発生順にソートされていないデータに MOVINGSUM 関数を適用すると、予期しない結果が返されるため、ORDER BY 句を使用してください。前の例と同様に、ORDER BY 句と GROUP BY 句の両方に **Date** 列を指定しなければなりません。

OLAP SUM **関数**

OLAP SUM 関数は、以下のウィンドウ フレームが指定されると、移動合計を算出します。

```
rows n preceding
```

FOLLOWING キーワードも使用可能ですが、PRECEDING 構文は RISQL MOVINGSUM 関数と同じ結果を返します。

OLAP ORDER BY 句は重要です。ORDER BY 句を指定すると、**OLAP SUM** 関数の入力行が適切にソートされます (この場合は、**Date** で昇順にソートされます)。

データの順位付け

例題

1999 年 3 月の合計売上に基づいて、West 地区の店舗を順位付けすると？

OLAP クエリ

```
select store_name, district, sum(dollars) as total_sales,
    rank() over(order by total_sales desc) as sales_rank
from market natural join store
    natural join sales
    natural join period
where year = 1999
    and month = 'MAR'
    and region = 'West'
group by store_name, district;
```

RISQL クエリ

```
select store_name, district, sum(dollars) as total_sales,
    rank(total_sales) as sales_rank
from market natural join store
    natural join sales
    natural join period
where year = 1999
    and month = 'MAR'
    and region = 'West'
group by store_name, district;
```

実行結果

Store_Name	District	Total_Sales	Sales_Rank
Cupertino Coffee Supply	San Francisco	18801.50	1
San Jose Roasting Company	San Francisco	18346.90	2
Beaches Brew	Los Angeles	18282.05	3
Java Judy's	Los Angeles	17826.25	4
Instant Coffee	San Francisco	15650.50	5
Roasters, Los Gatos	San Francisco	12694.50	6

WHERE 句に指定した検索条件を満たす日別売上が **Sales** テーブルから抽出され、合計された後、順位付けされます。

RISQL RANK 関数

複数の値を順位付けするには、検索項目リストに以下のように指定します。

```
RANK(<expression>)
```

検索項目リストで、expression は任意のデータ型を示します。<expression> が NULL の場合は、RANK によって NULL が返されます。数値表現式の詳細は、『SQL Reference Guide』を参照してください。

RISQL RANK 関数は、複数の値を順位付けするのに使用します。グループ内の最大値を 1、その次に大きな値を 2... というように順位を割り付けます。順位は、値の順序ではなく、大きさで決まります。

RANK は、順序に依存する表示関数ではありません。ORDER BY 句を指定せずに RANK 関数を使用したクエリは、デフォルト動作として、順位付けの基準となる値の大きなものから順にリザルト セットをソートします。値の小さなものから順位付けする場合は、単項否定演算子を使って順位付けの対象となる列の符号を逆にしてください。

```
RANK(-<expression>)
```

たとえば、以下のように指定します。

```
rank(-dollars) as sales_rank
```

非数値データ型の場合は、データベース ロケール設定で指定された照合シーケンスに応じてレベル分けされます。

OLAP RANK 関数

IBM Red Brick Warehouse では、RANK() や DENSE_RANK() などの一連の OLAP 順位付け関数がサポートされています。

これらの関数には引数を指定する必要はありませんが、その代わりに順序付けの基準となる列または式が OLAP ORDER BY 句によって定義されます。OLAP 関数のデフォルトのソート順は昇順 (ASC) です。これは、ビジネス クエリで頻繁に必要とされる順位付けではありません。上位から下位 (1 が最大値) に順位付けする場合は、OLAP ORDER BY 句に DESC キーワードを指定する必要があります。

DENSE_RANK 関数は、次の点で RANK と異なります。2 つ以上の行の順位が等値であっても、順位付けの連続番号に切れ目は生じません。つまり、2 つの行の順位が 8 でも、次の順位は 9 になります (以下参照)。

```
RISQL> select prod_name, dollars,
>      dense_rank() over(order by dollars desc) as dense_rank,
>        rank() over(order by dollars desc) as tie_rank
> from product natural join sales natural join period
> where date = '01-04-2000'
> ;
PROD_NAME                      DOLLARS   DENSE_RANK TIE_RANK
Espresso Machine Italiano       499.75            1        1
Cafe Au Lait                    392.00            2        2
Veracruzano                     360.00            3        3
Lotta Latte                     328.00            4        4
NA Lite                         306.00            5        5
Colombiano                      283.50            6        6
Darjeeling Special              207.00            7        7
Colombiano                      202.50            8        8
Colombiano                      202.50            8        8
Expresso XO                     201.50            9       10
Xalapa Lapa                     195.50           10       11
Lotta Latte                     168.00           11       12
Xalapa Lapa                     162.00           12       13
Veracruzano                     157.50           13       14
Veracruzano                     152.00           14       15
Demitasse Ms                    143.50           15       16
Cafe Au Lait                    136.00           16       17
Colombiano                      128.25           17       18
Aroma Roma                      123.25           18       19
Darjeeling Special              120.00           19       20
```

WHEN 句の使用

例題

Olympic Coffee Company の 2000 年第 1 四半期における Breakfast Blend 紅茶の売上上位 10 日間は ? 数量に基づく順位は ?

OLAP クエリ

```
select date, day, dollars as day_sales,
    rank() over(order by day_sales desc) as sales_rank,
    quantity as day_qty,
    rank() over(order by day_qty desc) as qty_rank
from product natural join sales
    natural join period
    natural join store
where qtr = 'Q1_00'
    and prod_name like 'Break%'
    and store_name like 'Olympic%'
when sales_rank <= 10
order by date;
```

RISQL クエリ

```
select date, day, dollars as day_sales,
    rank(day_sales) as sales_rank,
    quantity as day_qty,
    rank(day_qty) as qty_rank
from product natural join sales
    natural join period
    natural join store
where qtr = 'Q1_00'
    and prod_name like 'Break%'
    and store_name like 'Olympic%'
when sales_rank <= 10
order by date;
```

実行結果

Date	Day	Day_Sales	Sales_Rank	Day_Qty	Qty_Rank
2000-01-21	FR	30.00	9	8	9
2000-02-01	TU	56.25	3	15	2
2000-02-08	TU	30.00	9	8	9
2000-02-22	TU	71.25	1	19	1
2000-02-23	WE	41.25	7	11	6
2000-03-03	FR	59.50	2	14	4
2000-03-11	SA	55.25	5	13	5
2000-03-16	TH	56.25	3	15	2
2000-03-22	WE	38.25	8	9	8
2000-03-23	TH	42.50	6	10	7

順位付けする値が等しければ、同一順位になります。たとえば、この例では売上が第 3 位の行が 2 つあります。

WHEN 句

順位付けの対象となる値の算出後、リザルト テーブルに返す行を限定するには、WHEN 句を指定します。

```
SELECT select_list
FROM table_name
[WHERE search_condition]
[GROUP BY group_list]
[HAVING search_condition]
[WHEN condition]
[ORDER BY order_list]
    [RESET BY reset_list]
    [BREAK BY order_reference SUMMING select_reference_list]
[SUPPRESS BY column_list];
```

たとえば、全製品の売上を順位付けし、上位 10 製品や下位 10 製品を表示したり、検索条件で表現できるほかの組み合わせを指定することができます。この機能は、RISQL 順位付け関数と OLAP 順位付け関数の両方に該当します。

グループ別に値をレベル付けする ：NTILE

例題

1999 年の年間総売上に基づく上位 25% と下位 25% の製品は？

OLAP クエリ

```
select prod_name, sum(dollars) as total_sales,
    ntile(4) over(order by total_sales desc) as sales_rank
from sales natural join product
    natural join period
where year = 1999
group by prod_name
when sales_rank in (1, 4);
```

RISQL クエリ

```
select prod_name, sum(dollars) as total_sales,
    ntile(total_sales, 4) as sales_rank
from sales natural join product
    natural join period
where year = 1999
group by prod_name
when sales_rank in (1, 4);
```

実行結果

Prod_name	Total_sales	Sales_rank
Demitasse Ms	304727.00	1
Xalapa Lapa	263353.00	1
NA Lite	262162.00	1
Lotta Latte	251713.00	1
Cafe Au Lait	251086.50	1
Expresso XO	229201.25	1
Veracruzano	227769.50	1
La Antigua	223528.25	1
Aroma Roma	218574.75	1
Colombiano	218009.50	1
Aroma Sounds CD	5937.00	4
Aroma Sounds Cassette	5323.00	4
French Press, 4-Cup	4570.50	4
Spice Jar	4073.00	4
French Press, 2-Cup	3042.75	4
Travel Mug	1581.75	4
Easter Sampler Basket	1500.00	4
Coffee Mug	1258.00	4
Christmas Sampler	1230.00	4

このクエリは、1999 年の年間総売上に基づいて各製品を第 1 位、第 2 位、第 3 位、第 4 位のいずれかにレベル分けします。WHEN 句により、中位 50% (第 2 位と第 3 位) がリザルト セットから削除されます。

NTILE 関数を使うと、各値を第 1 (最高) から第 n (最低) までのレベルにグループ分けできます。NTILE 関数は、グループ内の各値の相対的な大きさに基づいてレベルを割り付けます。

レベルの境界上に複数の等値がある場合は、隣接グループに分散されます。指定したレベル数で割り切れない値があれば、NTILE 関数により剰余の行を上位グループに入れます。

RISQL NTILE 関数

複数の値を 100 のグループに等分にレベル分けする場合は、NTILE 関数、数値表現式またはレベル分けする列、100 の順に検索項目リストに指定します。

```
NTILE(<expression>, 100)
```

<expression> が NULL の場合は、NTILE によって NULL が返されます。数値表現式の詳細は、『SQL Reference Guide』を参照してください。

OLAP NTILE 関数

OLAP RANK 関数と同様で、NTILE 関数も OLAP ORDER BY 句の expression に依存します。関数に必要なのは、1 つの引数 (tile 値) のみです。

```
NTILE(n) OVER(ORDER BY <expression> DESC)
```

OLAP RANK 関数の前の使用例で示したように、上位から下位の expression 値に基づいて順位付けを行う場合は DESC キーワードを指定する必要があります。

重要: NTILE 関数を CASE 式で使用することにより、順序付けした値を不均等グループに再配分して、NTILE が割り当てたデフォルトの数値を有意なラベルに置き換えることができます。使用例は、「CASE 式と併用した NTILE 関数の使用」を参照してください。

CASE 式と併用した NTILE 関数の使用

例題

West 地区の店舗において、1998 年第 2 週の総売上に基づく上位 20%、中位 60%、下位 20% の製品は？

OLAP クエリ

```
select prod_name, sum(quantity) as quantity,
    sum(dollars) as sales,
    case ntile(5) over(order by sales desc)
        when 1 then 'TOP_20'
        when 2 then 'MID_60'
        when 3 then 'MID_60'
        when 4 then 'MID_60'
        when 5 then 'LOW_20' end as grp
from market natural join store natural join sales
        natural join period natural join product
where year = 1998 and week = 2 and region = 'West'
group by prod_name;
```

RISQL クエリ

```
select prod_name, sum(quantity) as quantity,
    sum(dollars) as sales,
    case ntile(sales, 5)
        when 1 then 'TOP_20'
        when 2 then 'MID_60'
        when 3 then 'MID_60'
        when 4 then 'MID_60'
        when 5 then 'LOW_20' end as grp
from market natural join store natural join sales
    natural join period natural join product
where year = 1998 and week = 2 and region = 'West'
group by prod_name;
```

実行結果

Prod_Name	Qty	Sales	Grp
Expresso XO	368	2887.00	TOP_20
Aroma Roma	246	1783.50	TOP_20
Colombiano	257	1757.75	TOP_20
Darjeeling Special	143	1655.00	TOP_20
Lotta Latte	198	1621.00	TOP_20
La Antigua	213	1589.25	TOP_20
Demitasse Ms	151	1503.75	MID_60
Xalapa Lapa	163	1395.50	MID_60
Ruby's Allspice	183	1018.50	MID_60
Veracruzano	120	925.50	MID_60
NA Lite	100	900.00	MID_60
Cafe Au Lait	106	869.50	MID_60
Assam Gold Blend	104	636.50	MID_60
English Breakfast	137	561.50	MID_60
Aroma t-shirt	44	481.80	MID_60
Coffee Sampler	16	480.00	MID_60
Assam Grade A	114	380.00	MID_60
Darjeeling Number 1	69	378.25	MID_60
Irish Breakfast	81	345.75	MID_60
Breakfast Blend	82	322.00	MID_60
Gold Tips	91	320.75	MID_60
Earl Grey	80	305.00	MID_60

Prod_Name	Qty	Sales	Grp
Special Tips	74	253.50	MID_60
Aroma Sheffield Steel Teapot	7	210.00	LOW_20
Espresso Machine Italiano	1	99.95	LOW_20
Aroma baseball cap	11	87.45	LOW_20
Spice Sampler	4	48.00	LOW_20
Travel Mug	1	10.95	LOW_20

(2 / 2)

レベル別グループに値を分ける :CASE と NTILE

NTILE 関数と CASE 式を効率的に組み合わせると、値の集合をレベル分けし、再分類できます。たとえば、NTILE を使って 5 つのグループに値を等分し、CASE 式を使ってレベル別グループに再分類できます。レベル別グループを、曲線状に分布させたいときなどに使用します。この機能は、RISQL NTILE 関数と OLAP NTILE 関数の両方に該当します。

このクエリは、CASE 式内に NTILE 関数を使用し、5 等分した値を 3 つの非均等グループに再分類します。最終的なリザルト セットにおいて上位 20 % は `TOP_20`、中位 60 % は `MID_60`、下位 20 % は `LOW_20` と表されます。評価結果が 2、3、4 のいずれかの場合は `MID_60` に置き換えられ、5 の場合は `LOW_20` に置き換えられます。

このクエリの末尾に WHEN 句を追加することで、リザルト セットの指定部分を削除できます。たとえば、以下のように指定します。

```
when grp = 'MID_60'
```

CASE の構文

CASE 式は、指定された列の値を別の値に置き換える条件付きスカラ式です。検索項目リスト内で使用できます。

```
CASE expression WHEN result THEN result1 ELSE result2
    END AS col_alias
```

<expression>　任意の有効な式

<result>　式の評価結果として返される値

<result1>　<result> と置き換えられる値

<result2>　expression が result に評価結果を返さない場合に使用されるデフォルト値 (オプション)

通常は、複数の WHEN...THEN 条件式を使用し、各々を目的の値に置き換えます。

重要： CASE 式の形式は、シンプル形式またはサーチ形式のどちらかです。この例では、シンプル形式を使用しています。詳細は、『SQL Reference Guide』を参照してください。サーチ形式の例については、4-6 ページ「CASE 式の使用」を参照してください。

TERTILE 関数の使用

例題

1999 年の Earl Grey 紅茶の売上数量に基づき、West 地区と South 地区の都市を上位、中位、下位にレベル分けすると？

RISQL クエリ

```
select city, sum(quantity) as qty_1999,
    tertile(sum(quantity)) as q_rk
from market natural join store
    natural join sales
    natural join product
    natural join period
where year = 1999
    and prod_name like 'Earl Grey%'
    and region in ('West', 'South')
group by city;
```

実行結果

City	Qty_1999	Q_RK
San Jose	1469	H
Los Angeles	911	H
Phoenix	814	H
Los Gatos	805	M
Miami	782	M
Cupertino	778	M
Houston	768	L
New Orleans	684	L
Atlanta	614	L

上位、中位、下位に値をレベル分けする：TERTILE

RISQL TERTILE 関数を使うと、数値グループの各値を上位、中位、下位のいずれかにレベル分けできます。TERTILE 関数は、グループにおける各値の相対的な大きさに基づいて H、M、L のいずれかの文字を割り付けます。

このクエリは、1999 年の Earl Grey 紅茶の売上数量に基づいて各都市をレベル分けします。リザルト テーブルが都市グループに分類され、都市ごとに **Quantity** 列の年間数量が合計されます。

3 で割り切れなければ、TERTILE 関数により剰余の行は上位グループに入れられます。レベルの境界上に複数の等値がある場合は、隣接グループに分散されます。

1 つのグループを 3 つのレベルに分類するには、TERTILE 関数と数値表現式またはレベル分けする列を検索項目リストに指定します。

```
TERTILE(<expression>)
```

<expression>　列名または数値表現式

<expression> が NULL の場合は、TERTILE によって NULL が返されます。TERTILE 関数が参照する列は数値列ですが、結果は必ず文字列になります。

TERTILE の代用関数 OLAP NTILE

TERTILE 関数に相当する OLAP 関数はありません。ただし、TERTILE 関数は NTILE の特殊な形式であると言えます。つまり、NTILE(3) 関数を CASE 式と併用すれば、同じ結果を得ることができます。

```
select city, sum(quantity) as qty_1999,
    case ntile(3) over(order by qty_1999 desc)
        when 1 then 'H'
        when 2 then 'M'
        when 3 then 'L'
    end as q_rk
from ...
```

比率の算出

例題

1999 年の第 3 四半期の San Jose と Los Angeles における Xalapa Lapa コーヒーの総売上に占める各月の売上の比率は？

OLAP クエリ

```
select city, month, sum(dollars) as total_sales,
    dec(ratiotoreport(total_sales) over()*100,5,2) as
pct_of_sales
from store natural join sales
    natural join product
    natural join period
where prod_name like 'Xalapa%'
    and qtr = 'Q3_99'
    and city in ('San Jose', 'Los Angeles')
group by city, month;
```

RISQL クエリ

```
select city, month, sum(dollars) as total_sales,
    dec(ratiotoreport(total_sales)*100,5,2) as pct_of_sales
from store natural join sales
    natural join product
    natural join period
where prod_name like 'Xalapa%'
    and qtr = 'Q3_99'
    and city in ('San Jose', 'Los Angeles')
group by city, month;
```

実行結果

City	Month	Total_Sales	Pct_of_Sales
San Jose	JUL	2499.50	26.99
Los Angeles	JUL	1627.00	17.57
San Jose	AUG	1004.00	10.84
Los Angeles	AUG	995.00	10.74
San Jose	SEP	1802.00	19.46
Los Angeles	SEP	1334.00	14.40

比率を算出する OLAP と RISQL クエリの結果は、若干異なる場合があります。RISQL RATIOTOREPORT 関数は、常に精度の低い浮動小数点データ型を返します。OLAP RATIOTOREPORT の計算は関数に指定する引数によって異なります (関数の引数が正確であれば、より精度の高い正確な結果が得られますが、引数が不正確であれば、結果も不正確になります)。

パーセントで比率を算出する :RATIOTOREPORT*100

RATIOTOREPORT 関数は、リザルト セットの 1 列について、合計値に対する数値行の値の比率を算出します。たとえば、複数製品の売上を表示した列の場合は、全製品の合計売上に対する比率として、その列の割合が表示されます。

パーセントで比率を算出するには、expression の後に以下のように記述します。

```
*100
```

このクエリは、1999 年の第 3 四半期に San Jose と Los Angeles の各店舗が販売した Xalapa Lapa コーヒーの総売上に対する月間売上の比率を表示します。

(**Pct_of_Sales** 列の値を合計すると、ちょうど 100 になります)

RISQL RATIOTOREPORT 関数

列に表示されたすべての値の合計値に対する各列値の比率を算出するには、RATIOTOREPORT 関数を検索項目リストに指定し、数値表現式か、数値を格納している列の名前を指定します。

```
RATIOTOREPORT(<expression>)
```

<expression> が NULL の場合は、RATIOTOREPORT によって NULL が返されます。数値表現式の詳細は、『SQL Reference Guide』を参照してください。

RATIOTOREPORT 関数は、ORDER BY 句の RESET BY サブ句によって、グループごとにリセットできます。RESET BY の詳細は、3-10 ページを参照してください。

