Oracle9*i* OLAP

開発者ガイド -Oracle OLAP DML

リリース 2（9.2）

2003 年 5 月

部品番号 : J07215-01

ORACLE®

Oracle9*i* OLAP 開発者ガイド -Oracle OLAP DML, リリース 2（9.2）

部品番号 : J07215-01

原本名：Oracle9*i* OLAP Developer's Guide to the OLAP DML, Release 2 (9.2)

原本部品番号：A95298-01

目次

3 データ・オブジェクトの定義

4 式の使用

6 データの選択

第 II 部　アプリケーション開発

7　プログラムの開発

8 モデルの使用

9 データのアロケーション

第 III 部 アナリティック・ワークスペース管理

10 リレーショナル表の処理

11 ファイルからのデータの読込み

12 データの集計

索引

はじめに

このマニュアルでは、プログラミング環境の概要を示し、アナリティック・ワークスペース・オブジェクトについて説明します。また、OLAP DML の機能を使用する方法についても説明します。さらに、プログラムを作成およびデバッグする方法について説明し、データへのアクセスおよびデータの処理を行うプログラムのプログラミング方法を示します。

> **注意：** このマニュアルで示すすべての OLAP DML コマンドの詳細は、Oracle9*i* OLAP DML Reference ヘルプを参照してください。特定のコマンドの詳細を表示するには、そのコマンド名を指定して検索してください。

ここでは、次の項目について説明します。

対象読者

このマニュアルは、次のタスクを実行するユーザーを対象としています。

- 多次元データへのアクセス
- OLAP DML を使用した分析
- アナリティック・ワークスペースの管理

このマニュアルを読むには、プログラミングの経験が役立ちますが、必須ではありません。

このマニュアルの構成

このマニュアルは、次のように構成されています。

第 I 部「概要」

第 1 章「基本的な概念」

この章では、OLAP データ操作言語の概要を示し、それにアクセスするための様々な方法について説明します。

第 2 章「アナリティック・ワークスペースの定義および処理」

この章では、新しいアナリティック・ワークスペースを作成する方法、および既存のアナリティック・ワークスペースを変更する方法について説明します。また、初期化プログラムおよびパスワード保護についても説明します。

第 3 章「データ・オブジェクトの定義」

この章では、様々なタイプのアナリティック・ワークスペース・オブジェクトについて説明し、これらを作成する方法について説明します。また、アナリティック・ワークスペースのデータ型を定義します。

第 4 章「式の使用」

この章では、式を定義および使用する方法について説明します。

第 5 章「アナリティック・ワークスペースの データ・オブジェクトの移入」

この章では、ディメンション・メンバーの追加、削除および並替えを行う方法、およびデータ・オブジェクトに値を代入する方法について説明します。

第 6 章「データの選択」

この章では、分析または表示用のデータを選択する方法について説明します。

第 II 部「アプリケーション開発」

第 7 章「プログラムの開発」
この章では、DML ストアド・プロシージャを作成、変更、コンパイルおよび実行する方法について説明します。

第 8 章「モデルの使用」
この章では、一連の方程式を作成、コンパイルおよび実行する方法について説明します。

第 9 章「データのアロケーション」
この章では、1 つ以上のディメンションで親から子へデータを分散する方法について説明します。

第 III 部「アナリティック・ワークスペース管理」

第 10 章「リレーショナル表の処理」
この章では、リレーショナル表からアナリティック・ワークスペース・オブジェクトへデータをフェッチする方法、およびアナリティック・ワークスペース・オブジェクトからリレーショナル表へデータを挿入する方法について説明します。

第 11 章「ファイルからのデータの読込み」
この章では、フラット・ファイルからアナリティック・ワークスペース・オブジェクトへデータをコピーする方法について説明します。

第 12 章「データの集計」
この章では、低レベルのデータをロールアップする方法について説明します。

関連文書

詳細は、次の Oracle リソースを参照してください。

- Oracle9*i* OLAP DML Reference ヘルプ
- 『Oracle9*i* OLAP ユーザーズ・ガイド』
- 『Oracle9*i* OLAP 開発者ガイド - Oracle OLAP API』
- Oracle9*i* OLAP API Javadoc
- 『Oracle9*i* データ・ウェアハウス・ガイド』

リリース・ノート、インストール・マニュアル、ホワイト・ペーパーまたはその他の関連文書は、OTN-J（Oracle Technology Network Japan）に接続すれば、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

```
http://otn.oracle.co.jp/membership/
```

すでに OTN-J のユーザー名およびパスワードを取得済であれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

```
http://otn.oracle.co.jp/documents/
```

表記規則

この項では、このマニュアルの本文およびコード例で使用される表記規則について説明します。この項の内容は次のとおりです。

- 本文中の表記規則
- コード例中の表記規則

本文中の表記規則

本文では、特別な用語をより迅速に識別できるように、様々な表記規則を使用します。次の表に、それらの表記規則を説明し、その使用例を示します。

規則	意味	例
太字	太字は、本文中で定義されている用語または用語集に記載されている用語（あるいはその両方）を示します。	この句を指定すると、**索引構成表**が作成されます。
固定幅フォントの大文字	固定幅フォントの大文字は、システムが提供する要素を示します。このような要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージおよびメソッドが含まれます。また、システムが提供する列名、データベース・オブジェクト、データベース構造、ユーザー名およびロールも含まれます。	NUMBER 列に対してのみに、この句を指定できます。 BACKUP コマンドを使用して、データベースのバックアップを取ることができます。 USER_TABLES データ・ディクショナリ・ビュー内の TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。

規則	意味	例
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびユーザーが提供する要素のサンプルを示します。このような要素には、コンピュータ名およびデータベース名、ネット・サービス名および接続識別子が含まれます。また、ユーザーが提供するデータベース・オブジェクトとデータベース構造と列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニットおよびパラメータ値も含まれます。 **注意：**大文字と小文字を組み合せて使用するプログラム要素もあります。これらの要素は、記載されているとおり入力してください。	`sqlplus` と入力して、SQL*Plus をオープンします。 パスワードは、`orapwd` ファイルで指定します。 `/disk1/oracle/dbs` ディレクトリ内のデータ・ファイルおよび制御ファイルのバックアップを取ります。 `hr.departments` 表には、`department_id`、`department_name` および `location_id` 列があります。 `QUERY_REWRITE_ENABLED` 初期化パラメータを `true` に設定します。 `oe` ユーザーとして接続します。 `JRepUtil` クラスが次のメソッドを実装します。
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*`parallel_clause`* を指定できます。 *`old_release`*`.SQL` を実行します。ここで、*`old_release`* とはアップグレード前にインストールしたリリースを示します。

コード例中の表記規則

コード例は、SQL、PL/SQL、SQL*Plus または他のコマンドライン文を説明します。コード例は、固定幅フォントで表示され、次の例に示すとおり通常のテキストと区別されます。

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

次の表に、コード例で使用される表記規則を説明し、その使用例を示します。

規則	意味	例
[ ]	大カッコは、任意に選択する 1 つ以上の項目を囲みます。大カッコは、入力しないでください。	`DECIMAL (`*`digits`* `[ , `*`precision`* `])`
{ }	中カッコは、2 つ以上の項目を囲み、そのうちの 1 つの項目は必須です。中カッコは、入力しないでください。	`{ENABLE | DISABLE}`
|	縦線は、大カッコまたは中カッコ内の 2 つ以上のオプションの選択項目を表します。いずれかのオプションを入力します。縦線は入力しないでください。	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

規則	意味	例
. . .	水平の省略記号は、次のいずれかを示します。	
	■ 例に直接関連しないコードの一部が省略されている。	CREATE TABLE ... AS *subquery*;
	■ コードの一部を繰り返すことができる。	SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected
その他の句読点	大カッコ、中カッコ、縦線および省略記号以外の句読点は、表示されているとおり入力する必要があります。	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。	CONNECT SYSTEM/*system_password* DB_NAME = *database_name*
大文字	大文字は、システムが提供する要素を示します。これらの用語は、ユーザー定義の用語と区別するために大文字で示されます。用語が大カッコ内にないかぎり、表示されているとおりの順序および綴りで入力します。ただし、これらの用語は大 / 小文字が区別されないため、小文字でも入力できます。	SELECT last_name, employee_id FROM employees; SELECT * FROM USER_TABLES; DROP TABLE hr.employees;
小文字	小文字は、ユーザー定義のプログラム要素を示します。たとえば、表名、列名、ファイル名などです。 **注意 :** 大文字と小文字を組み合せて使用するプログラム要素もあります。これらの要素は、記載されているとおり入力してください。	SELECT last_name, employee_id FROM employees; sqlplus hr/hr CREATE USER mjones IDENTIFIED BY ty3MU9;

OLAP DML の新機能

Oracle9*i* リリース 2（9.2）では、Oracle データベースでの多次元分析用の OLAP データ操作言語（DML）が提供されています。OLAP オプションをインストールすると、DML コマンドを実行してアナリティック・ワークスペース内のデータを操作できるようになります。OLAP DML には、Oracle OLAP Server リリース 6.3 からの新機能および変更された機能が含まれています。

参照：

- Oracle9*i* リリース 2（9.2）の OLAP オプションの概要、および OLAP Server と Oracle OLAP の相違点については、『Oracle9*i* OLAP ユーザーズ・ガイド』を参照してください。
- OLAP DML で追加、削除、名前変更、および大幅に変更されたコマンドのリストは、Oracle9*i* OLAP DML Reference ヘルプを参照してください。

次の項では、Oracle9*i* OLAP の新機能について説明します。

Oracle9*i* リリース 2（9.2）の OLAP DML の新機能

この項では、OLAP DML の新機能の概要を示します。

- **名前が変更された DATABASE コマンドおよびその関連コマンド**

  OLAP 計算エンジンは Oracle カーネルで実行され、アナリティック・ワークスペースはリレーショナル表に格納されるため、アナリティック・ワークスペースを格納する個別のファイルは存在しません。この変更は、既存の DML コマンドの新しい名前および機能に反映されます。

  参照： 第 2 章「アナリティック・ワークスペースの定義および処理」を参照してください。

- **XCA、SNAPI または ODBC ではなく、SQL および OLAP API を介した OLAP DML へのアクセス**

  XCA、SNAPI および ODBC を介した接続はサポートされません。関連コマンドは廃止されています。新しいアクセス方法では、セッション共有がサポートされないことに注意してください。

  参照： SQL アクセスの詳細は、『Oracle9*i* OLAP ユーザーズ・ガイド』および『Oracle9*i* OLAP 開発者ガイド - Oracle OLAP API』を参照してください。

- **アナリティック・ワークスペースへの変更を保存するには、UPDATE と COMMIT の両方のコマンドの使用が必要**

  `UPDATE` コマンドを実行すると、変更が一時領域からアナリティック・ワークスペースが格納されているデータベース表に移動します。OLAP DML または SQL のいずれかで `COMMIT` コマンドを実行するまで、変更は保存されません。

  参照： 第 2 章「アナリティック・ワークスペースの定義および処理」を参照してください。

- **カスタム集計のサポート**

  仮想ディメンション・メンバーは、`MODEL` オブジェクト内で新しい `AGGREGATION` コマンドを使用して実行時に定義できます。その後、`AGGREGATE` ファンクションが、他のすべての集計と同様にカスタム集計のデータを計算します。

  参照： 第 12 章「データの集計」を参照してください。

- **モデルを使用した非階層ディメンションのデータ集計**

  集計マップの MODEL コマンドによって、データのメンテナンス時（AGGREGATE コマンドを使用）または実行時（AGGREGATE ファンクションを使用）のいずれかにモデルが実行されます。

  参照： 第 12 章「データの集計」を参照してください。

- **階層へのデータのアロケーションのサポート**

  新しい ALLOCATE コマンドは、企業の予算システムや需要計画システムなどのプランニング・アプリケーションをサポートします。これらのアプリケーションでは、高度なアロケーション規則に基づいて、階層の低レベルにデータを割り当てる必要があります。

  参照： 第 9 章「データのアロケーション」を参照してください。

- **データベース表からアナリティック・ワークスペースへのデータのコピーを高いパフォーマンスで実行可能な SQL IMPORT コマンド**

  SQL IMPORT コマンドを実行すると、SQL FETCH 文より高速にアナリティック・ワークスペース・オブジェクトにファクト・データをロードできます。

  参照： 第 10 章「リレーショナル表の処理」を参照してください。

- **データベース表へのアナリティック・ワークスペース・データのロードを高いパフォーマンスで実行可能な SQL PREPARE コマンド**

  SQL PREPARE コマンドの新しいオプションを使用すると、リレーショナル表へのアナリティック・ワークスペース・データのダイレクト・パス・インサートを指定できます。

  参照： 第 10 章「リレーショナル表の処理」を参照してください。

- **連結ディメンションによる、複数のディメンションから 1 つのディメンションへの値の結合**

  連結ディメンションの定義では、複数のディメンションの値を 1 つのディメンションに結合することができます。連結ディメンションを使用して多次元構造をリレーショナル・スキーマにマップすることによって、リレーショナル・ソースからのデータのロードのパフォーマンスを向上できます。連結ディメンションは、カスタム集計および他のカスタム操作の実行にも使用できます。