OLAP RATIOTOREPORT または RATIO_TO_REPORT 関数

OLAP RATIOTOREPORT 関数および RATIO_TO_REPORT 関数は同義です。RISQL 関数と同様に、これらの関数には、引数として数式を指定する必要があります。

```
RATIO_TO_REPORT(<expression>) OVER()
```

OLAP ORDER BY および PARTITION BY 句はオプションですが、空の OVER 句は指定する必要があります。

DATEADD 関数の使用

例題

指定した日付より 90 日前の日付と、90 日後の日付を算出してください。

SQL 文例

```
select dateadd(day, -90, date) as due_date,
    date as cur_date,
    dateadd(day, 90, date) as past_due
from period
where year = 2000
    and month = 'JAN';
```

実行結果

Due_Date	Cur_Date	Past_Due
1999-10-03	2000-01-01	2000-03-31
1999-10-04	2000-01-02	2000-04-01
1999-10-05	2000-01-03	2000-04-02
1999-10-06	2000-01-04	2000-04-03
1999-10-07	2000-01-05	2000-04-04
1999-10-08	2000-01-06	2000-04-05
1999-10-09	2000-01-07	2000-04-06
1999-10-10	2000-01-08	2000-04-07
1999-10-11	2000-01-09	2000-04-08
1999-10-12	2000-01-10	2000-04-09
1999-10-13	2000-01-11	2000-04-10

(1 / 2)

Due_Date	Cur_Date	Past_Due
1999-10-14	2000-01-12	2000-04-11
1999-10-15	2000-01-13	2000-04-12
1999-10-16	2000-01-14	2000-04-13
1999-10-17	2000-01-15	2000-04-14
1999-10-18	2000-01-16	2000-04-15
1999-10-19	2000-01-17	2000-04-16
1999-10-20	2000-01-18	2000-04-17
1999-10-21	2000-01-19	2000-04-18
1999-10-22	2000-01-20	2000-04-19
1999-10-23	2000-01-21	2000-04-20
1999-10-24	2000-01-22	2000-04-21
1999-10-25	2000-01-23	2000-04-22
1999-10-26	2000-01-24	2000-04-23
1999-10-27	2000-01-25	2000-04-24
1999-10-28	2000-01-26	2000-04-25
1999-10-29	2000-01-27	2000-04-26
1999-10-30	2000-01-28	2000-04-27
1999-10-31	2000-01-29	2000-04-28
1999-11-01	2000-01-30	2000-04-29
1999-11-02	2000-01-31	2000-04-30

(2 / 2)

日付の増減：DATEADD

DATEADD 関数は、以下の 3 つの引数から日付時間を算出して返します。

- 日、月、年などの増減単位を指定する datepart
- 正か負で示す増減値
- 加算または減算される値 (列名か、日付時間式)

機能	戻り値
DATEADD(day, 90, '07-01-99')	1999-09-29
DATEADD(month, 3, '07-01-99')	1999-10-01
DATEADD(year, 1, '07-01-99')	2000-07-01

例題について

このクエリは、指定日から 90 日前の日付と 90 日後の日付を算出します。
DATEADD 関数は、ANSI SQL-92 標準の日時フォーマットで値を返します。

DATENAME 関数を使うと、DATETIME の値を月名に変更できます。以下のクエリは、WHERE 句に DATENAME 関数を使用しています。

```
select datename(month, dateadd(day, -90, date)) as prior,
    datename(month, date) as cur,
    datename(month, dateadd(day, 90, date)) as next
from period
where datename(yy, date) = '2000'
    and month = 'JAN';
```

Prior	Cur	Next
October	January	March
October	January	April
October	January	April
October	January	April
October	January	April
October	January	April
...		

2000 年の 2 月は 29 日まであるため、1 月 1 日に 90 日を加算すると 3 月 31 日になります。その結果、Next 列の先頭行は March になります。DATETIME 関数の詳細は、『SQL Reference Guide』を参照してください。

DATEDIFF 関数の使用

例題

1999 年の全店クリスマス特別セールは、何日間実施されましたか？

SQL 文例

```
select promo_desc, year,
    datediff(day, end_date, start_date)+1 as days_on_promo
from promotion p, period d
where p.start_date = d.date
    and promo_desc like 'Christmas%'
    and year = 1999;
```

実行結果

Promo_Desc	Year	Days_on_Promo
Christmas special	1999	31

経過日数の算出 :DATEDIFF

DATEDIFF 関数は、以下の 3 つの引数から日付時間を算出して返します。

- 日、月、年などの増減単位を指定する datepart
- データ型が DATE、TIME、TIMESTAMP いずれかの 2 つの日付時間式

機能	戻り値
DATEDIFF(day, '07-01-00','01-01-00')	182
DATEDIFF(month, '07-01-00','01-01-00')	6
DATEDIFF(quarter '07-01-00','01-01-00')	2

例題について

このクエリは、全店特別セールの開始日から終了日までに経過した日数を算出します。結果を返すには、次の DATEDIFF 関数を使って **Promotion** テーブル内の日付時間の値を処理させます。

```
datediff(day, end_date, start_date)+1
```

End_Date の値と Start_Date の値との差は 30 日ですが、販売促進期間には Start_Date の値と End_Date の値の両方が含まれるため、+1 を加えて 31 日にする必要があります。

使用上の注意

このクエリは、プライマリ キーと外部キーとの対応関係がない 2 つのテーブルをジョインする例でもあります。ジョインする各列は比較可能な日時データ型です。

```
where p.start_date = d.date
```

システム テーブルを含め、比較可能な列に基づいて任意の 2 テーブルをジョインできます。

この場合のジョインは、**Period** テーブルから Year の値を抽出することを目的としています。この値は、**Promotion** テーブルの datetime 列からも抽出できます。

EXTRACT 関数の使用

例題

1998 年初頭の 6 週間について、曜日、月名、日、月を **Period** テーブルの datetime の値から抽出すると？

SQL 文例

```
select datename(weekday, date) as day_name,
    extract(weekday from date) as day_num,
    extract(day from date) as day,
    extract(dayofyear from date) as day_yr,
    datename(month, date) as mo_name,
    extract(month from date) as mo_num
from period
where extract(year from date) = 1998
    and extract(week from date) < 7;
```

実行結果

Day_Name	Day_Num	Day	Day_Yr	Mo_Name	Mo_Num
Thursday	5	1	1	January	1
Friday	6	2	2	January	1
Saturday	7	3	3	January	1
Sunday	1	4	4	January	1
Monday	2	5	5	January	1
Tuesday	3	6	6	January	1

(1 / 2)

Day_Name	Day_Num	Day	Day_Yr	Mo_Name	Mo_Num
Wednesday	4	7	7	January	1
Thursday	5	8	8	January	1
Friday	6	9	9	January	1
Saturday	7	10	10	January	1
Sunday	1	11	11	January	1
Monday	2	12	12	January	1
Tuesday	3	13	13	January	1
Wednesday	4	14	14	January	1
Thursday	5	15	15	January	1
Friday	6	16	16	January	1
Saturday	7	17	17	January	1
Sunday	1	18	18	January	1
Monday	2	19	19	January	1
Tuesday	3	20	20	January	1
Wednesday	4	21	21	January	1
Thursday	5	22	22	January	1
Friday	6	23	23	January	1
Saturday	7	24	24	January	1
Sunday	1	25	25	January	1
Monday	2	26	26	January	1
Tuesday	3	27	27	January	1

datepart を整数として表示する：EXTRACT

EXTRACT 関数は、DATETIME の値の一部を示す整数を返します。次に示す 2 つの引数が必要です。

- 日、月、年などの増減単位を指定する datepart
- 日付時間式 (列名または DATETIME 式)

機能	戻り値
extract(weekday from date_col)	曜日を整数で表したもの (1, 2, ..., 7)
extract(day from date_col)	各月の日付を整数で表したもの (1, 2, ..., 31)

例題について

このクエリは、DATENAME および EXTRACT のスカラ関数を使い、1998 年初頭の 6 週間の曜日、月名、日、月を返します。

1 年の最初の週を除き、1 週間は、データベース ロケールで指定したテリトリに応じて、通常日曜または月曜から始まります。ロケール指定の詳細は、『Administrator's Guide』および『Installation and Configuration Guide』を参照してください。

まとめ

この章では次の内容を説明しました。

- データ分析には、RISQL 表示関数および SQL OLAP 関数を使用する。
- レベル分けした値を不均等グループに再配分する。
- DATETIME スカラ関数を使い、DATETIME 列から日付時間に関する情報を算出したり抽出する。

分析関数

SQL OLAP 関数および RISQL 表示関数は、順位、比率、移動合計、平均などに関連するビジネス上の多岐にわたる質問に対応するために使用されます。この章の例では、OLAP 関数または RISQL 関数のいずれかを使用して取り扱うことのできる計算について説明してきましたが、一般的に OLAP 関数の方が柔軟性に富んでおり、ウィンドウ フレームの機能は特に役立ちます。

CASE 式

検索項目リスト内の CASE 式は、列の値をほかの指定値に置き換えるのに使用します。たとえば、分析関数が返した数値を、意味のある文字列に置き換えることができます。

CASE 式のほかの使い方は、4-6 ページ「CASE 式の使用」にも記述されています。

DATETIME 関数

機能	動作
DATEADD	日付時間の値に一定期間を加算する。
DATEDIFF	2 つの日付時間の差異を算出する。
DATENAME	日付時間の値から、datepart の部分を文字列として抽出する。
EXTRACT	日付時間の値から、datepart の部分を整数として抽出する。

第 4 章

比較クエリ

この章について

この章は、データを比較するクエリを中心に構成されています。まず、クエリ作成者が直面する課題について説明します。たとえば、SQL を使用して、読みにくい縦表示の標準的なリザルト セットの代わりに、スプレッドシートつまりクロス集計レポートを作成する方法を説明します。これには、CASE 式かサブクエリを使用します。

最初にある値のグループを比較する単純で簡潔な方法として、CASE 式について説明します。次に、FROM 句と検索項目リストのサブクエリを例示します。サブクエリは、複数グループのデータを比較し、比較した値に対して演算を組み入れるという 2 つの操作を行えます。たとえば、一定期間の値に占める比率を算出することができます。

この章では、WHERE 句に条件として記述されるサブクエリについて説明します。これは、単純な比較クエリを作成するときに役立ちます。この章の最後で、ALL、EXISTS、SOME、および ANY のプレディケートについても説明します。これらのプレディケートを使用してサブクエリの結果に対する条件を表すことができます。

SQL でデータを比較する

例題

1998 年のパッケージ入りコーヒーの売上を、West 地区の各店舗で比較すると？

SQL 文

```
select store_name, prod_name, sum(dollars) as sales
    from market natural join store
        natural join sales
        natural join period
        natural join product
        natural join class
where region like 'West%'
    and year = 1998
    and class_type = 'Pkg_coffee'
group by store_name, prod_name
order by store_name, prod_name;
```

実行結果

Store_Name	Prod_Name	Sales
Beaches Brew	Aroma Roma	3483.50
Beaches Brew	Cafe Au Lait	3129.50
Beaches Brew	Colombiano	2298.25
Beaches Brew	Demitasse Ms	4529.25
Beaches Brew	Expresso XO	4132.75
Beaches Brew	La Antigua	4219.75
Beaches Brew	Lotta Latte	3468.00
Beaches Brew	NA Lite	4771.00
Beaches Brew	Veracruzano	4443.00

(1 / 2)

Store_Name	Prod_Name	Sales
Beaches Brew	Xalapa Lapa	4304.00
Cupertino Coffee Supply	Aroma Roma	4491.00
Cupertino Coffee Supply	Cafe Au Lait	4375.50
Cupertino Coffee Supply	Colombiano	2653.50
Cupertino Coffee Supply	Demitasse Ms	3936.50
Cupertino Coffee Supply	Expresso XO	4689.25
Cupertino Coffee Supply	La Antigua	2932.00
Cupertino Coffee Supply	Lotta Latte	5146.00
Cupertino Coffee Supply	NA Lite	4026.00
Cupertino Coffee Supply	Veracruzano	3285.00
Cupertino Coffee Supply	Xalapa Lapa	5784.00
Instant Coffee	Aroma Roma	3485.25
Instant Coffee	Cafe Au Lait	3599.50
Instant Coffee	Colombiano	3321.75
Instant Coffee	Demitasse Ms	5422.25
Instant Coffee	Expresso XO	2851.00
Instant Coffee	La Antigua	2937.25
Instant Coffee	Lotta Latte	4783.50
Instant Coffee	NA Lite	3740.00
Instant Coffee	Veracruzano	4712.00
Instant Coffee	Xalapa Lapa	3698.00
...		

単純な比較クエリ

特定の店舗における複数製品の売上は、単純な SELECT 文を使っても抽出できますが、リザルト テーブルの値が比較しにくいフォーマットになります。たとえば、前出のリザルト セットの一部を見ると、La Antigua コーヒーを販売した店舗は West 地区にいくつかありますが、その売上値だけをとりあげて比較するのは困難です。

このようなデータは、スプレッドシート形式にすると比較しやすくなります。スプレッドシートつまり「クロス集計」レポートを作成する方法は 2 つあります。CASE 式を使う方法と、サブクエリを使う方法です。この章では、両方の比較クエリの作成方法を例を示して説明します。

例題について

このクエリは、West 地区の各店舗で販売したパッケージ入りコーヒーの 1998 年の売上を結果として返しますが、この出力データのフォーマットでは製品単位や店舗単位の値の比較は容易ではありません。

CASE 式の使用

例題

1998 年のパッケージ入りコーヒーの売上を、West 地区の各店舗で比較すると？

SQL 文例

```
select prod_name,
    sum(case when store_name = 'Beaches Brew'
        then dollars else 0 end) as Beaches,
    sum(case when store_name = 'Cupertino Coffee Supply'
        then dollars else 0 end) as Cupertino,
    sum(case when store_name = 'Roasters, Los Gatos'
        then dollars else 0 end) as RoastLG,
    sum(case when store_name = 'San Jose Roasting Company'
        then dollars else 0 end) as SJRoastCo,
    sum(case when store_name = 'Java Judy''s'
        then dollars else 0 end) as JavaJudy,
    sum(case when store_name = 'Instant Coffee'
        then dollars else 0 end) as Instant
from market natural join store
    natural join sales
    natural join period
```

```
    natural join product
    natural join class
where region like 'West%'
    and year = 1998
    and class_type = 'Pkg_coffee'
group by prod_name
order by prod_name;
```

実行結果

Prod_Name	Beaches	Cupertino	RoastLG	SJRoastCo	JavaJudy	Instant
Aroma Roma	3483.50	4491.00	4602.00	4399.25	3748.25	3485.25
Cafe Au Lait	3129.50	4375.50	4199.00	3620.00	4864.50	3599.50
Colombiano	2298.25	2653.50	4205.00	3530.75	3509.00	3321.75
Demitasse Ms	4529.25	3936.50	4347.75	5699.00	6395.25	5422.25
Expresso XO	4132.75	4689.25	4234.50	3811.00	5012.25	2851.00
La Antigua	4219.75	2932.00	3447.50	4323.00	2410.25	2937.25
Lotta Latte	3468.00	5146.00	4469.50	5103.50	4003.00	4783.50
NA Lite	4771.00	4026.00	3250.00	2736.00	4791.00	3740.00
Veracruzano	4443.00	3285.00	4467.00	3856.00	4510.00	4712.00
Xalapa Lapa	4304.00	5784.00	3906.00	3645.00	3182.00	3698.00

データ比較の解決方法 : CASE 式

読みやすいスプレッドシート形式で比較データを効率よく簡潔に表示するには、検索項目リスト内で CASE 式を使用します。CASE 演算は、指定された式を評価し、条件によって異なる値を返します。

CASE の構文

通常の CASE 比較クエリでは、まず評価の対象となる範囲全体について、メインクエリつまりアウター クエリの WHERE 句に制約を指定します。そして、検索項目リスト内の CASE 式により、結果をサブセットに分割します。

```
CASE WHEN <search_condition> THEN <result1> ELSE <result2>
    END AS col_alias
```

<search_condition>	評価結果が真または偽になる論理条件
<result1>	search_condition が真になったとき使用される値
<result2>	search_condition が偽になったとき使用されるデフォルト値

重要： CASE 式の形式は、シンプル形式またはサーチ形式のどちらかです。この例では、サーチ形式を使用しています。詳細は、『SQL Reference Guide』を参照してください。シンプル形式の例については、3-28 ページ「CASE 式と併用した NTILE 関数の使用」を参照してください。

例題について

このクエリは、前述のクエリと同じビジネス上の質問を表したものですが、この場合は、CASE 式を使用してリザルト セット内に 6 つの列を作成し、店舗ごとに 1 列ずつ割り当てて集計金額を格納しています。

使用上の注意

Java Judy's という名前の店舗に対する WHEN 条件では、アポストロフィを 2 つの一重引用符で表す必要があります。

```
when store_name = 'Java Judy''s'
```

1 つだけでは、アポストロフィが文字列の終了引用符として解釈されてしまい、「不完全な文字列」というエラーが返されます。

FROM 句でのサブクエリの使用

例題

San Jose における 1998 年の年間製品売上と、同年 1 月の同都市の製品売上を比較すると？

SQL 文例

```
select product, jan_98_sales, total_98_sales
from
        (select p1.prod_name, sum(dollars)
        from product p1 natural join sales s1
            natural join period d1 natural join store r1
        where d1.year = 1998 and month = 'JAN'
            and r1.city like 'San J%'
        group by p1.prod_name) as sales1(product,
jan_98_sales)

    natural join

        (select p2.prod_name, sum(dollars) as total_98_sales
        from product p2 natural join sales s2
            natural join period d2 natural join store r2
        where d2.year = 1998 and r2.city like 'San J%'
        group by p2.prod_name) as sales2(product,
total_98_sales)

order by product;
```

実行結果

Product	Jan_98_Sales	Total_98_Sales
Aroma Roma	1653.00	21697.50
Aroma Sheffield Steel Teapot	120.00	1122.00
Aroma Sounds Cassette	58.50	866.00
Aroma baseball cap	7.95	2960.15
Aroma t-shirt	470.85	4470.50
Assam Gold Blend	652.00	11375.00
Assam Grade A	352.00	5429.00
Breakfast Blend	608.25	6394.75
Cafe Au Lait	1936.50	24050.50
Colombiano	2148.00	22528.50
Darjeeling Number 1	867.50	8590.00
Darjeeling Special	1355.00	17787.50
Demitasse Ms	2163.00	35523.50
Earl Grey	540.50	6608.50
English Breakfast	393.00	5365.50
Espresso Machine Italiano	899.55	4397.80
Expresso XO	2935.50	27362.00
French Press, 2-Cup	104.65	1196.00
French Press, 4-Cup	19.95	1109.20
Gold Tips	440.00	5381.50
Irish Breakfast	703.25	7455.50
...		

柔軟な解決方法 : FROM 句のサブクエリ

サブクエリとは、クエリ内にあるかっこで囲んだクエリ式のことです。クエリに対するサブクエリは、アウター クエリに対するインナー クエリ、親クエリに対する子クエリとも呼ばれます。

例題について

1 つの値を、複数の値の合計と比較することはよくあります。このクエリは、San Jose における 1998 年 1 月の製品売上を、同年の年間製品売上と比較します。このようなクエリには複数の集約が必要なため、1 つのグループまたは範囲の値だけを対象とした CASE 式では表現できません。このため、FROM 句でサブクエリを使用して比較を行います。

重要: FROM 句中にサブクエリとして表現できるクエリは、この章で後述するように、検索項目リストでもサブクエリとして表すことができます。ただし、FROM 句のサブクエリの方が一般に高速であり、概念的にも理解しやすいでしょう。

使用上の注意

この例は、2 つのサブクエリの結果をジョインする標準 SQL クエリ式の柔軟性を利用しています。クエリ式の詳細は、『SQL Reference Guide』を参照してください。

サブクエリの結果から得られたテーブルは、ほかの参照テーブルとジョインすることができます。そのためには、FROM 句のサブクエリに相関名を割り付けなければなりません。ただし、サブクエリによって得られるテーブルの列名はオプションです。たとえば、この例のサブクエリからは、次のテーブルが得られます。

```
sales1(product, jan_98_sales)
sales2(product, total_98_sales)
```

上記のテーブルを **Product** 列に基づいてナチュラル ジョインすると、3 つの列 (メインクエリの 3 つの検索項目のデータ) を持つテーブルが作成されます。

```
product, jan_98_sales, total_98_sales
```

テーブル ジョインの例は、第 5 章「ジョインとユニオン」を参照してください。

演算と比較

例題

San Jose における 1998 年の年間製品売上に対する、同年 1 月の同都市の製品売上比率をパーセントで表すと？この比率に基づく上位 10 製品は？

SQL 文例

```
select product, jan_98_sales, total_98_sales,
    dec(100 * jan_98_sales/total_98_sales,7,2) as pct_of_98,
    rank(pct_of_98) as rank_pct
from
        (select p1.prod_name, sum(dollars)
        from product p1 natural join sales s1
            natural join period d1 natural join store r1
        where d1.year = 1998 and month = 'JAN'
            and r1.city like 'San J%'
        group by p1.prod_name) as sales1(product,
jan_98_sales)

    natural join

        (select p2.prod_name, sum(dollars)
        from product p2 natural join sales s2
            natural join period d2 natural join store r2
        where d2.year = 1998
            and r2.city like 'San J%'
        group by p2.prod_name) as sales2(product,
total_98_sales)

when rank_pct <= 10
order by product;
```

実行結果

Product	Jan_98_Sales	Total_98_Sales	Pct_of_98	Rank_Pct
Aroma Sheffield Steel Teapot	120.00	1122.00	10.69	4
Aroma t-shirt	470.85	4470.50	10.53	5
Breakfast Blend	608.25	6394.75	9.51	9
Colombiano	2148.00	22528.50	9.53	8
Darjeeling Number 1	867.50	8590.00	10.09	7
Espresso Machine Italiano	899.55	4397.80	20.45	1
Expresso XO	2935.50	27362.00	10.72	3
Irish Breakfast	703.25	7455.50	9.43	10
La Antigua	2643.25	22244.50	11.88	2
Lotta Latte	3195.00	31200.00	10.24	6

FROM 句のサブクエリによる演算

比較クエリのリザルト セットは、各種演算のソース データに使用できます。たとえば、ある製品の年間売上に対して月間売上が占める割合は、単純なパーセント演算で表すことができます。

```
100 * monthly_sales / annual_sales
```

市場、製品、期間などの比率は、複雑なものでも FROM 句のサブクエリで行うことができます。

例題について

このクエリは、その前の例題を基本とし、San Jose における年間売上に対する同都市の 1 月の月間売上を製品別に売上比率で表します。第 3 章で説明した RANK 表示関数を使い、この売上比率に基づいて各製品をランク付けし、上位 10 製品だけをリザルト セットにします。

Pct_of_98 列の値を合計しても、100 にはなりません。全製品の年間売上に対する月間売上の比率ではなく、製品別の 1 年間の売上に対する 1 月の売上比率だからです。

使用上の注意

メインクエリの検索項目リストは、リザルト テーブルの列名、列エリアス、またはその列名や列エリアスを含む式だけで構成します。たとえば、次の検索項目リストの項目は、サブクエリのナチュラル ジョインで派生したテーブル内で指定される列を、乗算 (*) および除算 (/) のオペランドとして使用しています。

```
dec(100 * jan_98_sales/total_98_sales,7,2) as pct_of_98
```

そして最後の項目は、上記の式から得られた列エリアスを RANK 関数の引数として使用しています。

```
rank(pct_of_98) as rank_pct
```

ランク付け関数のほかの例題と、WHEN 句の詳しい使い方は、第 3 章「データの解析」を参照してください。

各種の比率や達成度を算出するクエリには、同様の命令を繰り返す行が多数必要になる場合があります。長い SQL 文を RISQL マクロによって簡略化し、汎用化する方法の詳細は、第 6 章「マクロ、ビュー、テンポラリ テーブル」を参照してください。

検索項目リスト中のサブクエリの使用

例題

San Jose Roasting Company の Lotta Latte 製品について、1999 年 12 月の売上が 1998 年 12 月の日別平均売上を下回った日は？

1998 年の日別平均値も表示してください。

SQL 文例

```
select prod_name, store_name, date, dollars as sales_99,
    (select dec(avg(dollars),7,2)
    from store natural join sales
        natural join product
        natural join period
    where year = 1998
        and month = 'DEC'
        and store_name = 'San Jose Roasting Company'
        and prod_name like 'Lotta%') as avg_98
from store natural join sales
        natural join product
        natural join period
where prod_name like 'Lotta%'
    and store_name = 'San Jose Roasting Company'
    and year = 1999
    and month = 'DEC'
    and dollars <
        (select avg(dollars)
        from store natural join sales
            natural join product
            natural join period
        where year = 1998
            and month = 'DEC'
            and store_name = 'San Jose Roasting Company'
            and prod_name like 'Lotta%');
```

実行結果

Prod_Name	Store_Name	Date	Sales_99	Avg_98
Lotta Latte	San Jose Roasting Company	1999-12-09	153.00	154.72
Lotta Latte	San Jose Roasting Company	1999-12-28	144.50	154.72

検索項目リスト中のサブクエリによる比較

メインクエリの検索項目リストにサブクエリが使用できるのは、そのサブクエリが 1 行だけを返すか、何も返さない場合だけです。このようなサブクエリをスカラ サブクエリと呼び、メインクエリが返す複数の値をサブクエリが返す 1 つの値と比較する、スプレッドシート型の分析に使用します。

例題について

このクエリは、San Jose Roasting Company が 1999 年に販売した Lotta Latte の売上で、1998 年の日別平均売上を下回ったものを返します。**Avg_98** には、1998 年の日別平均売上を示す値がリザルト セットの行数に関わらず繰り返し表示されます。

メインクエリには、同一のサブクエリが次の 2 回に使用されています。

- 検索項目リストに、列の定義として
- WHERE 句の条件中に、不等号演算子 (<) の被演算子として

このクエリは、以下の順で処理されます。

1. メインクエリの WHERE 句にある、検索条件を定義する 2 番目のサブクエリを実行する。
2. 2 番目のサブクエリから得られた値を、メインクエリの WHERE 句に挿入する。
3. 検索項目リストのサブクエリを実行する。
4. メインクエリを実行する。

使用上の注意

DEC スカラ関数は、リザルト セットの **Avg_98** 列に返される平均売上の値を切り捨てます。

```
dec(avg(dollars),7,2)
```

相関サブクエリの使用

例題

San Jose における 1998 年 1 月の製品別売上を、同年同都市の年間売上と比較すると？

SQL 文例

```
select p1.prod_name, sum(s1.dollars) as jan_98_sales,

    (select sum(s2.dollars)
    from store r2 natural join sales s2
        natural join product p2 natural join period d2
        where p1.prod_name = p2.prod_name
            and d1.year = d2.year
            and r1.city = r2.city) as total_98_sales

from store r1 natural join sales s1
    natural join product p1
    natural join period d1
where year = 1998 and month = 'JAN'
    and city like 'San J%'
group by p1.prod_name, d1.year, r1.city
order by p1.prod_name;
```

実行結果

Prod_Name	Jan_98_Sales	Total_98_Sales
Aroma Roma	1653.00	21697.50
Aroma Sheffield Steel Teapot	120.00	1122.00
Aroma Sounds Cassette	58.50	866.00
Aroma baseball cap	7.95	2960.15
Aroma t-shirt	470.85	4470.50
Assam Gold Blend	652.00	11375.00

(1 / 2)

実行結果

Prod_Name	Jan_98_Sales	Total_98_Sales
Assam Grade A	352.00	5429.00
Breakfast Blend	608.25	6394.75
Cafe Au Lait	1936.50	24050.50
Colombiano	2148.00	22528.50
Darjeeling Number 1	867.50	8590.00
Darjeeling Special	1355.00	17787.50
Demitasse Ms	2163.00	35523.50
Earl Grey	540.50	6608.50
English Breakfast	393.00	5365.50
Espresso Machine Italiano	899.55	4397.80
Expresso XO	2935.50	27362.00
French Press, 2-Cup	104.65	1196.00
French Press, 4-Cup	19.95	1109.20
Gold Tips	440.00	5381.50
Irish Breakfast	703.25	7455.50
La Antigua	2643.25	22244.50
Lotta Latte	3195.00	31200.00
NA Lite	1319.00	27457.00
...		

(2 / 2)

検索項目リスト中の相関サブクエリ

検索項目リスト中のサブクエリは、そのサブクエリが 1 行を返すか、何も返さないものにかぎりますが、メインクエリが返す結果を参照しながら何度も実行することができます。つまり、FROM 句の相関サブクエリと同じ結果になります。

相関クエリは、メインクエリが抽出した行の特定の値を相互参照するという点で、メインクエリと密接に関連しています。たとえば、メインクエリの **Month** 列の値を参照する相関サブクエリは、**Month** 列の値が変化するたびに新しい値を返します。このような相互参照には、FROM 句で割り付けたテーブル相関名を使います。

例題について

このクエリは、4-9 ページのクエリと同じビジネス上の質問を表したものですが、FROM 句ではなく検索項目リストにサブクエリを使用しています。San Jose における 1998 年 1 月の製品別売上を、同年の年間売上と比較しています。

サブクエリが 1 つの固定値ではなく、複数の値を返すようにするには、3 つの相互参照によってサブクエリとメインクエリとを関連づけます。

```
p1.prod_name = p2.prod_name
d1.year = d2.year
r1.city = r2.city
```

p2、d2、r2 という相関名をサブクエリの FROM 句に定義することで、参照先が明確になります。それぞれの相関条件で、メインクエリが処理している行の特定の製品、年度、都市が参照されます。このような相互参照を、アウターリファレンスと呼びます。

使用上の注意

メインクエリの検索項目リストに集約関数を使用する場合は、GROUP BY 句が必要になります。サブクエリの相関条件で参照される列名はメインクエリの GROUP BY 句で指定する必要があるので、次の列を **Prod_Name** 列とともに GROUP BY 句で指定する必要があります。

```
d1.year, r1.city
```

相関名はデータベース識別子として使用するため、文字で始まり、128 文字以内でなければなりません。先頭文字の後には、文字、数値、下線を組み合わせて使用することができます。キーワードは、データベース識別子としては使用できません。

相互参照

例題

1998 年と 1999 年の第 1 四半期の、San Jose における Lotta Latte の月間売上は？

SQL 文例

```
select q.prod_name, e.month, sum(dollars) as sales_99,
    (select sum(dollars)
    from store t natural join sales s
        natural join product p
        natural join period d
    where d.month = e.month
        and d.year = e.year-1
        and p.prod_name = q.prod_name
        and t.city = u.city) as sales_98
from store u natural join product q
    natural join period e natural join sales l
where qtr = 'Q1_99'
    and prod_name like 'Lotta Latte%'
    and city like 'San J%'
group by q.prod_name, e.month, e.year, u.city;
```

実行結果

Prod_Name	Month	Sales_99	Sales_98
Lotta Latte	JAN	1611.00	3195.00
Lotta Latte	FEB	3162.50	4239.50
Lotta Latte	MAR	2561.50	2980.50

式による相互参照

サブクエリ中の相互参照は、該当する列名とは限りません。式を使った相互参照も可能です。たとえば、次の式は有効な相互参照になります。

```
period.year-1 (previous year)
period.quarter-1 (previous quarter)
```

このように汎用化した相互参照は、クライアント ツール向けのアプリケーション設計を簡略化します。

例題について

このクエリは、1999 年と 1998 年の最初の 3 ヵ月における San Jose での Lotta Latte の月間売上を返します。この例題は、月が同じでも年度が異なるデータを抽出することに注目してください。

メインクエリの FROM 句では、ジョインする全テーブルに相関名を割り付けます。

```
from store u natural join product q
    natural join period e natural join sales l
```

WHERE 句に指定した以下の条件に基づき、サブクエリの実行結果とメインクエリの実行結果を関連づけます。

```
d.month = e.month
d.year = e.year-1
p.prod_name = q.prod_name
t.city = u.city
```

メインクエリが行を抽出するたびに、親クエリの各列の値が変わります。相関条件は、この変化をサブクエリに伝達する役割を果たします。前年を year-1 として相互参照することで、定数値 (1998) が除去されてサブクエリが汎用化されます。

ほかの期間について報告するようにクエリを変更する場合は、メインクエリの year の制約だけを変更すれば良いのです。

使用上の注意

相互参照にはできるだけ式を使用し、相関サブクエリを汎用化してユーザの入力を少なくしてください。クエリ汎用化の詳細は、第 6 章「マクロ、ビュー、テンポラリ テーブル」を参照してください。

四半期と年間の比率の計算

例題

1998 年第 1 四半期に San Jose で販売した 1 ポンドのパッケージ入り製品の月間売上合計は？四半期合計と年間合計に対する各月の売上の比率は？

SQL 文例

```
select pj.prod_name, dj.month, sum(dollars) as mon_sales_98,
    dec(100 * sum(dollars)/
        (select sum(si.dollars)
        from store ri natural join sales si
            natural join product pi
            natural join period di
        where di.qtr = dj.qtr
            and di.year = dj.year
            and pi.prod_name = pj.prod_name
            and pi.pkg_type = pj.pkg_type
            and ri.city = rj.city), 7, 2) as pct_qtr1,
    dec(100 * sum(dollars)/
        (select sum(si.dollars)
        from store ri natural join sales si
            natural join product pi
            natural join period di
        where di.year = dj.year
            and pi.prod_name = pj.prod_name
            and pi.pkg_type = pj.pkg_type
            and ri.city = rj.city), 7, 2) as pct_yr
from store rj natural join sales sj
    natural join product pj
    natural join period dj
where rj.city = 'San Jose'
    and dj.year = 1998
    and dj.qtr = 'Q1_98'
    and pkg_type = 'One-pound bag'
group by pj.prod_name, dj.month, dj.qtr, dj.year,
pj.pkg_type,
    rj.city
order by pj.prod_name, pct_qtr1 desc;
```

実行結果

Prod_Name	Month	Mon_Sales_98	Pct_Qtr1	Pct_Yr
Aroma Roma	FEB	688.75	39.91	8.73
Aroma Roma	JAN	594.50	34.45	7.54
Aroma Roma	MAR	442.25	25.63	5.60
Cafe Au Lait	MAR	742.00	40.61	10.27
Cafe Au Lait	JAN	600.50	32.86	8.31
Cafe Au Lait	FEB	484.50	26.51	6.71
...				

検索項目リストのサブクエリによる演算

四半期や年間などの期間に対する月間の比率は、検索項目リストのサブクエリで算出することができます。月間の売上はメインクエリが抽出し、四半期と年間の売上は 2 つのサブクエリで抽出します。月間の比率を算出するには、四半期に対する月間の売上比率と年間に対する月間の売上比率を求める単純な演算が必要です。

例題について

このクエリでは、1998 年第 1 四半期に San Jose で販売されたコーヒー製品の中から選択された製品について、四半期に対する月間の売上比率と年間に対する月間の売上比率を算出します。比率の算出後、このクエリは、リザルト テーブルを製品ごとに四半期比率の降順に並べ替えます。

使用上の注意

この検索項目リストのサブクエリも、前の例題と同様に相互参照を明示し、サブクエリの実行結果とメインクエリが抽出した行を関連づけなければなりません。

通常このような比較クエリは、FROM 句のサブクエリとして表した方が高速であり、概念的にも理解しやすいでしょう。ただし、相関サブクエリの方が使いやすく、クエリの性能も問題ない場合は、書き直しは不要です。どちらの方法も同じ機能を持ち、同じ結果を返します。

WHERE 句のサブクエリ

例題

Chicago 地区の店舗で販売した Lotta Latte について、1999 年 6 月の売上が 1998 年 6 月の日別平均売上を下回った日は？

SQL 文例

```
select prod_name, district, date, dollars as sales_99
from market natural join store
    natural join sales
    natural join product
    natural join period
where prod_name like 'Lotta%'
    and district like 'Chic%'
    and year = 1999
    and month = 'JUN'
    and dollars <
        (select avg(dollars)
            from market natural join store
                natural join sales
                natural join product
                natural join period
            where prod_name like 'Lotta%'
                and district like 'Chic%'
                and year = 1998
                and month = 'JUN');
```

実行結果

Prod_Name	District	Date	Sales_99
Lotta Latte	Chicago	1999-06-08	76.50
Lotta Latte	Chicago	1999-06-11	59.50
Lotta Latte	Chicago	1999-06-17	42.50
Lotta Latte	Chicago	1999-06-18	76.50
Lotta Latte	Chicago	1999-06-30	110.50

WHERE 句のサブクエリによる比較

ここまでは、サブクエリを検索項目リストで使用するか、FROM 句で使用するかによって、機能が同じでも構文が異なることを中心に説明してきました。サブクエリは、WHERE 句の検索条件またはプレディケートとして使用することもできます。これにより、メインクエリの実行段階で複雑な制約を適用することができます。たとえば、WHERE 句の単純な検索条件として集約関数を使用することはできませんが、サブクエリに組み込めば WHERE 句でも使用できます。

例題について

このクエリは、Chicago 地区の店舗が 1999 年に販売した Lotta Latte の売上で、1998 年の日別平均売上を下回ったものを返します。

この例題のサブクエリは、1 つの値を返すスカラ サブクエリです。サブクエリで算出した 1998 年のシカゴの日別平均売上は、メインクエリが返すすべての行に対する制約として使用されます。リザルト セットには、1999 年の平均値を下回る 1998 年の値だけが表示されます。平均値そのものは、検索項目リストか FROM 句にサブクエリを移動しないかぎり、表示されません。

使用上の注意

クエリの論理的な処理順により、WHERE 句の制約が適用されるのは、FROM 句に指定したテーブルがジョインされた直後で、AVG や SUM などの集約関数や RISQL 表示関数の演算を行う前になります。このため、これらの関数を WHERE 句の単純な検索条件に使用することはできません。

ALL 比較プレディケートの使用

例題

Conneticut 州の Hartford で、2000 年 1 月に最高の日別売上を記録した製品は？

SQL 文例

```
select prod_name, date, dollars
from store natural join sales
    natural join product
    natural join period
where year = 2000
    and city = 'Hartford'
    and month = 'JAN'
    and dollars >= all
        (select dollars
        from store natural join sales
            natural join product
            natural join period
        where year = 2000
            and city = 'Hartford'
            and month = 'JAN');
```

実行結果

Prod_Name	Date	Dollars
NA Lite	2000-01-24	414.00

サブクエリ中の比較プレディケート

ALL、ANY、SOME、および EXISTS の各プレディケートは、サブクエリが抽出した複数の値に対する条件を表すために使用します。比較プレディケートは、2 つの値の論理的関係を表しており、比較結果は、指定した行に対して真、偽、または不定のいずれかになります。ANY プレディケートと SOME プレディケートはシノニムです。

プレディケート	真と判定される場合	戻り値がない場合
ALL	サブクエリが返したすべての値について、比較条件が真の場合	真と判定
ANY、SOME	サブクエリが返した値の１つ以上について、比較条件が真の場合	偽と判定
EXISTS	サブクエリが 1 行以上を返した場合	偽と判定