  参照： 第 3 章「データ・オブジェクトの定義」および第 10 章「リレーショナル表の処理」を参照してください。

- **NUMBER ディメンションへの序数以外の数値の格納**

  Oracle OLAP DML には、リレーショナル・データベースの NUMBER データ型に相当する新しい NUMBER データ型が含まれています。NUMBER 値を含む NUMBER ディメンションを定義できます。Oracle OLAP は、常に NUMBER ディメンションの値を順序値ではなくディメンションの値として解釈します。NUMBER ディメンションを使用して、リレーショナル・データベース表のサロゲート・キー列など、一連の一意の数値を表すことができます。

  参照： 第 3 章「データ・オブジェクトの定義」を参照してください。

- **ディメンション・サロゲートが提供するディメンション値の代替ラベル**

  ディメンション・サロゲートは、新しい型の DML オブジェクトです。ディメンション・サロゲートはディメンションに基づいて定義しますが、ディメンションのデータ型とは異なるデータ型にすることができます。サロゲートは、ディメンションと同じ位置番号を持ちます。変数の場合と同様に、サロゲートに値を割り当てます。NUMBER ディメンションおよびディメンション・サロゲートを使用して、リレーショナル・データベースからアナリティック・ワークスペースにサロゲート・キーの値をロードし、これらのキー値を使用して、複数のリレーショナル・ファクト表から多次元構造にデータをロードできます。

  参照： 第 3 章「データ・オブジェクトの定義」を参照してください。

- **修飾オブジェクト名を使用した、アタッチされた複数のアナリティック・ワークスペースに含まれる同じ名前のオブジェクトの参照**

  OLAP DML コマンドでは、修飾オブジェクト名を使用してオブジェクトを指定できます。修飾オブジェクト名には、オブジェクトの名前のみでなく、オブジェクトが存在するアナリティック・ワークスペースの名前も含まれています。

  参照： 第 2 章「アナリティック・ワークスペースの定義および処理」を参照してください。

- **アナリティック・ワークスペースのフルネームを使用した、他のユーザーが所有するアナリティック・ワークスペースへのアクセス**

  OLAP DML コマンドでは、アナリティック・ワークスペースのフルネームを使用して、他のユーザーのスキーマに含まれるアナリティック・ワークスペースを指定できます。フルネームには、スキーマ名が含まれています。

  参照： 第 2 章「アナリティック・ワークスペースの定義および処理」を参照してください。

- **アナリティック・ワークスペース別名を使用した、アナリティック・ワークスペースの短縮名および一般名**

  フルネームより簡単に入力可能なアナリティック・ワークスペース別名を使用して、アナリティック・ワークスペースを参照できます。別名を使用すると、アナリティック・ワークスペースへの参照を含むがその名前をハードコードしない汎用コードを記述できます。

  参照： 第 2 章「アナリティック・ワークスペースの定義および処理」を参照してください。

- **ディレクトリ別名を使用した、OLAP DML コマンドでのディスク上のファイルへのアクセス**

  OLAP DML を使用してディスク・ファイルに対する読込みまたは書込みを実行する場合は、そのファイルが存在するディレクトリを直接指定しません。かわりに、Oracle データベース管理者が設定したディレクトリ別名を指定します。

  参照： 第 11 章「ファイルからのデータの読込み」および『Oracle9*i* OLAP ユーザーズ・ガイド』を参照してください。

- **データベースの NCHAR および NVARCHAR2 列のデータを保持可能な新しい NTEXT データ型**

  すべての `NTEXT` 値は、UTF-8 Unicode 変換フォーマットにエンコードされます。

  参照： 第 3 章「データ・オブジェクトの定義」を参照してください。

- **Oracle OLAP のデフォルト・キャラクタ・セットとして使用するデータベース・キャラクタ・セット**

  Oracle OLAP には、デフォルト・キャラクタ・セットを指定する構成手順がありません。Oracle OLAP のデフォルト・キャラクタ・セットは、データベース・キャラクタ・セットと同じです。

- **Oracle OLAP の NLS パラメータ設定とデータベースの NLS パラメータ設定の一元化**

  Oracle OLAP のすべての NLS 設定（`NLS_DATE_FORMAT` や `NLS_LANGUAGE` など）は、セッション全体の NLS パラメータ設定に反映されます。Oracle OLAP に NLS オプションを設定した場合、セッション全体の NLS パラメータ設定が変更されます。

  参照： Oracle9*i* OLAP DML Reference ヘルプの NLS オプションの項を参照してください。

- **読込み専用の DECIMALCHARS、THOUSANDSCHARS、YESSPELL および NOSPELL オプション**

  これらのオプションの値は、常に現行セッション全体の NLS パラメータ設定と同じになります。Oracle OLAP オプションの値を変更して、これらの設定を変更することはできません。

- **オペレーティング・システム・アクティビティへのアクセスを提供するコマンドの非サポート**

  Oracle データベース規則との互換性を保つために、Oracle OLAP はシステム・レベルの情報およびコマンドへの直接アクセスを提供しません。そのため、SYSINFO ファンクションのキーワードは削減され、CHDIR、CHDRIVE、MKDIR、SHELL などのコマンドは廃止されています。また、EXTCALL オブジェクトはサポートされません。

  参照： Oracle9*i* OLAP DML Reference ヘルプの廃止されたコマンドのリストを参照してください。

- **置換変数の非サポート**

  アナリティック・ワークスペースはデータベース表に格納されるため、置換変数の格納は実行できません。

- **対話型デバッグの非サポート**

  TRACE および WATCH コマンドを使用して OLAP Worksheet の対話型デバッグを行うことはできませんが、PRGTRACE、MODTRACE および DBGOUTFILE を使用して、プログラムおよびモデルの処理過程を記録できます。

  参照： 第 7 章「プログラムの開発」および第 8 章「モデルの使用」を参照してください。

- **OLAP DML コマンドではなく、リレーショナル・ビューを介したパフォーマンス統計の使用**

  DGCART コマンドとファンクション、および CACHEHITS、CACHEMISSES および CACHETRIES オプションは廃止されています。ただし、OLAP の動的パフォーマンス・ビューを使用してパフォーマンスを監視できます。

  参照：『Oracle9*i* OLAP ユーザーズ・ガイド』を参照してください。

- **マルチサイクル周期性が追加された予測機能の拡張**

  FCSET コマンドは、FCOPEN、FCCLOSE および FCEXEC を使用して作成された予測のマルチサイクル周期性に使用できます。

  > **参照：** Oracle9*i* OLAP DML Reference ヘルプの FCOPEN、FCCLOSE、FCEXEC、FCQUERY および FCSET コマンドの項を参照してください。