各プレディケートの詳細は、『SQL Reference Guide』を参照してください。

例題について

このクエリは、Hartford で 2000 年 1 月に最高の日別売上を記録した製品の名称と、その売上を記録した日付を返します。>= 演算子を <= 演算子に置き換えると、最低の日別売上を返すように変更することができます。

使用上の注意

この WHEN 句で RANK 関数を使用しても簡潔に表現できます。

```
select prod_name, date, dollars
from sales natural join period
    natural join product
    natural join store
where year = 2000
    and month = 'JAN'
    and city = 'Hartford'
when rank(dollars) = 1;
```

RANK クエリは、等値の行がある場合は複数の行を第 1 位として返しますが、WHERE 句のサブクエリは 1 行を返すか、何も返さないかのどちらかになります。

EXISTS プレディケートの使用

例題

2000 年 3 月中に 1 つ以上の注文を決済した仕入先は？

SQL 文例

```
select distinct name as supplier_name
from supplier
where exists
            (select * from orders
            where supplier.supkey = orders.supkey
            and extract(year from close_date) = 2000
            and extract(month from close_date) = 03);
```

実行結果

Supplier Name
Aroma East Mfg.
Aroma West Mfg.
Crashing By Design
Espresso Express
Leaves of London
Tea Makers, Inc.
Western Emporium

EXISTS プレディケート

EXISTS プレディケートはサブクエリの結果に対して動作し、「真」か「偽」のいずれかを判定します。「真」と判定された場合は、メインクエリからリザルト セットが返ります。「偽」と判定されると、メインクエリからは何も返されません。

例題について

このクエリは、2000 年 3 月中に Aroma Coffee Company の注文を 1 つ以上決済した仕入先の名前を返します。

サブクエリには、そのような仕入先が存在するかどうかを判定する 3 つの条件が指定されています。最初の条件は、**Supkey** 列を基準として **Supplier** テーブルと **Orders** テーブルをジョインします。2 番目と 3 番目の条件は EXTRACT 関数で表され、**Order** テーブルの **Close_Date** 列に該当する日付をチェックします。この関数の詳細な使用例については、3-44 ページ「EXTRACT 関数の使用」を参照してください。

使用上の注意

Supplier テーブル、**Orders** テーブル、**Period** テーブルをジョインして、同じ問い合わせを表すこともできます。

```
select distinct name as supplier_name
from supplier s, orders o, period p
where s.supkey = o.supkey
    and o.close_date = p.date
    and year = 2000
    and month = 'MAR';
```

このクエリでは、**Perkey** 列ではなく、**Close_Date** 列と **Date** 列を基準にして **Orders** テーブルと **Period** テーブルをジョインする必要があります。**Perkey** 列は、注文を入力した日付を示すものであるため、3 月より前になってしまう場合があるからです。たとえば、2 月の最後の週に入力した注文でも、品物の受領と注文の決済は 3 月の第 1 週かもしれません。

この **Oreders** テーブルと **Period** テーブルのジョインは、プライマリ キーや外部キーの関係を持たない列を基準としたジョインの例です。ジョインが可能なのは、**Close_Date** 列と **Date** 列が比較可能なデータ型だからです。

EXISTS と反対のプレディケートは、NOT EXISTS です。

```
...
where not exists (select...)
```

NOT EXISTS プレディケートの詳細は、『SQL Reference Guide』を参照してください。

SOME プレディケートと ANY プレディケートの使用

例題

注文金額が 10,000 ドルを超える注文をした仕入先は？その仕入先が 2000 年 3 月に決済した注文金額は？

SQL 文例

```
select name as supplier_name, price
from supplier natural join orders
    where extract(year from close_date) = 2000
        and extract(month from close_date) = 03
        and supplier_name = some
            (select name from supplier
                natural join orders
                where price > 10000)
order by supplier_name;
```

結果

Supplier_Name	Price
Aroma West Mfg.	4425.00
Espresso Express	30250.00
Espresso Express	25100.00
Espresso Express	26400.00
Espresso Express	22700.00
Western Emporium	10234.50

SOME プレディケートと ANY プレディケート

SOME と ANY の比較プレディケートは、サブクエリが返した値の中に、指定した条件を満たす値があれば「真」と判定します。SOME と ANY はシノニムなので、相互に入れ替えて使用することができます。

SOME と ANY は、インナー クエリに指定された条件はすべて満たしていても、アウター クエリの条件には完全に一致しない行をリザルト セットに残す場合に使用します。たとえば、アウター クエリの条件が、注文金額に関わらず特定の月に注文品を出荷した仕入先で、インナー クエリの条件が、月に関わらず特定の金額を超える注文品を出荷した仕入先という場合です。

例題について

このクエリは、SOME プレディケートを使って仕入先と注文金額を返します。

- サブクエリからは、10,000 ドルを超える注文を 1 件以上納入した仕入先のリストが返ります。
- メインクエリは、2000 年 3 月に決済された注文の記録と仕入先リストとを照合し、注文金額に関わらず、同月に注文を決済した仕入先を抽出します。

リザルト セットの最初の行には、Aroma West Mfg. が 2000 年 3 月に 4,425.00 ドルの注文品を納入したことが示されています。これは、ほかの時期に Aroma West Mfg. が納入した注文品の中に、10,000 ドルを超えるものが 1 件以上あったことを示します。

使用上の注意

このクエリでは、4-28 ページで説明した EXISTS プレディケートの例と同じ方法で EXTRACT 関数が使用されています。

各プレディケートの詳細は、『SQL Reference Guide』を参照してください。

まとめ

この章では、データを比較するクエリの作成方法と、読みやすいフォーマットで結果を表示する方法を説明しました。以下のような方法があります。

- CASE 式
- FROM 句のサブクエリ
- 相関サブクエリを含む、検索項目リストのサブクエリ
- WHERE 句のサブクエリ

この章の最後では、サブクエリの結果に対する条件として使用できる ALL、ANY、SOME、および EXISTS の比較プレディケートについて、例を示して説明しました。

その他、四半期や年間の値に対する比率をパーセントで算出する演算を比較クエリに組み込む複雑な例題も紹介しました。

重要：通常は、サブクエリを使用するよりも、CASE 式を使用した方が比較クエリの性能は良くなります。サブクエリが必要な場合は、検索項目リストでなく FROM 句に使用した方が効率的です。

第 5 章

ジョインとユニオン

この章について

この章では、複数テーブルのデータを結合する 2 種類の方法を説明します。

- テーブルのジョイン
- UNION、EXCEPT、および INTERSECT の各演算子を使用する方法

この章の前半では、インナー ジョインとアウター ジョインの例を示します。

後半では、UNION、EXCEPT、および INTERSECT の各演算子を使って複数テーブルのデータを結合する方法を説明します。1 つのクエリ式から中間結果を抽出し、別のクエリ式のリザルト セットと結合する演算子です。

2 テーブルのジョイン

State テーブル

City	State
Jacksonville	FL
Miami	FL
Nashville	TN

Region テーブル

City	Area
Jacksonville	South
Miami	South
New Orleans	South

SQL 文例

```
select * from state, region;
```

直積 (ジョイン プレディケートの指定なし)

City	State	City	Area
Jacksonville	FL	Jacksonville	South
Jacksonville	FL	Miami	South
Jacksonville	FL	New Orleans	South
Miami	FL	Jacksonville	South
Miami	FL	Miami	South
Miami	FL	New Orleans	South
Nashville	TN	Jacksonville	South
Nashville	TN	Miami	South
Nashville	TN	New Orleans	South

SQL 文例

```
select * from state, region
where state.city = region.city;
```

直積のサブセット (ジョイン プレディケートを指定)

State:City	State:State	Region:City	Region:Area
Jacksonville	FL	Jacksonville	South
Miami	FL	Miami	South

インナー ジョイン

ほとんどのクエリは、複数のテーブルから情報を取り出してジョインします。2 つのテーブルは、比較可能なデータ型の列を基準としてジョインすることができます。ジョインは、プライマリ キーや外部キーの関係に依存しません。

直積

複数のテーブルをクエリの FROM 句で参照すると、データベース サーバはテーブルをジョインします。FROM 句にも WHERE 句にもジョイン プレディケートを指定しない場合、m * n 行 で構成される直積が算出されます。m は第 1 テーブルの行数、n は第 2 テーブルの行数です。直積は、第 1 テーブルの行と第 2 テーブルの行を連結して作られるすべての可能な組み合わせの集合です。

重要： **rbw.config** ファイルの OPTION CROSS_JOIN パラメータが OFF (デフォルト) に設定されていると、クロス ジョイン クエリ (直積) は実行されません。

直積のサブセット

比較可能なデータ型の列に基づいてテーブルを明示的にジョインすると、直積のサブセットが算出されます。このサブセットには、ジョインする列内の値が一致する行だけが含まれています。クエリの実行中、サブセットは、ほかのテーブルまたはほかのクエリ式の結果とジョインできる派生テーブルとして機能します。

例題について

State テーブルと **Region** テーブルには、どちらも **City** 列が含まれています。City 列は、ジョインする列として WHERE 句に指定されています。このため、直積の中で、**City** キーが一致する行だけが実行結果に表示されます。このクエリでは、2 テーブルの直積は 9 行であるのに対して、リザルト テーブルには 2 行だけが表示されています。

ジョインする列は、5-7 ページに示すように FROM 句に指定することもできます。

重要： 次の 3 つのクエリで使用するテーブルは、**Aroma** データベースのテーブルではありません。この章で後述する例題では Aroma テーブルが使用されています。

各種のテーブル ジョイン

例題

1998 年と 1999 年のクリスマス特別セールの実施期間は？各年のこのセール期間中に販売した製品の総売上と、日別平均売上は？

SQL 文例 1

```
select promo_desc, year, sum(dollars) as sales,
    datediff(day, end_date, start_date)+1 as days_on_promo,
    string(sales/days_on_promo, 7, 2) as per_day
from period natural join sales
    natural join promotion
where promo_desc like 'Christmas%'
    and year in (1998, 1999)
group by promo_desc, year, days_on_promo;
```

SQL 文例 2

```
select promo_desc, year, sum(dollars) as sales,
    datediff(day, end_date, start_date)+1 as days_on_promo,
    string(sales/days_on_promo, 7, 2) as per_day
from period join sales on period.perkey = sales.perkey
    join promotion on promotion.promokey = sales.promokey
where promo_desc like 'Christmas%'
    and year in (1998, 1999)
group by promo_desc, year, days_on_promo;
```

SQL 文例 3

```
select promo_desc, year, sum(dollars) as sales,
    datediff(day, end_date, start_date)+1 as days_on_promo,
    string(sales/days_on_promo, 7, 2) as per_day
from period join sales using(perkey)
    join promotion using(promokey)
where promo_desc like 'Christmas%'
    and year in (1998, 1999)
group by promo_desc, year, days_on_promo;
```

実行結果 (検索方法は同一の結果となる)

Promo_Desc	Year	Sales	Days_on_Promo	Per_Day
Christmas special	1999	1230.00	31	39.67
Christmas special	1998	690.00	31	22.25

FROM 句のジョイン

FROM 句でテーブルを明示的にジョインする方法は 3 種類あります。

- ナチュラル ジョイン
- 指定した列に基づくジョイン (USING 構文)
- プレディケートに基づくジョイン (ON 構文)

例題について

このクエリは、同名の列を基準にして **Promotion**、**Period**、**Sales** という 3 つのテーブルをジョインします。クエリ 1 のように、NATURAL JOIN 構文によって簡略化することができます。クエリ 2 とクエリ 3 は、FROM 句でインナー ジョインを指定する方法です。3 つのクエリからは、同一のリザルト セットが返されます。ただし、ON を指定した場合と USING を指定した場合は、ジョインする列がそのまま中間リザルト セットに残されますが、NATURAL JOIN を指定すると、対応する 2 列が 1 列にジョインされます。

このクエリでは、以下のスカラ関数が使用されています。

- DATEDIFF 関数は、クリスマス セールの実施期間を算出します。詳細は、3-42 ページで説明しています。
- STRING 関数は、**Per_Day** 列の値を小数点第 2 位の精度に揃えるために使用します。この関数を使用しないと、次のような式から long 型の数値が返されます。

```
sales/promo_days
```

使用上の注意

ナチュラル ジョインは、同名の列があれば必ずジョインしてしまうため、注意が必要です。列名が同じものがあると、ジョインの対象でない列のテーブルがジョインされてしまうことがあります。

Aroma データベースの販売スキーマでは、同名の列に基づいてプライマリ キーと外部キーの関係が定義されているため、**Sales** テーブルとそのディメンジョンが関わるクエリにはナチュラル ジョインが正しく機能します。

プライマリ キー列や外部キー列を基準としないジョインの例は、3-43 ページ「経過日数の算出 :DATEDIFF」を参照してください。ジョインの構文の詳細は、『SQL Reference Guide』を参照してください。

システム テーブルのジョイン

例題

Aroma **Sales** テーブルのセグメント名と物理格納ユニット (PSU) 名は？

SQL 文例

```
select segname as storage, location as psu_location,
    tname as table_name
from rbw_storage join rbw_segments on
    rbw_storage.segname = rbw_segments.name
where table_name = 'SALES'
order by psu_location;
```

実行結果

Storage	PSU_Location	Table_Name
DEFAULT_SEGMENT_23	dfltseg23_psu1	SALES
DAILY_DATA1	sales_psu1	SALES
DAILY_DATA1	sales_psu2	SALES
DAILY_DATA2	sales_psu3	SALES
DAILY_DATA2	sales_psu4	SALES

システム テーブルのジョイン

データベース管理者は、テーブルとインデックス、テーブルとセグメントなどのデータベース オブジェクト間の関係を把握している必要があります。IBM Red Brick Warehouse システム テーブルは、ほかのデータベース テーブルと同じようにジョインできるため、このような情報にも簡単にアクセスできます。

例題について

このクエリは、2 つのテーブルをジョインし、**Aroma** データベースの **Sales** テーブルについて、デフォルト セグメントとユーザ定義セグメントの名前、ならびに関連した PSU の名前を表示します。

使用上の注意

次の WHERE 句の条件では、SALES は大文字で指定する必要があります。

```
table_name = 'SALES'
```

そのようにしないと、一致する行は見つかりません。

システム テーブルやテーブルのセグメント化などの詳細は、『Administrator's Guide』を参照してください。

セルフ ジョイン

例題

Product テーブルにある製品で、製品名が同じでパッケージが異なるものは？

SQL 文例

```
select a.prod_name as products,
    a.pkg_type
from product a, product b
where a.prod_name = b.prod_name
    and a.pkg_type <> b.pkg_type
order by products, a.pkg_type;
```

実行結果

Product	Pkg_Type
Aroma Roma	No pkg
Aroma Roma	One-pound bag
Assam Gold Blend	No pkg
Assam Gold Blend	Qtr-pound bag
Assam Grade A	No pkg
Assam Grade A	Qtr-pound bag
Breakfast Blend	No pkg
Breakfast Blend	Qtr-pound bag
Cafe Au Lait	No pkg
Cafe Au Lait	One-pound bag
Colombiano	No pkg

(1 / 2)

Product	Pkg_Type
Colombiano	One-pound bag
Darjeeling Number 1	No pkg
Darjeeling Number 1	Qtr-pound bag
Darjeeling Special	No pkg
Darjeeling Special	Qtr-pound bag
Demitasse Ms	No pkg
Demitasse Ms	One-pound bag
Earl Grey	No pkg
Earl Grey	Qtr-pound bag
English Breakfast	No pkg
English Breakfast	Qtr-pound bag
Expresso XO	No pkg
Expresso XO	One-pound bag
Gold Tips	No pkg
Gold Tips	Qtr-pound bag
Irish Breakfast	No pkg
Irish Breakfast	Qtr-pound bag
...	

(2 / 2)

テーブルのセルフ ジョイン

クエリでジョインするテーブルは、別個のテーブルでなくてもかまいません。1 つのテーブルに複数の参照名を与えれば、それ自身とジョインすることができます。セルフ ジョインは、1 つのテーブルにおける列間の関係を分析するのに役立ちます。

例題について

このクエリは、**Product** テーブルを **Prod_Name** 列に基づき、それ自身とジョインしていますが、a と b というエリアスを使ってテーブル参照を区別しています。

```
from product a, product b
```

このセルフ ジョインは、**Product** テーブル a を **Product** テーブル b と照合し、製品名が同じでパッケージが異なる行を捜します。

```
where a.prod_name = b.prod_name
    and a.pkg_type <> b.pkg_type
```

リザルト セットには、同名の製品が 2 つと、各製品のパッケージが表示されます。

2 つのテーブルのアウター ジョイン

SQL 文例 (レフト アウター ジョイン)

```
select * from state left outer join region
    on state.city = region.city;
```

実行結果

State:City	State:State	Region:City	Region:Area
Jacksonville	FL	Jacksonville	South
Miami	FL	Miami	South
Nashville	TN	NULL	NULL

SQL 文例 (ライト アウター ジョイン)

```
select * from state right outer join region
    on state.city = region.city;
```

実行結果

State:City	State:State	Region:City	Region:Area
Jacksonville	FL	Jacksonville	South
Miami	FL	Miami	South
NULL	NULL	New Orleans	South

SQL 文例 (フル アウター ジョイン)

```
select * from state full outer join region
    on state.city = region.city;
```

実行結果

State:City	State:State	Region:City	Region:Area
Jacksonville	FL	Jacksonville	South
Miami	FL	Miami	South
Nashville	TN	NULL	NULL
NULL	NULL	New Orleans	South

重要： 上記の例題は、5-3 ページに示したテーブルを使用しています。

アウター ジョイン

通常のテーブルのジョインは、値が一致する行だけを見つける検索条件に基づいて行われます。このタイプのジョインは、インナー イクイジョンと呼ばれます。意思決定支援分析では、一致する行と一致しない行の両方を抽出するアウター ジョインや、不等号の関係などを表すノン イクイジョンも必要になります。

アウター ジョインからは、インナー ジョインが返すすべての行と、他方のテーブルの行と一致しないすべての行が返ります。左のテーブルの行を残すものをレフト アウター ジョイン、右のテーブルの行を残すものをライト アウター ジョイン、両方のテーブルの行を残すものをフル アウター ジョインと呼びます。FROM 句で最初に指定したテーブルをレフト テーブルと言い、2 番目をライト テーブルと言います。いずれのアウター ジョインも、一致しない行の空白列は NULL で表されます。

構文

前の例に示したように、2 つのテーブルの間のアウター ジョインは、FROM 句の中で OUTER JOIN キーワード、ON サブ句の順に指定します。

```
FROM table_1LEFT|RIGHT|FULL OUTER JOIN table_2
    ON table_1.column = table_2.column
```

このほかに、FROM 句にアウター ジョイン プレディケートを指定する方法の詳細は、『SQL Reference Guide』を参照してください。

例題について

- レフト アウター ジョインからは、**State** テーブルのすべての行と、**Region** テーブルの一致する行が返されます。**Region** テーブルだけにある行は表示されません。
- ライト アウター ジョインからは、**Region** テーブルのすべての行と、**State** テーブルの一致する行が返されます。**State** テーブルだけにある行は表示されません。
- フル アウター ジョインからは、各テーブル固有の行と、両テーブルに共通の行が返ります。

ファクトとファクトのジョイン

例題

注文番号 3619 ～ 3626 について、品目ごとの支払い金額、注文ごとの支払い金額、またはその両方の支払金額は？

SQL 文例

```
select coalesce(o.order_no, l.order_no) as order_num,
    order_type, o.price as full_cost,
    l.price as line_cost
from orders o left outer join line_items l
    on o.order_no = l.order_no
    join period on o.perkey = period.perkey
where o.order_no between 3619 and 3626
order by order_num;
```

実行結果

Order_Num	Order_Type	Full_Cost	Line_Cost
3619	Tea	4325.25	725.25
3619	Tea	4325.25	400.00
3619	Tea	4325.25	400.00
3619	Tea	4325.25	400.00
3619	Tea	4325.25	400.00
3619	Tea	4325.25	400.00
3619	Tea	4325.25	400.00
3619	Tea	4325.25	400.00
3619	Tea	4325.25	400.00
3619	Tea	4325.25	400.00

(1 / 2)

実行結果

Order_Num	Order_Type	Full_Cost	Line_Cost
3620	Tea	4325.25	NULL
3621	Spice	10234.50	10234.50
3622	Spice	10234.50	10234.50
3623	Hardware	4425.00	400.00
3623	Hardware	4425.00	400.00
3623	Hardware	4425.00	500.00
3623	Hardware	4425.00	450.00
3623	Hardware	4425.00	500.00
3623	Hardware	4425.00	275.00
3623	Hardware	4425.00	650.00
3623	Hardware	4425.00	1250.00
3624	Hardware	4425.00	400.00
3624	Hardware	4425.00	500.00
3624	Hardware	4425.00	450.00
3624	Hardware	4425.00	400.00
3624	Hardware	4425.00	500.00
3624	Hardware	4425.00	275.00
3624	Hardware	4425.00	650.00
3624	Hardware	4425.00	1250.00
3625	Clothing	3995.95	2500.00
3625	Clothing	3995.95	1495.95
3626	Hardware	16500.00	NULL

レフト アウター ジョイン

アウター ジョインは、1 つのテーブルだけでは得られない関連情報をファクト テーブルの値と比較する方法として、よく使用されます。

例題について

Orders テーブルと **Line_Items** テーブルには、関連したデータが格納されていますが、品目の詳細情報がデータベースにロードされるのは注文情報より後になる場合があります。注文単位と品目単位の金額を両方とも知りたい場合や、品目の金額がなければ注文金額だけでも見たいという場合は、アウター ジョインが必要になります。

このクエリは、注文単位の金額と品目単位の金額を返します。品目の金額がなければ、注文単位の金額が表示され、**Line_Price** 列と **Line_Orders** 列には、NULL が表示されます。この結果は、**Orders** テーブルをレフト テーブルとするレフト アウター ジョインによって得ることができます。

COALESCE 関数は、検索項目リストに指定した 2 つの列から、レポートの列見出しに対応する値を 1 つ作成します。

```
coalesce(o.order_no, l.order_no) as order_num
```

いずれかの列が NULL である場合は、NULL でない列の値を COALESCE 関数が返します。この関数を使用しないと、リザルト セットの注文番号列が重複します。

使用上の注意

標準 SQL でアウター ジョインの条件を表すその他の方法の詳細は、『SQL Reference Guide』を参照してください。

この例は、**Aroma** 仕入れスキーマのテーブルを使用しています。仕入れスキーマの詳細は、付録 A「Aroma データベース詳細」で説明しています。

ファクトとファクトのジョイン

例題

2000 年の第 12 週と第 13 週について、販売収入と注文支出を比較すると？

SQL 文例

```
select date, extract(week from date) as wk_no, prices, sales
from
        ((select d1.date, sum(price)
        from orders natural join period d1
        where d1.year = 2000 and d1.week in (12, 13)
        group by d1.date) as t1
    full outer join
        (select d2.date, sum(dollars)
        from sales natural join period d2
        where d2.year = 2000 and d2.week in (12, 13)
        group by d2.date) as t2
    on t1.date = t2.date) as t3(order_date, prices, date,
sales)
order by wk_no, date
break by wk_no summing prices, sales;
```

実行結果

Date	Wk_No	Prices	Sales
2000-03-12	12	NULL	9991.65
2000-03-13	12	31800.00	10162.75
2000-03-14	12	NULL	9514.55
2000-03-15	12	NULL	9074.10
2000-03-16	12	NULL	11009.55
2000-03-17	12	NULL	9177.90
2000-03-18	12	NULL	7412.65

(1 / 2)

Date	Wk_No	Prices	Sales
NULL	12	31800.00	66343.15
2000-03-19	13	NULL	8620.25
2000-03-20	13	27025.25	8417.95
2000-03-21	13	NULL	8230.05
2000-03-22	13	NULL	9870.20
2000-03-23	13	NULL	8757.50
2000-03-24	13	NULL	8394.25
2000-03-25	13	3995.95	10046.90
NULL	13	31021.20	62337.10
NULL	NULL	62821.20	128680.25

(2 / 2)

ORDER BY、BREAK BY によるフル アウター ジョイン

フル アウター ジョインからは、ジョインする列の値が一致しているかに関わらず、レフト テーブルとライト テーブルの行を含む結果が返されます。リザルト セットでは、値が一致しない列に NULL が格納されます。

例題について

Sales テーブルと **Line_Items** テーブルには、異なるファクトのセットが格納されていますが、これらのテーブルは **Product** と **Period** という 2 つのディメンジョン テーブルを共有しています。特定期間の注文と売上に関するレポートを作成するには、各ファクト テーブルを **Period** テーブルとインナー ジョインし、その結果をアウター ジョインします。これは、FROM 句でサブクエリを使用しないと実行できません。

最初のサブクエリは **t1** という名前のテーブルを生成し、2 番目のサブクエリは ***t2*** という名前のテーブルを生成します。テーブル ***t3*** は、**t1** と **t2** をフル アウター ジョインした結果です。テーブル **t3** は次の 4 列で構成されます。

```
t3(order_date, prices, date, sales)
```

メインクエリの検索項目リストは、**Prices**、**Date**、および **Sales** という 3 つの列を参照します。検索項目リストの 4 番目の列 (**Wk_No**) は、EXTRACT スカラ関数によって **Date** 列から抽出されます。

```
extract(week from date) as wk_no
```

ORDER BY 句と BREAK BY サブ句は、週別と日付別にデータをソートし、**Prices** 列と **Sales** 列の両方について、週単位の小計を表示します。リザルト セットの最終行には、総計が表示されます。

使用上の注意

FROM 句には同一のテーブル名を繰り返し使用できないため、このクエリではテーブル エリアスが必要になります。たとえば、あるジョイン指定では **Period** テーブルは **d1** として参照され、別のジョイン指定では **d2** として参照されます。

BREAK BY 句で参照する列は、ORDER BY 句にも指定しなければなりません。各プレディケートの詳細は、『SQL Reference Guide』を参照してください。

前述の例は、第 4 章「比較クエリ」で紹介した FROM 句のサブクエリと構造が似ています。

日付時間型を扱うスカラ関数を使用するクエリの例は、第 3 章「データの解析」を参照してください。

OR と UNION の違い

例題

「Medium (中規模)」と分類される Aroma 店舗における 1999 年第 52 週の総売上は ?
「Large (大規模)」と分類される店舗の同期間の総売上は ?

OR 条件を使った SQL 例文

```
select store_name as store, store_type as size, state,
   sum(dollars) as sales
from period t join sales s on t.perkey = s.perkey
   join store r on r.storekey = s.storekey
where (store_type = 'Medium' or store_type = 'Large')
   and year = 1999
   and week = 52
group by store, size, state
order by size, store;
```

UNION を使った SQL 例文

```
select store_name as store, store_type as size, state,
   sum(dollars) as sales
from period t join sales s on t.perkey = s.perkey
   join store r on r.storekey = s.storekey
where store_type = 'Medium'
   and year = 1999
   and week = 52
group by store, size, state
union
select store_name as store, store_type as size, state,
   sum(dollars)
from period t join sales s on t.perkey = s.perkey
   join store r on r.storekey = s.storekey
where store_type = 'Large'
   and year = 1999
   and week = 52
group by store, size, state
order by size, store;
```

実行結果 (2 つのクエリは同一の結果となる)

Store	Size	State	Sales
Beaches Brew	Large	CA	2908.80
Miami Espresso	Large	FL	4582.00
Olympic Coffee Company	Large	GA	3732.50
San Jose Roasting Company	Large	CA	3933.15
Beans of Boston	Medium	MA	3772.75
Cupertino Coffee Supply	Medium	CA	2893.00
Java Judy's	Medium	AZ	3011.25
Moulin Rouge Roasting	Medium	LA	3972.00
Texas Teahouse	Medium	TX	3382.75

リザルト セットの和 ：UNION

UNION、EXCEPT、および INTERSECT の各演算子は、2 つ以上のクエリ式の結果の和をとり、行と列で構成される 1 つの集合に統合します。各クエリ式をサーバが個別に評価して結果を結合し、最初の式で使用した列名または列エリアスを列見出しにします。ALL キーワードを指定した場合を除き、重複行は削除されます。

UNION、INTERSECT、EXCEPT

```
<query_expression> UNION | INTERSECT | EXCEPT [ALL]
<query_expression>
[ORDER BY <order_list>]
[SUPPRESS BY <suppress_list>];
```

<query_expression>　『SQL Reference Guide』で定義されているジョイン クエリ式または非ジョイン クエリ式

SUPPRESS BY 句と ORDER BY 句を使用する場合は、最初のクエリ式の検索項目リストで指定した列を参照していなければなりません。

例題について

この質問の答えを得るには、1 つの SELECT 文で OR 条件を指定するか、2 つのクエリ式を UNION 演算子で結合します。

この例題では OR 接続詞の方が簡単ですが、UNION を使った方がクエリ性能が向上する場合もあります。たとえば、クエリが 2 つの大きなファクト テーブル内のデータをアクセスする必要がある場合です。そのアウター ジョイン演算を 1 つのクエリで行う場合は、UNION 演算を使用して 2 つのクエリ式の結果を組み合わせるよりも多くの処理を必要とします。

ORDER BY 句は、列名ではなく、最初のクエリ式の検索項目リストで指定した列エリアスを参照しなければなりません。

```
order by size, store
```

使用上の注意

UNION、EXCEPT、および INTERSECT の各クエリは、対称形でなければなりません。つまり、UNION 演算子の両側にある検索項目リストの列数と列の順序の双方が一致しなければなりません。対応する列は、列名は違ってもかまいませんが、同一もしくは比較可能なデータ型でなければなりません。

UNION、EXCEPT、および INTERSECT の演算子は、1 つのステートメントで何度も使用できます。かっこで評価順を指定した場合を除き、演算は左から右に評価されます。

INTERSECT 演算

例題

2000 年のセール期間中に San Jose で販売された計り売り紅茶製品で、1999 年に New Orleans のセール期間中にも販売されたものは？その製品を対象としたセールの内容は？

SQL 文例

```
select prod_name as tea_name, promo_desc
from sales natural join class
    natural join product
    natural join store
    natural join period
    natural join promotion
where city = 'San Jose'
    and year = 2000
    and class_desc like 'Bulk tea%'
intersect
select prod_name, promo_desc
from sales natural join class
    natural join product
    natural join store
    natural join period
    natural join promotion
where city = 'New Orleans'
    and year = 1999
    and class_desc like 'Bulk tea%'
    and promo_desc not like 'No promo%'
order by promo_desc;
```

実行結果

Tea_Name	Promo_Desc
Irish Breakfast	Aroma catalog coupon
Special Tips	Aroma catalog coupon
Darjeeling Special	Store display
Darjeeling Special	Temporary price reduction
Gold Tips	Temporary price reduction

共通行の抽出 :INTERSECT

INTERSECT 演算子は、2 つ以上のクエリ式から返された結果の共通行を抽出します。

例題について

このクエリは、2000 年のセール期間中に San Jose で販売された計り売り紅茶製品をリストアップするクエリと、1999 年に New Orleans のセール期間中で販売された製品を表示するクエリの共通項目を抽出します。両方のリザルト セットに共通しない行は INTERSECT 演算子により削除されます。

使用上の注意

UNION、EXCEPT、および INTERSECT の結果は、最初のクエリ式で指定した列名または列エリアスを列見出しにします。この場合、**Tea_Name** という列エリアスは、最初のクエリ式だけに指定すれば良いのです。

サブクエリ内の INTERSECT 演算

例題

2000 年 3 月に仕入れた製品のうち、同月に Coffee Connection 店で販売されたものは？

同月における上記製品の仕入れ総額は？

North 地区における同製品の同月の総収入 (売上合計) は？

SQL 文例

```
select product, cost_of_orders, revenue_north
from (select prod_name
        from product natural join sales natural join period
            natural join store
        where year = 2000 and month = 'MAR'
            and store_name = 'Coffee Connection'
        intersect
        select prod_name
        from product natural join line_items natural join
period
        where year = 2000 and month = 'MAR') as p(product)
    natural join
        (select prod_name, sum(price)
        from product natural join line_items natural join
period
        where year = 2000 and month = 'MAR'
        group by prod_name) as c(product, cost_of_orders)
    natural join
        (select prod_name, sum(dollars)
        from product natural join sales natural join period
            natural join store natural join market
        where year = 2000 and month = 'MAR' and region =
'North'
        group by prod_name) as r(product, revenue_north)
order by product;
```

実行結果

Product	Cost_of_Orders	Revenue_North
Aroma Roma	7300.00	3190.00
Cafe Au Lait	7300.00	3975.50
Colombiano	7300.00	3907.50
Demitasse Ms	8500.00	6081.25
Expresso XO	7300.00	4218.50
La Antigua	7300.00	3510.50
Lotta Latte	7300.00	4273.00
NA Lite	7300.00	6480.00
Veracruzano	7300.00	4055.00
Xalapa Lapa	7300.00	6896.50

ファクト テーブル データの INTERSECT

UNION、EXCEPT、および INTERSECT の演算子は、同一もしくは比較可能なデータを含むテーブルのクエリに便利です。

例題について

このクエリは、FROM 句に 3 つのサブクエリを指定しています。最初のサブクエリの INTERSECT 演算子は、2000 年 3 月に注文した製品で、同月に Coffee Connection 店でも販売されたものを抽出します。これには、**Sales** テーブルと **Line_Items** テーブルをジョインする 2 つのクエリ式の間に INTERSECT 演算子を挿入します。

2 番目のサブクエリは、最初のサブクエリから抽出した製品について、2000 年 3 月における仕入れ総額を算出します。

3 番目のサブクエリは、North 地区における同製品の同月の売上合計を算出します。

使用上の注意

前述のクエリは、第 4 章「比較クエリ」で紹介した FROM 句のサブクエリと構造が似ています。メインクエリの検索項目リストは、サブクエリから得られたテーブルの列だけで構成されます。

EXCEPT 演算子

例題

Market テーブルの HQ cities (所轄本部の都市) として定義されていない、California 州の都市にある店舗の 1999 年の総売上は？

SQL 文例

```
select city, store_name, sum(dollars) as sales_99
from (select city
        from store
        where state = 'CA'
        except
        select hq_city
        from market
        where hq_state = 'CA')
        as except_cities(city)
    natural join store
    natural join sales
    natural join period
where year = 1999
group by city, store_name
order by sales_99 desc;
```

実行結果

City	Store_Name	Sales_99
Cupertino	Cupertino Coffee Supply	196439.75
Los Gatos	Roasters, Los Gatos	175048.75

EXCEPT : 2 つのリザルト セットからの例外の検出

EXCEPT 演算子は、2 つのクエリ式の結果から例外または差異を検出します。たとえば、EXCEPT で 2 つの店舗で販売した製品のリストを比較し、両方の店舗で販売した製品を除外して、最初のクエリ式で指定した店舗だけで販売した製品を残すことができます。

例題について

この例の EXCEPT 演算子は、**Store** テーブルの **City** 列にある California 州の都市で、**Market** テーブルの **Hq_City** 列にはない都市を選択します。

FROM 句のサブクエリは、**Sales**、**Store**、および **Period** の各テーブルとジョインできる派生テーブルを作成します。サブクエリから派生したテーブルには、相関名と列名が 1 つずつ与えられています。

```
except_cities(city)
```

この派生テーブルは、**City** 列に基づいて **Store** テーブルとナチュラル ジョインすることができます。

使用上の注意

EXCEPT 演算の結果をテストするには、サブクエリを独立したクエリとして実行してみます。

```
select city
from store
where state = 'CA'
except
select hq_city
from market
where hq_state = 'CA';

CITY
Cupertino
Los Gatos
```

ほかのサブクエリの例は、第 4 章「比較クエリ」を参照してください。

まとめ

この章では、以下の項目について説明しました。

- テーブルのジョイン方法
- UNION、INTERSECT、および EXCEPT の各演算子を使い、2 つのクエリ式の結果をジョインする方法

テーブルのジョイン

クエリの FROM 句に 2 つ以上のテーブルが指定されていると、サーバは各テーブルをジョインします。2 つのテーブルは、比較可能なデータ型の列に基づいて、インナー ジョインとアウター ジョインのどちらの方法でもジョインできます。ジョインは、FROM 句か WHERE 句を使用して指定します。

UNION、INTERSECT、および EXCEPT 演算子

```
<query_expression>
UNION | INTERSECT | EXCEPT [ALL]
<query_expression>
[ORDER BY <order_list>]
[SUPPRESS BY <suppress_list>];
```

第 6 章

マクロ、ビュー、テンポラリテーブル

この章について

この章では、RISQL マクロを使って SQL 文を簡略化する方法を説明します。

マクロは、複雑な式を略記したものです。マクロを使うと、簡潔で再利用可能な SQL 文を作成することができます。

この章では、データの抽出を簡略化するその他の方法として、ビューとテンポラリ テーブルの簡単な例も紹介します。

この章では、次の手順を例題を使って説明します。

- 複雑な、または使用頻度の高い式やクエリを、マクロを使って省略する
- ほかのマクロを組み込んだマクロを記述する
- パラメータを使用する汎用マクロを記述する
- ビューの作成およびビューへのクエリ
- テンポラリ テーブルの作成、登録、テンポラリ テーブルへのクエリ

基本的なマクロ

例題

1999 年の紅茶製品の売上合計は？

CREATE MACRO 文

```
create macro tea_products as
    (pt.classkey = 2 or pt.classkey = 5);
```

Example Query

```
select prod_name,
    case pt.classkey when 2 then 'Bulk Tea'
        when 5 then 'Pkg Tea' end as class,
    sum(dollars) as sales_99
from product pt join sales sa
        on pt.classkey = sa.classkey
        and pt.prodkey = sa.prodkey
    join period pd on pd.perkey = sa.perkey
where tea_products
    and year = 1999
group by prod_name, pt.classkey
order by sales_99 desc;
```

実行結果

Prod_Name	Class	Sales_99
Darjeeling Special	Bulk Tea	80610.50
Darjeeling Special	Pkg Tea	51266.00
Assam Gold Blend	Bulk Tea	42329.00
Darjeeling Number 1	Bulk Tea	34592.75
Irish Breakfast	Bulk Tea	27763.75

(1 / 2)

Prod_Name	Class	Sales_99
Assam Gold Blend	Pkg Tea	27192.50
English Breakfast	Bulk Tea	25848.00
Breakfast Blend	Bulk Tea	24594.00
Darjeeling Number 1	Pkg Tea	24232.00
Earl Grey	Bulk Tea	23269.50
Special Tips	Bulk Tea	22326.00
Assam Grade A	Bulk Tea	21964.00
Gold Tips	Bulk Tea	21584.50
Irish Breakfast	Pkg Tea	20084.00
English Breakfast	Pkg Tea	18955.00
Breakfast Blend	Pkg Tea	17031.50
Gold Tips	Pkg Tea	16783.25
Special Tips	Pkg Tea	16773.25
Assam Grade A	Pkg Tea	16724.00
Earl Grey	Pkg Tea	16108.00

(2 / 2)

基本的なマクロの使用

マクロは、複雑な式を略記したものです。たとえば、短くわかりやすい名前を数値コードに設定し、数値列ではなくマクロ名でコードを参照することができます。同様に、複数の条件にマクロを定義し、そのマクロ名によって条件全体をクエリ内で参照できます。この条件は、SELECT 文全体の場合、または SELECT 文中の特定の句である場合があります。