- **プログラムのストリップの非サポート**

  STRIP コマンドは廃止されています。かわりに、HIDE コマンドを使用します。以前のリリースでは、アナリティック・ワークスペース・ファイルがアプリケーションの一部として配置される前に、アナリティック・ワークスペース・ファイル内のプログラム定義がプログラムからストリップされていました。そのため、コンパイル済コードのみが配置されていました。今回のリリースでは、アナリティック・ワークスペースが、定義のみを含み、コンパイル済コードを含むことができない EIF ファイルとして配置されます。この新しいコンテキストでは、ストリップされたプログラムを実行できません。

第 I 部

概要

第 I 部では、OLAP DML の基本的な機能について説明します。

第 I 部に含まれる章は、次のとおりです。

- 第 1 章「基本的な概念」
- 第 2 章「アナリティック・ワークスペースの定義および処理」
- 第 3 章「データ・オブジェクトの定義」
- 第 4 章「式の使用」
- 第 5 章「アナリティック・ワークスペースの データ・オブジェクトの移入」
- 第 6 章「データの選択」

1

基本的な概念

この章では、OLAP DML でプログラミングを開始する前に理解しておく必要がある基本的な概念の概要を示します。この章では、次の項目について説明します。

- OLAP DML の概要
- OLAP DML の使用
- OLAP Worksheet からアナリティック・ワークスペースへのアクセス
- SQL ベースのアプリケーションからアナリティック・ワークスペースへのアクセス
- Java アプリケーションからアナリティック・ワークスペースへのアクセス

OLAP DML の概要

OLAP DML はデータ操作言語です。DML コマンドおよびファンクションを使用すると、複雑なデータ分析を実行できます。また、DML コマンドおよびファンクションを含むプログラムを作成することもできます。

OLAP DML の基本的な構文単位は次のとおりです。

- 操作を開始するコマンド
- 操作を開始し、値を戻すファンクション
- ユーザーが値を割り当て、アナリティック・ワークスペースの処理環境に様々な影響を及ぼす可能性があるオプション

OLAP DML コマンド、ファンクションおよびオプションを総称してコマンドといいます。このマニュアルでは、これらの多くのコマンドの概要を示します。各コマンドの構文、使用上の注意および例については、Oracle9*i* OLAP DML Reference ヘルプを参照してください。

OLAP DML を使用すると、アプリケーション開発者は、SQL や OLAP API などの問合せ言語の分析機能を拡張できます。

OLAP DML の用途を理解するには、次のようないくつかの重要な概念を理解しておく必要があります。

- アナリティック・ワークスペース
- SQL と OLAP DML の関係
- OLAP API と OLAP DML の関係

アナリティック・ワークスペース

アナリティック・ワークスペースは、多次元データ・ソースです。アナリティック・ワークスペースは、一時的（セッション期間中のみ）または永続的に保持することができます。アナリティック・ワークスペースが永続的な場合、データは LOB としてリレーショナル表に格納されます。

アナリティック・ワークスペースの多次元モデルは、高速で高度な計算をサポートするように設計されています。また、アナリティック・ワークスペースは、集計データの格納方法としてマテリアライズド・ビューのかわりに使用することもできます。

アプリケーションは、アナリティック・ワークスペースに存在するデータに次のいずれかの方法でアクセスできます。1 つ目の方法は、アナリティック・ワークスペース・データへのアクセス用に Oracle で提供されている PL/SQL パッケージを使用することです。もう 1 つの方法は、Java Application Programing Interface である Oracle OLAP API を使用することです。PL/SQL パッケージと OLAP API のいずれの方法でも、OLAP DML コマンドおよびプログラムを明示的に実行できます。

SQL と OLAP DML

SQL 表ファンクションは、入力として行セットを取り、物理的なデータベース表と同様に問合せ可能な出力として行セットを作成します。Oracle では、表ファンクションを使用してアナリティック・ワークスペースに存在する多次元データのビューを作成する PL/SQL パッケージが提供されています。これらのビューには、SQL アプリケーションでアクセスできます。そのため、OLAP 計算エンジンおよびアナリティック・ワークスペース・データは SQL でアクセスでき、SQL ベースのアプリケーションで分析ファンクションおよび予測ファンクションを使用可能にします。SQL アプリケーションは、Oracle Call Interface（OCI）または Java Database Connectivity（JDBC）のいずれかを使用してデータベースに接続できます。

アプリケーション・プログラマは、PL/SQL プロシージャを使用して SQL ビューとしてアナリティック・ワークスペースのデータにアクセスする他に、Oracle OLAP パッケージを使用して OLAP DML コマンドを直接実行して、結果をアプリケーションに戻すことができます。

OLAP API と OLAP DML

OLAP API を使用する Java プログラムは、SQL 表またはアナリティック・ワークスペースのいずれかに格納されたデータにアクセスできます。OLAP API は様々な分析ファンクションを提供し、アプリケーションはこれらのファンクションを使用して、データから計算されたメジャーを導出できます。

ただし、OLAP API は、予測、モデルの解決、ある種類の統合（集計）、アロケーションなどの、アプリケーションに必要なデータの計算方法を提供しない場合があります。この場合、OLAP API 内で OLAP DML コマンドを直接実行して、アナリティック・ワークスペース内でこのデータを計算することができます。

OLAP DML の使用

OLAP DML は、次のような場合に使用します。

- データ・ウェアハウスの抽出、変換およびロード（ETL）処理の一部として、または Java OLAP API を使用して計算できないデータを計算する必要がある場合。
- アプリケーションで様々な計算を実行および保持する場合で、SQL 表にこの計算をすぐにコミットしない場合。
- アナリティック・ワークスペースに格納されたデータを操作する場合。

OLAP DML を使用する一般的な計算は、次のとおりです。

- 予測
- モデル（1 つの計算の結果が別の計算の入力として使用される一連の計算）
- アロケーション（特定の分散スキームに基づいて、データをより低いレベルに分散する「逆集計」）

- 階層の加重平均などの、一部の非加算的な集計（統合）

これらの操作の他に、OLAP DML を使用して、ETL 処理または OLAP API を使用して簡単に実行できない計算を実行することもできます。

SQL 表に対してはデータをコミットせずに、アナリティック・ワークスペースに対してコミットすることができます。これは、処理中の操作に対して有効です。たとえば、予測アプリケーションで、ユーザーが個人的な予測を保存し、後のセッションでそれらを再利用できるようにするが、ユーザーが SQL 表に対してその予測をコミットしないようにする場合があります。

OLAP DML を使用してデータを分析する方法

OLAP DML を使用するには、次の手順を実行します。

1. アナリティック・ワークスペースを作成します。
2. アナリティック・ワークスペース内のデータ・オブジェクトを定義します。
3. これらのオブジェクトにデータをロードします。
4. OLAP DML コマンドおよびプログラムを定義および実行します。

OLAP DML を使用してデータを分析すると、次の操作を実行できます。

- OLAP API または SQL を使用したアナリティック・ワークスペース内のデータの表示
- SQL 表へのデータの書込み

アナリティック・ワークスペースの作成

次のようなコマンドを使用して、アナリティック・ワークスペースを作成できます。

```
AW CREATE salesforecast
```

このコマンドを実行すると、`salesforecast` という名前の、新しい空のアナリティック・ワークスペースが作成されます。

アナリティック・ワークスペースを作成する方法の詳細は、第 2 章「アナリティック・ワークスペースの定義および処理」を参照してください。

アナリティック・ワークスペースへのデータのロード

OLAP DML を使用するには、アナリティック・ワークスペースにデータが存在する必要があります。データは、SQL 表またはフラット・ファイルからアナリティック・ワークスペースにロードできます。ほとんどの場合、データベース内の表がデータ・ソースになります。アナリティック・ワークスペースにデータをロードするには、OLAP DML のコマンドを使用します。

アナリティック・ワークスペースへデータをロードする方法の詳細は、第 10 章「リレーショナル表の処理」および第 11 章「ファイルからのデータの読込み」を参照してください。

一時アナリティック・ワークスペースと永続アナリティック・ワークスペース

アナリティック・ワークスペースは、必要に応じて一時的または永続的に保持することができます。特定の計算を実行するためのみにアナリティック・ワークスペースが必要であり、その計算結果をアナリティック・ワークスペースに保存しておく必要がない場合は、セッションの終了時にアナリティック・ワークスペースを廃棄できます。たとえば、アプリケーションで小量の売上データを予測する必要がある場合などに、このような処理が発生します。予測は常に再実行できるため、結果を保存しても意味がありません。

また、アナリティック・ワークスペースを複数のセッションで継続させることもできます。大量のデータ（大規模な予測やモデルの解決結果など）を計算した場合、または非加算的な集計メソッドを使用してデータを集計した場合に、データをアナリティック・ワークスペースに保存する場合があります。

アナリティック・ワークスペースのデータの共有

アナリティック・ワークスペースのデータは、多数のユーザーで共有できます。アナリティック・ワークスペースのデータを共有するには、共有期間中、アナリティック・ワークスペースを保存しておく必要があります。

たとえば、予測の結果をユーザーで共有可能にする場合、ユーザーによるアナリティック・ワークスペースの保存を許可できます。別のユーザーがアプリケーションのセッション中にそのアナリティック・ワークスペースにアタッチすると、他のユーザーの予測を参照することができます。

OLAP Worksheet からアナリティック・ワークスペースへのアクセス

OLAP Worksheet は、次のタスクの実行に使用可能な、Oracle OLAP に対する対話型のインタフェースです。

- アナリティック・ワークスペースへの接続
- 多くの OLAP DML コマンドの実行
- データ・オブジェクトの作成および移入
- DML プログラムの作成、変更、コンパイルおよび実行
- SQL 文の実行

OLAP Worksheet では、「Command Input」ウィンドウおよびプログラムの「Edit」ウィンドウが表示されます。

「Command Input」ウィンドウの一番下の問合せ（入力）ペインにコマンドを入力できます。また、一番上の応答（出力）ペインに結果が表示されます。

OLAP Worksheet を開くと、Oracle OLAP への接続、アナリティック・ワークスペースのオープン、OLAP DML コマンドの実行、プログラムの作成とデバッグ、変更の保存、アナリティック・ワークスペースのクローズおよび接続のクローズを実行できます。

注意： 次の項では、様々なタスクを実行できる各メニュー項目について説明します。ウィンドウの左側のアイコンを使用すると、いくつかのタスクをショートカットで実行できます。

OLAP Worksheet を開く手順

OLAP Worksheet は、Oracle Enterprise Manager またはオペレーティング・システムのコマンドラインのいずれかから開くことができます。

Oracle Enterprise Manager から OLAP Worksheet を開くには、次の手順を実行します。

1. Oracle Enterprise Manager を起動し、データベースへの接続をオープンします。
2. データベース・フォルダを展開します。
3. OLAP を右クリックしてメニューを表示し、「**OLAP Worksheet**」を選択します。

ヒント： OLAP Worksheet を開くことができない場合、システム変数 HOMEDRIVE および HOMEPATH を確認してください。これらの変数は定義する必要がありません。定義されている場合は、有効な値を設定する必要があります。

UNIX 上のコマンドラインから OLAP Worksheet を開くには、次の手順を実行します。

1. コマンドライン・インタフェースを使用して、OLAP Worksheet のインストール・ディレクトリの `bin` サブディレクトリに移動します。
2. `runapp.sh` スクリプトを実行します。

Windows の場合は、インストール時に作成された OLAP Worksheet のアイコンをクリックします。

参照： OLAP Worksheet の使用方法の詳細は、OLAP Worksheet のオンライン・ヘルプを参照してください。

接続の確立

Oracle OLAP への接続を確立するには、次の手順を実行します。

1. OLAP Worksheet のメニュー・バーで、「**Server**」を選択します。
2. 「**Connect**」を選択します。

   「Login to Database」ダイアログ・ボックスが表示されます。
3. 表示された「Login to Database」ダイアログ・ボックスに、有効なデータベース・ユーザー資格証明および接続情報を入力します。

   「Service」ダイアログ・ボックスに、次の形式で Oracle データベースの ID を入力します。

   host:port:SID

   例 : `mycomputer:1521:rel9i`

Oracle は、データベース・ユーザー ID に基づいてデータへのアクセスを制御します。ユーザー ID には、OLAP Worksheet で使用するアナリティック・ワークスペースおよびリレーショナル表へのアクセス権が付与されている必要があります。アクセス権を持っていない場合、アクセスの試行時にエラーが発生します。

コマンドの実行

OLAP Worksheet の「Command Input」ウィンドウで、OLAP DML コマンドおよび SQL 文を実行できます。

「Options」メニューのオプションを選択して、OLAP DML コマンドまたは SQL 文のどちらを実行するかを指定できます。また、コマンドを個別に実行するか、またはバッファに保存して同時に実行するかを指定することもできます。

- OLAP DML コマンドを実行するには、「Options」メニューの「**SQL Off**」を選択します。SQL 文を実行するには、「**SQL On**」を選択します。
- [Enter] を押した直後にコマンドを実行するには、「Options」メニューの「**Execute on Enter**」を選択します。

  コマンドをバッファに保存するには、「**Execute on Enter**」を選択解除します。その後、問合せペインに入力したすべてのコマンドを実行するには、「View」メニューの「**Execute**」を選択します。

コマンドを入力する前に、問合せペインにカーソルを置いてください。長いコマンドを複数の行で入力する場合、現在の行の終わりに接続文字（-）を入力して、次の行にコマンドを続けることができます。