マクロ名は、文字で始まる 128 字以内の文字列です。データベース サーバは、大文字 / 小文字を区別しません。たとえば、share と SHARE は同一のものとして扱われます。RISQL キーワードは、マクロ名としては使用できません。

CREATE MACRO の構文

```
CREATE MACRO <macro_name> AS <definition>;
```

<macro_name>　SQL 文中でこのマクロ定義を呼び出すための固有名称です。

<definition>　SQL 文の全体または一部です。SQL 文全体の場合は、1 度に 1 つのステートメントしか定義できません。

同名の新しいマクロを定義する場合は、既存マクロを削除しなければなりません。

```
DROP MACRO <macro_name>;
```

CREATE MACRO コマンドと DROP MACRO コマンドには、オプションの引数もあります。詳細は、『SQL Reference Guide』を参照してください。

例題について

tea_products というマクロは、**Classkey** 値の 2 と 5 が、それぞれ計り売り紅茶製品とパッケージ詰め紅茶製品を示しているという認識に基づいています。**Classkey** 値は、**Class** テーブルではなく **Product** テーブルから問い合わせることで、クエリ内のジョインを簡略化しています。**Classkey** 値をわかりやすくテキストに変換するには、CASE 式を使用します。

このクエリは、マクロを使って、すべての紅茶製品の、1999 年の売上合計を算出します。データベース サーバはクエリを解析する際に、マクロを CREATE MACRO 文に定義された文字列に置き換えます。

クエリ例ではマクロ定義をかっこで囲む必要があります。かっこを付けることで、マクロ内で定義されている論理演算子が正しく評価されます。

埋め込みマクロ

例題

1999 年の紅茶製品の売上合計は？

CREATE MACRO 文

```
create macro case_tea as
    case pt.classkey when 2 then 'Bulk Tea'
        when 5 then 'Pkg Tea'
        end as class;
create macro tea_totals as
    select prod_name, case_tea, sum(dollars) as sales_99
    from product pt join sales sa
        on pt.classkey = sa.classkey and pt.prodkey =
sa.prodkey
        join period pd on pd.perkey = sa.perkey
    where tea_products
        and year = 1999
    group by prod_name, class
    order by sales_99 desc;
```

SQL 文例

```
tea_totals;
```

実行結果

Prod_Name	Class	Sales_99
Darjeeling Special	Bulk Tea	80610.50
Darjeeling Special	Pkg Tea	51266.00
Assam Gold Blend	Bulk Tea	42329.00
Darjeeling Number 1	Bulk Tea	34592.75
Irish Breakfast	Bulk Tea	27763.75

(1 / 2)

実行結果

Prod_Name	Class	Sales_99
Assam Gold Blend	Pkg Tea	27192.50
English Breakfast	Bulk Tea	25848.00
Breakfast Blend	Bulk Tea	24594.00
Darjeeling Number 1	Pkg Tea	24232.00
Earl Grey	Bulk Tea	23269.50
Special Tips	Bulk Tea	22326.00
Assam Grade A	Bulk Tea	21964.00
Gold Tips	Bulk Tea	21584.50
Irish Breakfast	Pkg Tea	20084.00
English Breakfast	Pkg Tea	18955.00
Breakfast Blend	Pkg Tea	17031.50
Gold Tips	Pkg Tea	16783.25
Special Tips	Pkg Tea	16773.25
Assam Grade A	Pkg Tea	16724.00
Earl Grey	Pkg Tea	16108.00

(2 / 2)

埋め込みマクロの使用

埋め込みマクロは、ほかのマクロの定義内で使用されるマクロです。

例題について

この例の CREATE MACRO 文は **case_tea** と **tea_totals** という 2 つのマクロを定義しています。

- 1 つめのマクロには、各 Classkey を有意なクラス タイプと置き換える CASE 式が含まれています。この CASE 式は、6-4 ページのクエリの FROM 句で定義されたものと同じです。
- 2 つめのマクロは、6-4 ページで定義された 2 つの埋め込みマクロ **case_tea** と **tea_products** が含まれた完全な SELECT 文です。

tea_totals マクロを実行するには、そのマクロ名を入力します。

```
tea_totals;
```

6-4 ページで紹介した例と同じリザルト セットが返されます。

使用上の注意

マクロは、何重にも下のレベルに埋め込むことができます。

マクロは既存のマクロを使用して定義できますが、マクロ定義の中で別のマクロを定義することはできません。

引数を使ったマクロ

例題

ある年の紅茶製品の年間売上合計は？

CREATE MACRO 文

```
create macro tea_sales(yr) as
    select year, prod_name, case_tea,
        sum(dollars) as us_sales
    from product pt join sales sa
        on pt.classkey = sa.classkey and pt.prodkey =
sa.prodkey
        join period pd on pd.perkey = sa.perkey
    where tea_products
        and year = yr
    group by year, prod_name, class
    order by us_sales desc;
```

SQL 文例

```
tea_sales(1998);
```

実行結果

Year	Prod_Name	Class	US_Sales
1998	Darjeeling Special	Bulk Tea	75582.00
1998	Darjeeling Special	Pkg Tea	51625.00
1998	Assam Gold Blend	Bulk Tea	43091.00
1998	Darjeeling Number 1	Bulk Tea	36442.00
1998	Assam Gold Blend	Pkg Tea	28328.00
1998	Irish Breakfast	Bulk Tea	27440.75
1998	English Breakfast	Bulk Tea	27071.00
1998	Darjeeling Number 1	Pkg Tea	25841.25
1998	Earl Grey	Bulk Tea	24721.00
1998	Breakfast Blend	Bulk Tea	24689.25
1998	Gold Tips	Bulk Tea	23181.25
1998	Special Tips	Bulk Tea	22712.25
1998	Assam Grade A	Bulk Tea	22418.00
1998	Irish Breakfast	Pkg Tea	21318.25
1998	Breakfast Blend	Pkg Tea	17606.25
1998	English Breakfast	Pkg Tea	17310.00
1998	Assam Grade A	Pkg Tea	16787.00
1998	Earl Grey	Pkg Tea	16416.00
1998	Special Tips	Pkg Tea	15883.75
1998	Gold Tips	Pkg Tea	15732.50

引数を使ったマクロの使用

マクロは、任意の数の引数を使って汎用化することができます。引数は、マクロを実行するたびに変更できます。たとえば、年を表す引数を使ってマクロを作成し、データベースに格納された任意の年の値を抽出することができます。同様に、ある製品の市場のパラメータを持つマクロは、どんな特定の市場のデータも検索できます。

CREATE MACRO 文

次のコマンドを使い、引数を使ったマクロを定義してください。

```
CREATE MACRO <macro_name>([<parameter> [, <parameter>] ・ ) AS
<definition>;
```

<macro_name> マクロ定義を参照するための固有名称です。

<parameter> 汎用マクロをカスタマイズする値です。マクロを使用するたびに変更することができます。

<definition> SQL 文の全体または一部です。SQL 文全体の場合は、1 度に 1 つのステートメントしか定義できません。

引数を使ったマクロを SELECT 文の中で呼び出す場合は、CREATE MACRO 文で定義した各引数の値も指定しなければなりません。

例題について

この CREATE MACRO 文は、年を表す引数 (yr) を含む SELECT 文を定義します。**tea_sales(1998)** というマクロを実行すると、データベース サーバは yr という引数が出てくるたびに **1998** に置き換えます。このクエリは、販売データがデータベースに格納されている任意の年 (**Aroma** データベースの場合は 1998、1999、2000 年のいずれか) に使用することができます。

複数の引数をとるマクロ

例題

ある年のある都市で、売上が最高だった製品は？

CREATE MACRO 文

```
create macro top_rank(yr, locn, nbr) as
select prod_name, city, year, sum(dollars) as sales,
    rank() over (order by sales desc) as ranking
from product pt join sales sa
    on pt.prodkey = sa.prodkey and pt.classkey = sa.classkey
    join period pd on pd.perkey = sa.perkey
    join store se on se.storekey = sa.storekey
where city = locn and year = yr
group by prod_name, city, year
when ranking <= nbr;
```

SQL 文例 1 と実行結果

```
top_rank(1998, 'Los Angeles', 5);
```

Prod_Name	City	Year	Sales	Ranking
Xalapa Lapa	Los Angeles	1998	14930.00	1
Demitasse Ms	Los Angeles	1998	14402.25	2
Ruby's Allspice	Los Angeles	1998	14339.00	3
Aroma Roma	Los Angeles	1998	14253.25	4
Expresso XO	Los Angeles	1998	13179.50	5

SQL 文例 2 と実行結果

```
top_rank(1999, 'San Jose', 1);
```

Prod_Name	City	Year	Sales	Ranking
Demitasse Ms	San Jose	1999	32887.75	1

SQL 文例 3 と実行結果

```
top_rank(2000, 'Hartford', 3);
```

Prod_Name	City	Year	Sales	Ranking
NA Lite	Hartford	2000	5061.00	1
Cafe Au Lait	Hartford	2000	4665.00	2
Xalapa Lapa	Hartford	2000	4610.00	3

複数の引数を使ったマクロ

マクロには、複数の引数を使用することができます。たとえば、年、地域、製品名を表す引数を使ったマクロを定義することができます。

例題について

この CREATE MACRO 文は、次の 3 つの引数 (yr、locn、nbr) を含む SELECT 文を定義します。

```
(yr, locn, nbr)
```

この引数は、それぞれ表示される年、都市 (場所)、何位までのデータを返すかを表します。

SQL 文 1 は、1998 年の Los Angeles における売上上位 5 製品を抽出します。

```
top_rank(1998, 'Los Ang%', 5);
```

このマクロを実行すると、データベース サーバが各引数をそれぞれ 1998、Los Angeles、および 5 に置き換えます。SQL 文 2 と SQL 文 3 は、ほかの年、場所、順位について結果を返します。

使用上の注意

日別、週間、月間、四半期、年間などの期間は、引数で表すと便利です。製品名、ブランド、商標、仕入先などにも応用できます。

このクエリで使用される RANK 関数は、SQL OLAP 関数の 1 つです。OLAP 関数の例は、第 3 章「データの解析」を参照してください。

San Jose には 2 つの Aroma 店舗がありますが、Hartford と Los Angeles には 1 つずつしかありません。SQL 文 2 の結果は、San Jose の 2 店舗の売上合計を返しています。

SQL 文 3 のリザルト セットでは、売上金額が非常に少なくなっています。**Aroma** データベースには、1998 年と 1999 年は第 1 ～ 4 四半期の売上が格納されていますが、2000 年については第 1 四半期の売上しかないためです。

比較

例題

1999 年第 1 四半期と 2000 年第 1 四半期について、San Jose における Lotta Latte の月間売上を売上金額と販売数量の両方で比較すると？

CREATE MACRO 文

```
create macro lotta_sales(facts, yr) as (
    select sum(facts)
        from store t natural join sales s
            natural join product p
            natural join period d
        where d.month = e.month
            and d.year = e.yr
            and p.prod_name = q.prod_name
            and t.city = u.city);
```

SQL 文例 1 と実行結果

```
select q.prod_name, e.month, sum(dollars) as sales_99,
    lotta_sales(dollars, year+1) as sales_00
from store u natural join product q natural join period e
    natural join sales l
where qtr = 'Q1_99'
    and prod_name like 'Lotta Latte%'
    and city like 'San J%'
group by q.prod_name, e.month, e.year, u.city;
```

Prod_Name	Month	Sales_99	Sales_00
Lotta Latte	JAN	1611.00	3475.00
Lotta Latte	FEB	3162.50	2409.50
Lotta Latte	MAR	2561.50	2831.50

SQL 文例 2 と実行結果

```
select q.prod_name, e.month, sum(dollars) as sales_99,
    lotta_sales(dollars, year+1) as sales_00,
    lotta_sales(quantity, year) as qty_99,
    lotta_sales(quantity, year+1) as qty_00
from store u natural join product q natural join period e
    natural join sales l
where qtr = 'Q1_99'
    and prod_name like 'Lotta Latte%'
    and city like 'San J%'
group by q.prod_name, e.month, e.year, u.city;
```

Prod_Name	Month	Sales_99	Sales_00	Qty_99	Qty_00
Lotta Latte	JAN	1611.00	3475.00	197	426
Lotta Latte	FEB	3162.50	2409.50	391	298
Lotta Latte	MAR	2561.50	2831.50	314	348

比較マクロの使用

クエリでは、基本的な命令ブロックを少しずつ変化させながら繰り返し使用することがよくあります。この変化する要素にマクロの引数を利用すると便利です。たとえば、今年の売上と昨年の売上を比較するクエリには、よく似た命令ブロックが含まれています。今年の売上を抽出するブロックと、昨年の売上を抽出するブロックです。year を表す引数を使うと、入力する命令の数を減らすことができます。

例題について

例 1 と例 2 では、1999 年第 1 四半期について、San Jose における Lotta Latte の月間売上をメインクエリが抽出し、2000 年の値をマクロ (サブクエリ) が抽出します。

次のマクロ サブクエリは、指定された年について、売上金額と販売数量という 2 種類のファクトのどちらかを抽出できます。

```
lotta_sales(facts, yr)
```

Sales テーブルには売上金額と販売数量に対する追加された列のみが含まれています。生産データベースにはさらに多くの種類のファクトが含まれています。

SQL 文 1 のマクロは、**Sales** テーブルの **Dollars** 列と 2000 年を参照します。

```
lotta_sales(dollars, year+1)
```

メインクエリ内の WHERE 句の制約によって 1999 年が参照されるため、次の式の評価結果は 2000 になります。

```
year+1
```

このマクロは、複雑なクエリ作成に非常に便利です。たとえば、SQL 文 2 でこのマクロは 3 回参照され、3 つの列をリザルト セットに返します。

使用上の注意

このマクロは、検索項目リストに定義された相関サブクエリとして紹介していますが、同等のサブクエリを FROM 句で定義した方が高速になります (第 4 章「比較クエリ」参照)。

比率の比較

例題

2000 年と 1999 年の第 1 四半期の、San Jose における Lotta Latte の月間売上は？各年で四半期に占める各月の割合 (%) は？

CREATE MACRO 文

```
create macro lotta_qtr_sales(facts, yr) as
    (select sum(facts)
    from store t natural join sales s
        natural join product p
        natural join period d
    where substr(d.qtr,1,2) = substr(e.qtr,1,2)
        and d.year = e.yr
        and p.prod_name = q.prod_name
        and t.city = u.city);
```

SQL 文例

```
select q.prod_name, e.month, sum(dollars) as sales_99,
    dec(100*sales_99/lotta_qtr_sales(dollars, year),7,2) as
        share_qtr_99,
    lotta_sales(dollars, year+1) as sales_00,
    dec(100*sales_00/lotta_qtr_sales(dollars, year+1),7,2)
as
        share_qtr_00
from store u natural join product q
    natural join period e
    natural join sales l
where qtr = 'Q1_99'
    and prod_name like 'Lotta Latte%'
    and city like 'San J%'
group by q.prod_name, e.month, e.qtr, e.year, u.city,
sales_00;
```

実行結果

Prod_Name	Month	Sales_99	share_qtr_99	Sales_00	Share_Qtr_00
Lotta Latte	JAN	1611.00	21.96	3475.00	39.86
Lotta Latte	FEB	3162.50	43.11	2409.50	27.64
Lotta Latte	MAR	2561.50	34.92	2831.50	32.48

比率比較マクロの使用

マクロは、演算も簡略化することができます。たとえば、ある製品の月間売上を抽出するマクロと、その製品の四半期または年間の売上を算出するマクロを作成すれば、四半期または年間の総売上に占める月間売上の割合を簡単に算出することができます。

比率は、単純な演算でパーセントとして表すことができます。たとえば、以下のように指定します。

```
100*(monthly_sales/quarterly_sales)
```

このようなマクロは、ほかの年にも応用することができます。

例題について

このクエリは、1999 年および 2000 年の第 1 四半期の San Jose における Lotta Latte の月間売上を抽出し、これらの年の各月の売上がその四半期の売上に占める割合 (%) を算出します。この SQL 文は 3 つのマクロを使用しないと、非常に長くわかりにくいクエリになります。

次のマクロは、このクエリにおいて、この章の前の例で示したときと同じように機能します。

```
lotta_sales(facts, yr)
```

次のマクロは、この場合に指定された年の四半期売上金額を計算する別のサブクエリです。

```
lotta_qtr_sales(facts, yr)
```

また、このマクロを使用して、四半期販売数量を計算することもできます。このマクロの結果はレポートには表示されませんが、比率計算のソース データとして使用されます。

SUBSTR 関数は、**Qtr** 列値が最初の 2 文字 (Q1) を基にするよう関連させるためにマクロで定義されます。この制約は、**Period** テーブルの **Qtr** 値は各年に特定なので必須です (たとえば、**Q1_99** と **Q1_00** など) 。SUBSTR 関数の詳細は、『SQL Reference Guide』を参照してください。

使用上の注意

GROUP BY 句には、**Sales_00** 列とともに、検索項目リストにあるかまたはサブクエリの相関条件で参照されるほかの非集約列も指定する必要があります。

変化率

例題

San Jose における 1999 年第 1 四半期と 2000 年第 1 四半期の Lotta Latte の月間売上を比較した場合

- 各月の売上は増加したか、減少したか ? 増減の比率は ?
- 四半期に対する月間の売上比率は増加したか、減少したか ? 増減の比率は ?

SQL 文例

```
select q.prod_name, e.month, sum(dollars) as sales_99,
    lotta_sales(dollars, year+1) as sales_00,
    dec(100*((sales_00 - sales_99)/sales_99),7,2) as
sales_chg,
    dec(100*
        ((sales_00/lotta_qtr_sales(dollars, year+1))
        -
        (sales_99/lotta_qtr_sales(dollars, year)))
        ,7,2)
        as share_chg
from store u natural join product q
    natural join period e natural join sales l
where e.year = 1999
    and e.qtr = 'Q1_99'
    and q.prod_name like 'Lotta Latte%'
    and u.city like 'San J%'
group by q.prod_name, e.month, e.qtr, e.year, u.city,
sales_00;
```

実行結果

Prod_Name	Month	Sales_99	Sales_00	Sales_Chg	Share_Chg
Lotta Latte	JAN	1611.00	3475.00	115.70	17.90
Lotta Latte	FEB	3162.50	2409.50	-23.81	-15.47
Lotta Latte	MAR	2561.50	2831.50	10.54	-2.43

変化率を算出するマクロ

マクロを使うと、SQL で問い合わせを簡単に記述できるため、売上や市場動向のビジネス分析に専念できます。

たとえば、ある製品の月間売上の 2 年間の変化は、次の演算式を使用すればパーセントで表すことができます。

```
100*((monthly_sales_00 - monthly_sales_99)/monthly_sales_99)
```

同様に、ある製品の売上の四半期における変化は次のように計算できます。

```
100*(monthly_sales_00/quarterly_sales_00)
-
(monthly_sales_99/quarterly_sales_99)
```

いずれも、比率の算出は難しくありませんが、マクロを使うとクエリの作成が簡略化されます。

例題について

前に定義した 2 つのマクロ **lotta_sales** と **lotta_qtr_sales** をこのクエリで使用して、Lotta Latte 製品についての 1999 ～ 2000 年の月間売上の変化と、月間売上が四半期売上に占める割合の変化をパーセントで算出します。

ビュー

例題

Assam Gold Blend 紅茶の、1999 年の店舗別売上合計と各店舗の順位は？

CREATE VIEW 文

```
create view tea_sales99
    as select prod_name, store_name, sum(dollars) as
tea_dollars,
    rank() over(order by tea_dollars desc) as tea_rank
from sales natural join product
    natural join period
    natural join store
where sales.classkey in (2, 5) and year = 1999
group by prod_name, store_name;
```

SQL 文例

```
select prod_name, store_name, tea_dollars, tea_rank
from tea_sales99
where prod_name like 'Assam Gold%';
```

実行結果

Prod_Name	Store_Name	Tea_Dollars	Tea_Rank
Assam Gold Blend	Beans of Boston	6201.50	15
Assam Gold Blend	Beaches Brew	6080.00	16
Assam Gold Blend	Texas Teahouse	5422.50	17
Assam Gold Blend	Olympic Coffee Company	5350.50	18
Assam Gold Blend	Cupertino Coffee Supply	5277.00	19
Assam Gold Blend	Moroccan Moods	5178.50	20
Assam Gold Blend	Coffee Brewers	5151.00	21
Assam Gold Blend	Moulin Rouge Roasting	4977.00	22
Assam Gold Blend	East Coast Roast	4769.00	25
Assam Gold Blend	Miami Espresso	4506.50	28
Assam Gold Blend	Roasters, Los Gatos	4414.50	29
Assam Gold Blend	San Jose Roasting Company	4226.50	32
Assam Gold Blend	Instant Coffee	4190.50	33
Assam Gold Blend	Java Judy's	3776.50	40

ビューからの選択

データベースに格納されたファクトとディメンジョンに対して、特定の製品や期間だけについて分析する場合があります。ビューを作成することで、クエリの対象となる特定のデータを簡単かつ高速にアクセスできるようになります。ビューは読み取り専用のテーブルで、既存のテーブルやビューから抽出した情報のサブセットが格納されます。

CREATE VIEW の構文

```
CREATE VIEW <view_name> AS <query_expression>
```

<query_expression>　『SQL Reference Guide』で定義されているジョイン クエリ式または非ジョイン クエリ式

事前計算ビューを作成するために、別の USING 句を追加する必要があります。事前計算ビューについては、『IBM Red Brick Vista User's Guide』を参照してください。

例題について

このビューには、以下の 4 列が格納されています。

- 製品名 (**Prod_Name**)
- 店舗名 (**Store_Name**)
- 1999 年の店舗別、紅茶製品別売上合計 (**Tea_Totals**)
- 売上合計に基づく各店舗の順位 (**Tea_Rank**)

このクエリは、**Prod_Name** 列だけに制約を適用し、Assam Gold Blend 紅茶に関する各店舗の売上合計と順位を返します。

次の検索条件により、ビューが紅茶製品だけを選択するようにします。

```
where sales.classkey in (2, 5)
```

Class テーブルの **Classkey** の値は製品グループを表しています。

INSERT INTO SELECT 文

例題

衣料製品の日別売上と累積合計を格納するテンポラリ テーブルを作成してください。このテンポラリ テーブルから、Los Angeles の店舗に関するデータだけを抽出する SELECT 文を実行してください。

CREATE TEMPORARY TABLE 文

```
create temporary table clothing_sales
(date date,
prod_name char(30),
city char(20),
dollars dec(7,2),
cume_tot integer);
```

INSERT 文

```
insert into clothing_sales
    (date, prod_name, city, dollars, cume_tot)
        select date, prod_name, city, dollars, cume(dollars)
        from store s join sales l on s.storekey = l.storekey
            join period t on l.perkey = t.perkey
            join product p on l.classkey = p.classkey
                and l.prodkey = p.prodkey
            join class c on p.classkey = c.classkey
        where class_type = 'Clothing'
        order by date, city
        reset by date;


** INFORMATION ** (209) Rows inserted: 816.
```

SQL 文例

```
select date, prod_name, dollars, cume_tot
from clothing_sales
where city = 'Los Angeles'
    and extract(year from date) = 2000
order by date;
```

実行結果

Date	Prod_Name	Dollars	Cume_Tot
2000-01-08	Aroma t-shirt	197.10	308
2000-01-18	Aroma t-shirt	131.40	131
2000-01-18	Aroma baseball cap	135.15	266
2000-01-23	Aroma baseball cap	15.90	15
2000-02-01	Aroma t-shirt	175.20	175
2000-02-04	Aroma t-shirt	164.25	164
...			

テンポラリ テーブルの作成

データベース リソースの使用権またはデータベース アクセス (DBA) 権限を持っている場合は、クエリのリザルト セットを格納するテンポラリ テーブルを作成することができます。テンポラリ テーブルは、元のクエリを再実行せずに、リザルト セットに対する分析を繰り返し実行する場合に便利です。たとえば、分析関数の結果をテンポラリ テーブルに格納し、この結果データにさらに制約を加える SELECT 文をテンポラリ テーブルに対して実行することができます。

INSERT INTO SELECT

```
INSERT INTO <table_name> <select_statement>
```

<table_name>　有効なテーブル名

<select_statement>　『SQL Reference Guide』に定義されている完全なまたは部分的な SELECT 文

例題について

この例は、**Clothing_Sales** というテンポラリ テーブルを作成し、日別売上と累積合計をテーブルに挿入し、標準の SELECT 文によってテーブルへの問い合わせを行う方法を示しています。

標準的な SELECT 文で検索条件が制限されていても、この例におけるクエリの結果は抽出できます。ただし、累積合計を格納するテンポラリ テーブルを作成することで、大きなファクト テーブルを使用して処理を行うときにクエリの性能が向上します。

使用上の注意

テンポラリ テーブルは、SQL セッションの最後に自動的に削除されます。このテーブルは、同じデータベースに接続するほかのユーザは検索することができません。

IBM Red Brick Warehouse にテーブルを作成するには、データベース リソース権限または DBA 権限を持っていなければなりません。リソース権限や DBA 権限には、作成したテーブルにデータを挿入できる INSERT 特権が付随しています。権限と特権の詳細は、『SQL Reference Guide』を参照してください。

テンポラリ テーブルを作成する CREATE TABLE 文には、元となるテーブルの列と同一のデータ型とサイズを持つ列を定義してください。INSERT INTO...SELECT 文からの入力データとテンポラリ テーブルの列の互換性が得られなくなるからです。

まとめ

この章では、RISQL マクロを使用して SQL 文を簡略にする方法と、CREATE VIEW コマンド、CREATE TEMPORARY TABLE コマンド、および INSERT INTO...SELECT コマンドを使用してビューやテンポラリ テーブルを作成する方法を説明しました。

CREATE MACRO 文

```
CREATE MACRO <macro_name>(<parameter> [, <parameter>] … ) AS <definition> ;
```

マクロ名は、文字で始まる 128 字以内の文字列です。マクロ名には、大文字 / 小文字の区別がありません。RISQL キーワードは、マクロ名として使用できません。

引数を使ったマクロを呼び出す場合は、CREATE MACRO 文で定義した各引数の値も指定しなければなりません。

CREATE VIEW 文

```
CREATE VIEW <view_name> AS <query_expression>
```

CREATE TEMPORARY TABLE 文

```
CREATE TEMPORARY TABLE <table_name>(<column_definitions>)
```

INSERT INTO SELECT 文

```
INSERT INTO <table_name> <select_statement>
```

A

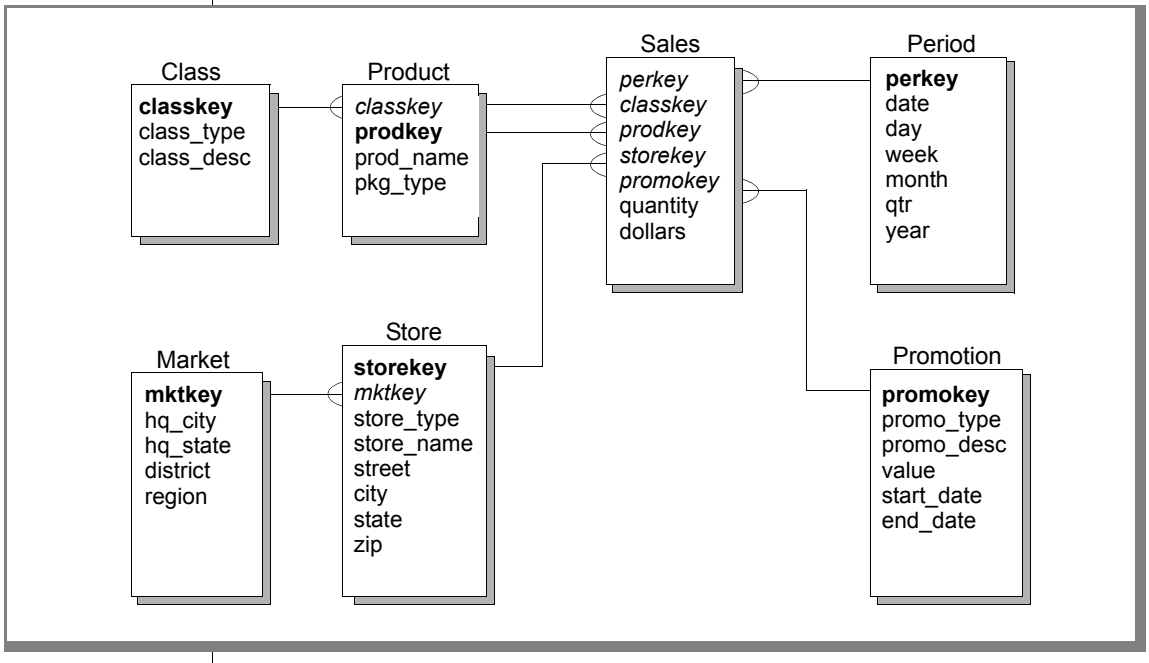
Aroma データベース詳細

付録 A では、**Aroma** データベースの全テーブルについて説明します。Aroma データベースは、販売情報を対象とした単純なスター スキーマと、仕入れ情報を対象としたマルチスター スキーマという 2 つのスキーマで構成されています。

本書の例はほとんど、販売スキーマを使用しています。仕入れ情報のテーブルは、柔軟性の高いスキーマを必要とする例をわかりやすく説明する際に使用しています。

Aroma データベース - 販売スキーマ

本書の例はほとんどの場合、Aroma Coffee and Tea Company が所有する店舗の日別販売売上を格納する、基本 **Aroma** データベースのデータを使用しています。次の図は、この基本スキーマを示したものです。



この図で 3 つの分岐線は、2 つのテーブル間に 1 対多の関係があることを表しています。たとえば、**Period** テーブルの **Perkey** 列のある値が、Period テーブルでは 1 回しか現れないのに、**Sales** テーブルでは何度も現れることがあるということを示しています。

基本の Aroma スキーマ

基本的な **Aroma** データベースは次のテーブルで構成されています。

Period	日、月、年などの期間を登録します。
Class	販売店で販売する製品のクラスを登録します。
Product	計り売りとパッケージ詰めのコーヒー、紅茶製品、コーヒー メーカーを含め、販売店で販売する製品を登録します。
Market	ビジネスの市場を地理的に分類して登録します。
Store	Aroma Coffee and Tea Company が所有し、運営する販売店を登録します。
Promotion	各種製品について実施される販売促進活動の種類、期間、割引などの関連情報を登録します。
Sales	各店舗の各期間における Aroma 製品の売上を記録します。

Period、**Class**、**Product**、**Market**、**Store**、および **Promotion** テーブルは、代表的なビジネス ディメンジョンの例です。これらのテーブルには、ユーザが理解しやすい説明的なデータが含まれています。

Sales テーブルは、ファクト テーブルの良い例です。主に集約可能なデータが格納された何千もの行から成るテーブルで、参照するディメンジョン テーブルとジョインするクエリによりアクセスすることができます。

Class テーブルと Product テーブルのサンプル データ

SQL 文

```
select * from class;
```

実行結果

Classkey	Class_Type	Class_Desc
1	Bulk_beans	Bulk coffee products
2	Bulk_tea	Bulk tea products
3	Bulk_spice	Bulk spices
4	Pkg_coffee	Individually packaged coffee products
5	Pkg_tea	Individually packaged tea products
6	Pkg_spice	Individually packaged spice products
7	Hardware	Coffee mugs, teapots, spice jars, espresso machines
8	Gifts	Samplers, gift boxes and baskets, etc.
12	Clothing	T-shirts, caps, etc.

SQL 文

```
select * from product;
```

実行結果

Classkey	Prodkey	Prod_Name	Pkg_Type
1	0	Veracruzano	No pkg
1	1	Xalapa Lapa	No pkg
1	10	Colombiano	No pkg
1	11	Expresso XO	No pkg
1	12	La Antigua	No pkg
1	20	Lotta Latte	No pkg
1	21	Cafe Au Lait	No pkg
1	22	NA Lite	No pkg
1	30	Aroma Roma	No pkg
1	31	Demitasse Ms	No pkg
2	0	Darjeeling Number 1	No pkg
2	1	Darjeeling Special	No pkg
2	10	Assam Grade A	No pkg
...			

Class テーブルと Product テーブル

Product テーブルは、**Aroma** データベースに登録された製品の情報です。**Class** テーブルは、各製品分類の情報です。

ディメンジョン テーブル内にほかのディメンジョン テーブルを参照する外部キー列がある場合、参照先のテーブルはアウトボード テーブルまたはアウトリガー テーブルと呼ばれます。**Product** テーブルの **Classkey** 列は、**Class** テーブルを参照する外部キー列なので、**Class** テーブルはアウトボード テーブルです。

列の説明 : Class テーブル

列名	内容
classkey	**Class** テーブルの行を一意に識別する整数。**Classkey** はプライマリ キーです。
class_type	製品分類名。
class_desc	製品分類を説明する文字列。

列の説明 : Product テーブル

列名	内容
classkey	**Class** テーブルを参照する外部キー。
prodkey	**Classkey** の値と合わせて、**Product** テーブルの行を一意に識別する整数。**Classkey/Prodkey** は、2 列から成る複合プライマリキーです。
prod_name	製品名。このデータベースには、59 の製品が格納されています。実際のデータベースは、製品数がさらに多くなります。Aroma 製品には、製品名が同じであっても分類やパッケージの種類が異なるものがあることに注意してください。
pkg_type	各製品のパッケージの種類を示す文字列。

Store テーブルと Market テーブルのサンプル データ

SQL 文

```
select * from market;
```

実行結果

Mktkey	HQ_City	HQ_State	District	Region
1	Atlanta	GA	Atlanta	South
2	Miami	FL	Atlanta	South
3	New Orleans	LA	New Orleans	South
4	Houston	TX	New Orleans	South
5	New York	NY	New York	North
...				

SQL 文

```
select * from store;
```

実行結果

Storekey	Mktkey	Store_Type	Store_Name	Street	City	State	Zip
1	14	Small	Roasters, Los Gatos	1234 University Ave	Los Gatos	CA	95032
2	14	Large	San Jose Roasting	5678 Bascom Ave	San Jose	CA	95156
3	14	Medium	Cupertino Coffee	987 DeAnza Blvd	Cupertino	CA	97865
4	3	Medium	Moulin Rouge	898 Main Street	New Orleans	LA	70125
5	10	Small	Moon Pennies	98675 University	Detroit	MI	48209
6	9	Small	The Coffee Club	9865 Lakeshore Bl	Chicago	IL	06060
...							

レイアウト上、列が一部省略されています。

Market テーブルと Store テーブル

Store テーブルは、Aroma 製品を販売する店舗を登録します。**Market** テーブルは、各店舗が所属する市場について説明します。各市場は、主要な都市名で表されます。**Market** テーブルは、**Class** テーブルと同様にアウトボード テーブルです。

Market テーブル - 列の説明

列名	内容
mktkey	**Market** テーブルの行を一意に識別する整数。**Mktkey** は、プライマリ キーです。
hq_city	都市名。**Market** テーブルには、17 の都市が登録されています。実際のデータベースには、何千もの都市を登録することができます。
state	州の名前。
district	主要な都市に基づく地区名。国際的なデータベースでは、国名または地理的なディメンジョンも登録されます。
region	地域名。**Market** テーブルには、全米で 4 地域だけが登録されています。本格的なデータベースでは、地域数や地域内の地区数が多くなります。

Store テーブル - 列の説明

列名	内容
storekey	**Store** テーブルの行を一意に識別する整数。**Storekey** は、プライマリ キーです。
mktkey	**Market** テーブルを参照する外部キー。
store_type	店舗の規模を示す文字列。
store_name	店名。
street, city, state, zip	各店舗の所在地を識別する列。

Period テーブルのサンプル データ

SQL 文

```
select * from period;
```

実行結果

Perkey	Date	Day	Week	Month	Qtr	Year
1	1998-01-01	TH	1	JAN	Q1_98	1998
2	1998-01-02	FR	1	JAN	Q1_98	1998
3	1998-01-03	SA	1	JAN	Q1_98	1998
4	1998-01-04	SU	2	JAN	Q1_98	1998
5	1998-01-05	MO	2	JAN	Q1_98	1998
6	1998-01-06	TU	2	JAN	Q1_98	1998
7	1998-01-07	WE	2	JAN	Q1_98	1998
8	1998-01-08	TH	2	JAN	Q1_98	1998
9	1998-01-09	FR	2	JAN	Q1_98	1998
10	1998-01-10	SA	2	JAN	Q1_98	1998
11	1998-01-11	SU	3	JAN	Q1_98	1998
12	1998-01-12	MO	3	JAN	Q1_98	1998
13	1998-01-13	TU	3	JAN	Q1_98	1998
14	1998-01-14	WE	3	JAN	Q1_98	1998
15	1998-01-15	TH	3	JAN	Q1_98	1998
16	1998-01-16	FR	3	JAN	Q1_98	1998
17	1998-01-17	SA	3	JAN	Q1_98	1998

(1 / 2)

Perkey	Date	Day	Week	Month	Qtr	Year
18	1998-01-18	SU	4	JAN	Q1_98	1998
19	1998-01-19	MO	4	JAN	Q1_98	1998
20	1998-01-20	TU	4	JAN	Q1_98	1998
...						

(2 / 2)

Period テーブル

Period テーブルは、1998 年と 1999 年ならびに 2000 年の第 1 四半期の期間の日、週、月、四半期、年を登録します。

列の説明

列名	内容
perkey	**Period** テーブルの行を一意に識別する整数。**Perkey** は、プライマリ キーです。
date	1998 年 1 月 1 日～ 2000 年 3 月 31 日までの日付。
day	曜日を表す文字列。
week	年間の各週を表す 1 ～ 53 までの整数。各週は、日曜日から始まります。
month	月名を表す文字列。
qtr	年間の四半期を一意に識別する表す文字列 (たとえば、Q1_98、Q3_99) 。
year	年を表す整数。

Promotion テーブルのサンプル データ

SQL 文

```
select * from promotion;
```

実行結果

Promokey	Promo_Type	Promo_Desc	Value	Start_Date	End_Date
0	1	No promotion	0.00	9999-01-01	9999-01-01
1	100	Aroma catalog coupon	1.00	1998-01-01	1998-01-31
2	100	Aroma catalog coupon	1.00	1998-02-01	1998-02-28
3	100	Aroma catalog coupon	1.00	1998-03-01	1998-03-31
4	100	Aroma catalog coupon	1.00	1998-04-01	1998-04-30
5	100	Aroma catalog coupon	1.00	1998-05-01	1998-05-31
6	100	Aroma catalog coupon	1.00	1998-06-01	1998-06-30
7	100	Aroma catalog coupon	1.00	1998-07-01	1998-07-31
8	100	Aroma catalog coupon	1.00	1998-08-01	1998-08-31
9	100	Aroma catalog coupon	1.00	1998-09-01	1998-09-30
10	100	Aroma catalog coupon	1.00	1998-10-01	1998-10-31
11	100	Aroma catalog coupon	1.00	1998-11-01	1998-11-30
12	100	Aroma catalog coupon	1.00	1998-12-01	1998-12-31
13	100	Aroma catalog coupon	1.00	1999-01-01	1999-01-31
14	100	Aroma catalog coupon	1.00	1999-02-01	1999-02-28
15	100	Aroma catalog coupon	1.00	1999-03-01	1999-03-31
16	100	Aroma catalog coupon	1.00	1999-04-01	1999-04-30

(1 / 2)

Promokey	Promo_Type	Promo_Desc	Value	Start_Date	End_Date
17	100	Aroma catalog coupon	1.00	1999-05-01	1999-05-31
18	100	Aroma catalog coupon	1.00	1999-06-01	1999-06-30
19	100	Aroma catalog coupon	1.00	1999-07-01	1999-07-31
20	100	Aroma catalog coupon	1.00	1999-08-01	1999-08-31
...					