SQL オプションが「ON」の場合、SQL 文を入力して、[Enter] を押すのみです。SQL 文の終わりにセミコロンを付けないでください。エラーが発生します。

OLAP DML プログラムの編集

「Edit」ウィンドウで DML プログラムを開いて、プログラムの内容を追加または変更できます。複数の「Edit」ウィンドウを同時に開くことができますが、オブジェクト定義は一度に 1 つの「Edit」ウィンドウにのみ表示されます。

「Edit」ウィンドウは、プログラムを操作する他に、モデルまたは集計マップの編集にも使用できます。

プログラムを編集するには、次の手順を実行します。

1. 「Command Input」ウィンドウの入力ペインで次のコマンドを入力します。

   ```
   edit object_name
   ```

   ここで、*object_name* は既存の DML プログラム・オブジェクトの名前です。新しいプログラム・オブジェクトを作成するには、DEFINE コマンドを使用します。モデルまたは集計マップを編集する場合は、オブジェクト名の前に MODEL または AGGMAP を入力します。

2. 「Edit」ウィンドウで、プログラムの内容を追加、変更または削除します。
3. 「File」メニューの「**Save**」を選択して、変更を保存します。これによって、UPDATE コマンドが実行され、この時点までのアナリティック・ワークスペース内のすべての変更が更新されることに注意してください。
4. 「File」メニューの「**Quit**」を選択して、「Edit」ウィンドウを閉じます。

> **参照：** DML プログラムの詳細は、第 7 章「プログラムの開発」を参照してください。

接続のクローズ

Oracle OLAP への接続をクローズするには、次の手順を実行します。

1. OLAP Worksheet のメニュー・バーで、「**Server**」を選択します。
2. 「**Disconnect**」を選択します。
3. 切断するかどうかを求められたら、「**Yes**」を選択します。

切断すると、OLAP Worksheet は COMMIT コマンドを実行してからセッションを終了します。UPDATE コマンドを実行するか、または接続前に「Edit」ウィンドウで「File」メニューの「**Save**」を選択した場合は、更新前に加えた変更が永続的になります。それ以外の場合、変更は破棄されます。更新後に加えたすべての変更は切断時に破棄されます。

SQL ベースのアプリケーションからアナリティック・ワークスペースへのアクセス

SQL プログラマは、OLAP 表ファンクションを使用する SQL SELECT 文を使用し、SQL スクリプトに OLAP DML コマンドを埋め込むことによって、アナリティック・ワークスペース内のデータを問い合せることができます。これらの操作の詳細は、『Oracle9*i* OLAP ユーザーズ・ガイド』を参照してください。

SQL SELECT 文の使用

SQL プログラマは、SQL SELECT 文を使用して、アナリティック・ワークスペースのデータを問い合せることができます。

- CWM2_OLAP_AW_ACCESS PL/SQL パッケージを使用してアナリティック・ワークスペースがリレーショナル・スキーマに定義済である場合、アナリティック・ワークスペースのビューが作成されています。これらの SQL ビューに対して SQL SELECT 文を使用して、アナリティック・ワークスペースを問い合せることができます。この方法を行うには、アナリティック・ワークスペース内の基礎となるデータに関する最小限の情報が必要です。
- SQL SELECT 文で OLAP_TABLE ファンクションを使用して、アナリティック・ワークスペースを問い合せることができます。この方法を行うには、アナリティック・ワークスペースのデータに関する詳細な情報が必要です。OLAP_TABLE ファンクションは、Oracle OLAP に付属しています。

埋込み OLAP DML コマンドの使用

SQL プログラマは、DBMS_AW パッケージのプロシージャおよびファンクションを使用して、アナリティック・ワークスペースのデータに対して OLAP DML 文を実行できます。SQL プログラマは、リレーショナル表からアナリティック・ワークスペースにデータを移動し、データの高度な分析（予測など）を行った後、アナリティック・ワークスペースのデータをリレーショナル表に戻すことができます。

Java アプリケーションからアナリティック・ワークスペースへのアクセス

通常、Java アプリケーションは、OLAP API を使用してリレーショナル・データにアクセスします。また、Oracle OLAP API は、アナリティック・ワークスペース内に存在するデータへのアクセスをサポートします。これらの動作の詳細は、『Oracle9*i* OLAP 開発者ガイド - Oracle OLAP API』および OLAP API Javadoc を参照してください。

OLAP メタデータの使用

Java アプリケーションは、OLAP API を介して、OLAP メタデータに情報が登録されているアナリティック・ワークスペース・データにアクセスできます。OLAP メタデータは OLAP API の多次元メタデータ・モデル（MDM）と互換性があるため、Java アプリケーションは、OLAP API の Java クラスを使用してアナリティック・ワークスペース・データを操作できます。データベース管理者がアナリティック・ワークスペース・データの情報を OLAP メタデータに記録する方法の詳細は、『Oracle9*i* OLAP ユーザーズ・ガイド』を参照してください。

埋込み OLAP DML コマンドの使用

OLAP API は、Java アプリケーションを使用してアナリティック・ワークスペースのデータを直接操作する方法も提供しています。この場合、メタデータは不要で、OLAP API のデータ操作クラスを使用する必要もありません。Java アプリケーションは、OLAP API の `SPLExecutor` クラスを使用して、アナリティック・ワークスペースを開いて DML コマンドを直接 Oracle OLAP に送信し、アナリティック・ワークスペースで実行します。

2

アナリティック・ワークスペースの定義および処理

この章では、アナリティック・ワークスペースの作成、アタッチおよび管理について説明します。この章では、次の項目について説明します。

- OLAP DML を使用したアナリティック・ワークスペースの処理
- 複数のアナリティック・ワークスペースのアタッチ
- アナリティック・ワークスペースの名前および別名の使用
- アナリティック・ワークスペースの変更の保存
- プログラムの自動実行
- アナリティック・ワークスペースへのセキュリティの追加
- アナリティック・ワークスペース・オブジェクトのインポートおよびエクスポート
- アナリティック・ワークスペースの情報の取得

OLAP DML を使用したアナリティック・ワークスペースの処理

アナリティック・ワークスペースのデータおよびオブジェクトの定義をセッションで使用可能にするには、そのアナリティック・ワークスペースをアタッチする必要があります。現在アタッチされているアナリティック・ワークスペースは、**アクティブ・アナリティック・ワークスペース**といいます。アナリティック・ワークスペースをアタッチする方法の詳細は、2-3 ページの「アナリティック・ワークスペースをアタッチする方法」を参照してください。

AW コマンドを LIST キーワードとともに使用すると、アクティブ・アナリティック・ワークスペースのリストを表示できます。

```
AW LIST
```

このコマンドでは、アクティブ・アナリティック・ワークスペースのリストが表示されます。express アナリティック・ワークスペースは、内部的に使用されるオブジェクトを含むシステム・アナリティック・ワークスペースであり、アナリティック・ワークスペース・リストに常に表示されます。