(2 / 2)

Promotion テーブル

Promotion テーブルは、各製品の販売促進活動の期間と内容を説明するディメンジョン テーブルです。Promotion テーブルは商品の販売条件を示すため、条件テーブルと呼ばれることもあります。

列の説明

列名	内容
promokey	**Promotion** テーブルの行を一意に識別する整数。**Promokey** は、プライマリ キーです。
promo_type	販売促進活動を表す番号またはコード。
promo_desc	販売促進活動の種類。
value	割引額やクーポン金額など、販売促進活動に関連した金額を表す 10 進数。
start_date, end_date	販売促進活動の開始日と終了日。

Sales テーブルのサンプル データ

SQL 文

```
select * from sales;
```

実行結果

Perkey	Classkey	Prodkey	Storekey	Promokey	Quantity	Dollars
2	2	0	1	116	8	34.00
2	4	12	1	116	9	60.75
2	1	11	1	116	40	270.00
2	2	30	1	116	16	36.00
2	5	22	1	116	11	30.25
2	1	30	1	116	30	187.50
2	1	10	1	116	25	143.75
2	4	10	2	0	12	87.00
2	4	11	2	0	14	115.50
2	2	22	2	0	18	58.50
2	4	0	2	0	17	136.00
2	5	0	2	0	13	74.75
2	4	30	2	0	14	101.50
2	2	10	2	0	18	63.00
2	1	22	3	0	11	99.00
2	6	46	3	0	6	36.00
2	5	12	3	0	10	40.00

(1 / 2)

Perkey	Classkey	Prodkey	Storekey	Promokey	Quantity	Dollars
2	1	11	3	0	36	279.00
2	5	1	3	0	11	132.00
2	5	10	3	0	12	48.00
...						

(2 / 2)

Sales テーブル

Sales テーブルはファクト テーブルであり、**Aroma** データベースの中で最大のテーブルで、セグメントと呼ばれる 2 つのデータベース格納領域にデータが分割されています。セグメントについては、『Administrator's Guide』を参照してください。**Sales** テーブルは、ほかの Aroma テーブルより大きいのですが、顧客サイトで使用される一般的なファクト テーブルほど大きくはありません。ファクト テーブルには通常、何百万もの行が格納されます。

複合プライマリ キー

Sales テーブルには、複合プライマリ キーが格納されています。5 つの列のそれぞれが、ほかのテーブルのプライマリ キーを参照する外部キーです。

```
perkey, classkey, prodkey, storekey, promokey
```

上記の各プライマリ キーは、**Period**、**Product**、**Store**、および **Promotion** の各ディメンジョンに **Sales** テーブル データを結びつけます。

Sales テーブルの複合プライマリ キーは、クエリ性能の良い STARindex™ という構造を採用しています。STAR インデックスがあると、クエリ内で販売テーブルをジョインするときに、STARjoin™ という処理ができます。ジョインを必要とするクエリの例は、第 5 章「ジョインとユニオン」を参照してください。STAR インデックスの詳細は、『Administrator's Guide』を参照してください。

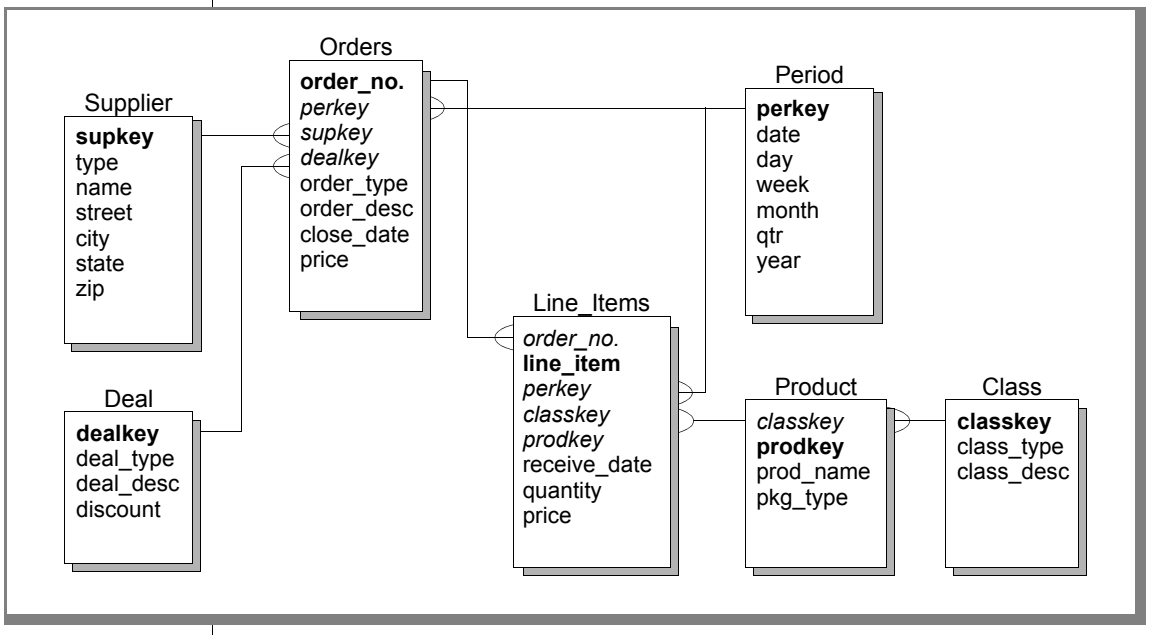
列の説明

列名	内容
perkey	**Period** テーブルを参照する外部キー。
classkey	**Product** テーブルを参照する外部キー。
prodkey	**Product** テーブルを参照する外部キー。
storekey	**Store** テーブルを参照する外部キー。
promokey	**Promotion** テーブルを参照する外部キー。
quantity	日別の総販売数量を表す整数。
dollars	日別の売上金額を表す 10 進数。

Aroma データベースの仕入れスキーマ

本書の例には、Aroma Company が仕入先に発注した製品の注文情報のテーブルを使用するものもあります。この仕入れスキーマは、販売スキーマと共通の **Product**、**Class**、および **Period** ディメンジョンを使用しますが、**Deal** と **Supplier** という固有のディメンジョンも使用します。**Line_Items** テーブルと **Orders** テーブルは、どちらもファクトを記録します。**Orders** テーブルは、**Line_Items** テーブルの参照するディメンジョン テーブルとして使うこともできます。

次の図は、仕入れスキーマの各テーブルを示したものです。



マルチスター スキーマ

Line_Items テーブルと **Orders** テーブルのプライマリ キーは、ディメンジョン テーブルの外部キーの組み合わせと一致しません。ディメンジョン テーブルのプライマリ キーの任意の組み合わせによって、ファクト テーブル内の複数の行を指示できます。この種類のテーブルは、マルチスター ファクト テーブルまたはデータ リストと呼ばれます。

たとえば、**Orders** テーブルの異なる注文番号に対する、**Supplier**、**Deal**、および **Period** の組み合わせが同一になる場合があります。

Order_No	Perkey	Supkey	Dealkey
3699	817	1007	0
3700	817	1007	0

仕入れテーブル

仕入れスキーマには、価格や数量といった、**Sales** テーブル内のファクトと同じ種類のファクトが格納されています。価格は、仕入先に対する注文単位または品目単位の支払金額です。数量は発注した製品の数量です。

仕入れスキーマを使用して、Aroma の仕入履歴について問合せをすることができます。たとえば、ある製品の割引率が最も高い仕入先はどこか、注文への対応が最も早い仕入先はどこか、などです。

Aroma Company が仕入先に発注する製品と各店舗で販売する製品は共通なので、両方のスキーマを使ったクエリを作成し、注文量と販売量を比較したり、単純な収益計算を行うこともできます。

Aroma データベースの仕入れスキーマは次のテーブルで構成されています。

Period	日、月、年などの期間を登録します。
Class	製品分類 (販売、仕入れに共通) を登録します。
Product	販売店で販売する製品と仕入先に発注する製品 (販売、仕入れに共通) を登録します。
Supplier	Aroma Company が発注する製品の仕入先を登録します。
Deal	仕入先が受注した製品に適用する割引を登録します。
Line_Items	品目単位の価格や数量など、注文に関する品目単位の詳細を格納します。
Orders	注文単位の金額、発注した製品のタイプなど、注文に関する情報を格納します。

Supplier テーブルと **Deal** テーブルは仕入れスキーマに固有のもので、**Orders** テーブルから参照されます。

ヒント：仕入れスキーマには、2000 年第 1 四半期のデータしか登録されていません。

Supplier テーブルと Deal テーブルのサンプル データ

SQL 文

```
select * from supplier;
```

実行結果

Supkey	Type	Name	Street	City	State	Zip
1001	Bulk coffee	CB Imports	100 Church Stre	Mountain View	CA	94001
1002	Bulk tea	Tea Makers,	1555 Hicks Rd.	San Jose	CA	95124

...

レイアウト上、列が一部省略されています。

SQL 文

```
select * from deal;
```

実行結果

Dealkey	Deal_Type	Deal_Desc	Discount
0	1000	No deal	0.00
1	100	Orders over $10,000	100.00
2	100	Orders over $20,000	500.00
3	100	Supplier catalog coupon	50.00
4	100	Supplier catalog coupon	100.00

(1 / 2)

Dealkey	Deal_Type	Deal_Desc	Discount
37	200	Supplier coffee special	75.00
38	200	Supplier coffee special	50.00
39	200	Supplier tea special	40.00
40	200	Supplier tea special	20.00

(2 / 2)

Supplier テーブルと Deal テーブル

列の説明 :Supplier テーブル

列名	内容
supkey	**Supplier** テーブルの行を一意に識別する整数。**Supkey** は、プライマリ キーです。
type	製品分類名を表す文字列。
name	仕入先の名称を表す文字列。
street, city, state, zip	仕入先の所在地を識別する列。

列の説明 :Deal テーブル

列名	内容
dealkey	**Deal** テーブルの行を一意に識別する整数。**Dealkey** は、プライマリ キーです。
deal_type	割引の種類を表す整数 (コード番号)。
deal_desc	割引の種類の記述を表す文字列。
discount	注文に適用された割引金額を示す 10 進数。

共有ディメンジョン

仕入れスキーマと販売スキーマは、**Period**、**Product**、**Class** の各テーブルを共有しています。

販売スキーマと仕入れスキーマについて、それぞれ個別に問い合わせることも、両スキーマのテーブルを含んだクエリを作成することも可能です。たとえば、**Sales** テーブルと **Line_Items** テーブルをジョインし、仕入れた製品と販売した製品の数量を比較することができます。このようなクエリでは、共有ディメンジョンを使用して製品や期間を限定します。

Orders テーブルと Line_Items テーブルのサンプル データ

SQL 文

```
select * from orders;
```

実行結果

Order_No	Perkey	Supkey	Dealkey	Order_Type	Order_Desc	Close_Date	Price
3600	731	1001	37	Coffee	Whole coffee b	2000-01-07	1200.46
3601	732	1001	37	Coffee	Whole coffee b	2000-01-07	1535.94
3602	733	1001	0	Tea	Loose tea, bul	2000-01-07	780.00
3603	740	1001	39	Tea	Loose tea, bul	2000-01-21	956.45
3604	744	1005	0	Spice	Pre-packed spi	2000-01-16	800.66
3605	768	1003	2	Coffee	Whole-bean and	2000-02-12	25100.00
3606	775	1003	2	Coffee	Whole-bean and	2000-02-19	25100.00
3607	782	1003	2	Coffee	Whole-bean and	2000-02-25	25100.00

(1 / 2)

Order_No	Perkey	Supkey	Dealkey	Order_Type	Order_Desc	Close_Date	Price
3608	789	1003	2	Coffee	Whole-bean and	2000-03-03	30250.00
3609	796	1003	2	Coffee	Whole-bean and	2000-03-15	25100.00
...							

SQL 文

```
select * from line_items;
```

実行結果

Order_No	Line_Item	Perkey	Classkey	Prodkey	Receive_Da	Qty	Price
3600	1	731	1	1	2000-01-07	40	180.46
3600	2	731	2	10	2000-01-07	150	300.00
3600	3	731	2	11	2000-01-07	80	240.00
3600	4	731	2	12	2000-01-07	150	240.00
3600	5	731	1	20	2000-01-07	60	240.00
3601	1	732	1	0	2000-01-07	60	240.00
3601	2	732	1	1	2000-01-07	60	240.00
3601	3	732	1	10	2000-01-07	60	240.00
3601	4	732	1	11	2000-01-07	60	240.00
3601	5	732	1	12	2000-01-07	60	240.00
3601	6	732	1	31	2000-01-07	70	335.94
3602	1	733	2	0	2000-01-08	70	130.00
3602	2	733	2	1	2000-01-08	70	130.00
...							

Orders テーブルと Line_Items テーブル

Orders テーブルと **Line_Items** テーブルには、仕入れに関する事実 (ファクト) が記録されています。各テーブルの詳細は、A-16 ページを参照してください。

列の説明 : Orders テーブル

列名	内容
order_no	**Orders** テーブルの行を一意に識別する整数。Order_No は、プライマリ キーです。
perkey	**Period** テーブルを参照する外部キー。
supkey	**Supplier** テーブルを参照する外部キー。
dealkey	**Deal** テーブルを参照する外部キー。
order_type	発注した製品分類を表す文字列。
order_desc	注文の種類の記述を表す文字列。
close_date	注文の決済日 (締め日) 。
price	注文の全額を示す 10 進数。

列の説明 : Line_Items テーブル

列名	内容
order_no	**Orders** テーブルの行を一意に識別する整数。**Order_No** は、プライマリ キーです。
line_item	発注品目の番号を表す整数。
perkey	**Period** テーブルを参照する外部キー。
classkey	**Product** テーブルを参照する外部キー。
prodkey	**Product** テーブルを参照する外部キー。

(1 / 2)

列名	内容
receive_date	品目を受領した日付。
quantity	品目単位の発注数量を表す整数。
price	品目単位の原価を表す 10 進数。

（2 / 2）

特記事項

本書に記載されている製品、サービス、または機能は、国によっては提供されない場合があります。各地域で現在利用可能な IBM 製品およびサービスについては、該当地域の担当者にお問い合わせください。IBM 製品、プログラム、またはサービスのすべての記述箇所は、対象の IBM 製品、プログラム、またはサービスのみが使用されることを明示的または暗示的に示すものではありません。IBM の知的所有権を侵害しない、同等の機能を有する製品、プログラム、またはサービスを使用できる場合があります。ただし、IBM 以外の製品、プログラム、またはサービスの操作に関する評価および確認は、お客様の責任の元に行ってください。

本書に記載の各事項は、特許または特許出願により保護されている場合があります。本書の提供は、これらの特許の使用許諾をお客様に付与するものではありません。使用許諾に関する質問は、下記に書面にてお問い合わせください。

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

ダブル バイト (DBCS) に関する使用許諾については、各国 IBM の知的財産部門にお問い合わせいただくか、下記まで書面にてお問い合わせをご送付ください。

IBM World Trade Asia Corporation
Licensing
〒 106 - 0032 東京都港区六本木 3 丁目 2 - 31

以下の段落の記述は、英国、または記述の内容が地域法に合致しない国には適用されません。International Business Machines Corporation は、本書を「現状のまま」提供し、明示的または黙示的を問わずいかなる保証の責任も負わないものとし、第三者の権利侵害のないことへの保証、商品性の保証、または特定目的への適合性の保証などを含み、かつこれらに限定されない、いかなる黙示的な保証の責任も負わないものとします。特定の取引における明示的または黙示的な保証の制限が国または地域によって禁じられる場合、この記述は適用されないものとします。

この情報には、技術上不正確な点や誤植が含まれている可能性があります。本書の情報は定期的に見直され、必要な変更は本書の改訂版に反映されます。IBM は本書に記載されている製品またはプログラムに対して、随時、予告なしに変更または改良を加えることがあります。

本書において参照されている IBM 以外の Web サイトは、単に便宜上の理由で記載されているだけであり、それらのサイトに対する IBM の保証または支持を示すものではありません。それらの Web サイトで提供される内容は、本 IBM 製品には関係ありません。これらの利用は、お客様の責任の元で行ってください。

お客様により提供された情報は、IBM が適切と判断するすべての方法で、使用または頒布される場合があります。これにより、お客様に責任が発生することはありません。

本プログラムのライセンス保持者で、次の事項を可能にするための情報を希望する方は、下記にお問い合わせください。(i) 独自に作成されたプログラムと、他のプログラム (本製品を含む) との間の情報交換、(ii) 交換された情報の共用

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

上記の情報は、適切な使用条件の元で利用可能ですが、有償の場合もあります。

本書で説明されるライセンス プログラムおよびこれに関して利用できるライセンス資料は、IBM Customer Agreement、IBM International Program License Agreement、またはそれと同等の合意の各条項に基づいて、IBM より提供されます。

この文書に含まれるすべての実行データは、管理環境下で決定されたものです。このため、操作環境によって実行結果が大きく変化する場合があります。開発レベルのシステムで測定が行われた可能性がありますが、その測定値が一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値には推定値が含まれている場合があります。このため、実際の結果とは異なる可能性があります。本書を利用されるお客様は、それぞれの特定の環境に適したデータを検証する必要があります。

IBM 以外の製品に関する情報は、各製品の供給者、出版物、または公的に入手可能な情報源から取得されたものです。IBM ではこれらの製品をテストしておらず、IBM 以外の製品のパフォーマンス、互換性、またはその他の要件については確認していません。IBM 以外の製品の機能に関する質問については、各製品の供給者にお問い合わせください。

商標

AIX、DB2、DB2 Universal Database、Distributed Relational Database Architecture、NUMA-Q、OS/2、OS/390、および OS/400、IBM Informix®、C-ISAM®、Foundation.2000™、IBM Informix® 4GL、IBM Informix® DataBlade® Module、Client SDK™、Cloudscape™、Cloudsync™、IBM Informix® Connect、IBM Informix® Driver for JDBC、Dynamic Connect™、IBM Informix® Dynamic Scalable Architecture™ (DSA)、IBM Informix® Dynamic Server™、IBM Informix® Enterprise Gateway Manager (Enterprise Gateway Manager)、IBM Informix® Extended Parallel Server™、i.Financial Services™、J/Foundation™、MaxConnect™、Object Translator™、Red Brick™、IBM Informix® SE、IBM Informix® SQL、InformiXML™、RedBack®、SystemBuilder™、U2™、UniData®、UniVerse®、wintegrate® は、International Business Machines Corporation の商標または登録商標です。

Java および Java ベースの商標およびロゴは、米国およびその他の国における Sun Microsystems, Inc の商標または登録商標です。

Windows、Windows NT、および Excel は、米国およびその他の国における Microsoft Corporation の商標または登録商標です。

UNIX は、米国およびその他の国における、X/Open Company Limited が独占的にライセンスしている登録商標です。

本書におけるその他の会社、製品およびサービスの名称は、それぞれ各社の商標またはサービス マークです。

索引

記号

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

は

ひ

ふ

へ

ま

め

も

ゆ

ら

り

る

れ

ろ

わ