現行のアナリティック・ワークスペース

現行のアナリティック・ワークスペースは、AW コマンドに LIST キーワードを指定して表示する、アクティブ・アナリティック・ワークスペース・リスト内の先頭のアナリティック・ワークスペースです。デフォルトでは、新しいアナリティック・ワークスペース・オブジェクトを定義した場合、別のアクティブ・アナリティック・ワークスペースの名前を指定しないかぎり、現行のアナリティック・ワークスペースに格納されます。また、LISTNAMES などのプログラムでは、現行のアナリティック・ワークスペースのオブジェクトのみが表示されます。ただし、アクティブ・アナリティック・ワークスペースが現行のアナリティック・ワークスペースでない場合でも、そのデータの変更と更新、そのプログラムの編集と実行、およびそのオブジェクト定義の変更を行うことができます。

セッションでは、現行のアナリティック・ワークスペースは必要ありません。アナリティック・ワークスペース名を指定せずに Oracle OLAP を起動すると、express アナリティック・ワークスペースがリストの先頭に配置されます。ただし、express アナリティック・ワークスペースは現行のアナリティック・ワークスペースではありません。AW コマンドで指定しないかぎり、現行のアナリティック・ワークスペースは存在しません。

AW ファンクションを NAME キーワードとともに使用すると、現行のアナリティック・ワークスペースの名前を取得できます。

2 つのアナリティック・ワークスペースがアタッチ済であり、一方の名前が marketing、他方の名前が personnel であると想定します。次のコマンドでは、AW ファンクションを NAME キーワードとともに使用して、現行のアナリティック・ワークスペースの名前を MYTEXT という名前の変数に取得し、MYTEXT の値を表示します。この値は、コマンドの後に表示されます。

```
mytext = AW(NAME)
SHOW mytext
```

```
PERSONNEL
```

アナリティック・ワークスペースを作成する方法

AW コマンドを使用すると、新しいアナリティック・ワークスペースを作成できます。次の例では、finance という名前のアナリティック・ワークスペースが作成されます。

```
AW CREATE finance
```

アナリティック・ワークスペースを作成する場合、Oracle OLAP によって COMMIT コマンドが自動的に実行されます。

アナリティック・ワークスペースには、作成者であるユーザーのみがアクセスできます。他のユーザーにアナリティック・ワークスペースの使用を許可する場合は、それらのユーザーにアナリティック・ワークスペースが格納されているリレーショナル表へのアクセス権を付与する必要があります。表の名前は AW$ で始まり、その後に AW CREATE コマンドで指定したアナリティック・ワークスペースの名前が続きます。

他のユーザーに読込み権限を付与するには、SQL で次のようなコマンドを実行します。この例では、アナリティック・ワークスペースの名前は demo、ユーザーの名前は scott です。

```
GRANT SELECT ON aw$demo TO scott
```

他のユーザーに書込み権限を付与するには、次のような SQL コマンドを実行します。

```
GRANT UPDATE ON aw$demo TO scott
```

他の SQL GRANT コマンドの場合と同様に、ユーザーのかわりにグループまたはロールを指定できます。

アナリティック・ワークスペースをアタッチする方法

AW コマンドを使用すると、セッション中にアナリティック・ワークスペースをアタッチおよびデタッチできます。また、セッションでの作業中に、AW コマンドを使用してアクティブ・アナリティック・ワークスペースの切替えを任意に行うことができます。

AW コマンドを ATTACH キーワードとともに使用すると、アナリティック・ワークスペースをアタッチできます。指定したアナリティック・ワークスペースは自動的にアタッチされ、現行のアナリティック・ワークスペースになります。次の例では、finance という名前の既存のアナリティック・ワークスペースがアタッチされ、それが現行のアナリティック・ワークスペースになります。新しいアナリティック・ワークスペースをリストの先頭に追加できるように、アタッチしたアナリティック・ワークスペースはアタッチ済アナリティック・ワークスペースのリストの末尾に移動します。

```
AW ATTACH finance
```

アナリティック・ワークスペースをアタッチする際、デフォルトのアクセス権は読込み専用になっています。別のアタッチ・モードにする場合は、AW コマンドでそれを明示的に指定

する必要があります（2-4 ページの「アナリティック・ワークスペースのアタッチ・モードの指定」を参照）。

注意： アナリティック・ワークスペースのアタッチ時に自動的に実行されるプログラムを作成できます。詳細は、2-11 ページの「プログラムの自動実行」を参照してください。

アナリティック・ワークスペースのアタッチ・モードの指定

AW コマンドの RO、RW または RX キーワードを使用すると、読込み専用モード、読込み / 書込み非排他モードまたは読込み / 書込み排他モードのいずれかで、アナリティック・ワークスペースのアタッチを指定できます。

読込み / 書込み非排他モードまたは読込み専用モードでアタッチしたアナリティック・ワークスペースには、複数のセッションで同時にアクセスできます。ただし、読込み / 書込みアクセスでアナリティック・ワークスペースをオープンできるのは、1 つのセッションのみです。すでに別のユーザーが読込み / 書込みモードでアナリティック・ワークスペースをアタッチしている場合、そのユーザーがデタッチするまで、同じアナリティック・ワークスペースを読込み / 書込みモードでアタッチできません。

読込み / 書込み排他モードでアタッチされたアナリティック・ワークスペースへは、他のセッションからアクセスできません。すでに他の複数のユーザーがアナリティック・ワークスペースをアタッチしている場合、それらのすべてのユーザーがデタッチするまで、同じアナリティック・ワークスペースを読込み / 書込み排他モードでアタッチできません。

アナリティック・ワークスペースの共有

アナリティック・ワークスペースの作成者によってセッションのユーザーにアクセス権が付与されていると想定すると、アナリティック・ワークスペースには、複数のセッションから同時にアクセスできます。複数のセッションで 1 つのアナリティック・ワークスペースにアクセスできますが、任意の時間に読込み / 書込みアクセスでそのアナリティック・ワークスペースをオープンできるのは、1 つのセッションのみです。

アナリティック・ワークスペースをアタッチする際、デフォルトのアクセス権は、読込み専用になっています。Oracle OLAP は、1 人の書込みユーザーと複数の読込みユーザーによるアナリティック・ワークスペースへの同時アクセスをサポートしています。ユーザー ID が適切なアクセス権を持っている場合、アナリティック・ワークスペースを使用しているユーザー数に関係なく、常にアナリティック・ワークスペースに読込み専用でアクセスできます。別のユーザーが読込み / 書込みアクセスを行い、アナリティック・ワークスペースの変更をコミットした場合、アナリティック・ワークスペースのビューは変更されません。変更を確認するには、アナリティック・ワークスペースをデタッチして再度アタッチする必要があります。

読込み / 書込みアクセスを行うには、AW コマンドでそれを明示的に指定する必要があります。アナリティック・ワークスペースが別のセッションで読込み / 書込みモードでアタッチ

された場合、アクセスの要求に対する応答は、AW コマンドで使用するキーワードによって異なります。

AW コマンドの WAIT または NOWAIT キーワードを使用すると、要求するアクセスのタイプでアナリティック・ワークスペースが使用可能になるまで待機するかどうかを指定できます。

- NOWAIT キーワード（デフォルト）を指定し、要求しているアクセスのタイプではアナリティック・ワークスペースが使用可能でない場合、アナリティック・ワークスペースが使用不可であることを示すエラー・メッセージが生成されます。
- WAIT キーワードを指定し、要求しているアクセスのタイプではアナリティック・ワークスペースが使用可能でない場合、Oracle OLAP によってユーザーがアナリティック・ワークスペースの待機リストに入れられます。

アナリティック・ワークスペースをデタッチする方法

アナリティック・ワークスペースをデタッチするには、DETACH キーワードとともに AW コマンドを使用します。次のコマンドを実行すると、finance アナリティック・ワークスペースがデタッチされます。

```
AW DETACH finance
```

デタッチされたアナリティック・ワークスペースは、データベース内に残ります。ただし、セッションでアクセスすることはできません。デタッチされたアナリティック・ワークスペースに再度アクセスするには、AW コマンドを ATTACH キーワードとともに使用します。

アナリティック・ワークスペースを削除する方法

アナリティック・ワークスペースをデータベースから削除するには、AW コマンドを DELETE キーワードとともに使用します。削除を行う前に、アナリティック・ワークスペースをデタッチする必要があります。次のコマンドを実行すると、finance アナリティック・ワークスペースが削除されます。

```
AW DETACH finance
AW DELETE finance
```

削除されたアナリティック・ワークスペースは、データベースに残りません。このアナリティック・ワークスペースに再度アクセスすることはできません。アナリティック・ワークスペースをデータベースから削除する場合、Oracle OLAP によって COMMIT コマンドが自動的に実行されます。

アナリティック・ワークスペースのローカライゼーション設定

Oracle は、キャラクタ・セット、日付書式、通貨記号およびその他の言語固有の特性が異なるロケールをサポートしています。Oracle グローバリゼーション・サポートは、「NLS」で始まるパラメータの値に基づきます。NLS パラメータの詳細は、『Oracle9*i* SQL リファレン

ス』および『Oracle9*i* Database グローバリゼーション・サポート・ガイド』を参照してください。

セッション中に、「NLS」で始まる OLAP DML オプションを使用して設定することによって、いくつかの NLS パラメータの値を動的に変更できます。たとえば、OLAP DML で NLS_LANG または NLS_TERRITORY の値を設定できます。OLAP DML の NLS オプションの値を設定すると、その設定はアナリティック・ワークスペース内での作業のみでなく、データベース・セッション全体に影響を及ぼします。

また、次の SQL コマンドを使用して、Oracle OLAP を含むセッション全体に対する NLS パラメータを変更できます。

```
ALTER SESSION SET parameter = value
```

OLAP DML の NLS オプションの詳細は、Oracle9*i* OLAP DML Referece ヘルプを参照してください。

複数のアナリティック・ワークスペースのアタッチ

一度に複数のアナリティック・ワークスペースをアタッチできます。ただし、複数のアナリティック・ワークスペースを処理する場合は、オブジェクトの名前の指定時に注意が必要です。DESCRIBE コマンドを使用するか、コマンドまたはプログラムでそれを参照してオブジェクトを名前で要求する場合、指定したオブジェクトが検出されるまで、すべてのアクティブ・アナリティック・ワークスペースが検索されます。複数のアナリティック・ワークスペースを同時に使用する場合、オブジェクトの参照時に修飾オブジェクト名を使用できる場合を除き、異なるアナリティック・ワークスペースのオブジェクトに同じ名前を指定しないでください。

修飾オブジェクト名

複数のアナリティック・ワークスペースをアタッチする際に、複数のアナリティック・ワークスペースのオブジェクト名が重複している場合は、修飾オブジェクト名を使用して、参照するオブジェクトを指定する必要があります。

修飾オブジェクト名は、アナリティック・ワークスペースの名前を含めることによって、オブジェクトを一意に識別します。修飾オブジェクト名を使用すると、アクセスするオブジェクト（およびそのオブジェクトが存在するアナリティック・ワークスペース）を Oracle OLAP に明確に認識させることができます。

たとえば、SALES という名前の変数を持つ NORTHEAST アナリティック・ワークスペースをアタッチしており、同様に SALES という名前の変数を持つ SOUTHEAST アナリティック・ワークスペースをアタッチしている場合は、次の修飾オブジェクト名（QON）を使用してこれらの変数を指定する必要があります。

```
northeast!sales
southeast!sales
```

QON の最初の部分はアナリティック・ワークスペースの名前、2 番目の部分はオブジェクトの名前です。2 つの部分は感嘆符（!）で結合されます。

修飾名と非修飾名を混在させて使用することができます。修飾名は、アタッチした別のアナリティック・ワークスペースに同じ名前を持つオブジェクトが存在する際に、1 つのアナリティック・ワークスペースに存在する特定のオブジェクトを指定する場合に使用する必要があります。いずれのオブジェクトに対しても QON を指定しない場合、Oracle OLAP によっていずれかのオブジェクトが使用されるため、結果は不特定になります。

複数の AUTOGO プログラムおよび権限プログラム

現在アタッチしているアナリティック・ワークスペースに AUTOGO プログラムまたは権限プログラムを定義している場合、そのプログラムが実行されます。ただし、現在アタッチしている複数のアナリティック・ワークスペースにアナリティック・ワークスペースの権限プログラムを定義している場合、それらのプログラムを編集したり、別の用途で使用する際には、修飾オブジェクト名を使用する必要があります。これによって、適切なバージョンのプログラムにアクセスできます

参照： AUTOGO プログラムの詳細は、2-11 ページの「プログラムの自動実行」を参照してください。権限プログラムの詳細は、2-12 ページの「アナリティック・ワークスペースへのセキュリティの追加」を参照してください。

アナリティック・ワークスペースの名前および別名の使用

OLAP DML では、明確で柔軟なコードを作成できるように、アナリティック・ワークスペースを参照するための複数の方法が提供されています。

アナリティック・ワークスペースの名前

アナリティック・ワークスペースの名前は、AW CREATE コマンドを使用してアナリティック・ワークスペースを作成する際に割り当てます。たとえば、aw create demo というコマンドを実行すると、アナリティック・ワークスペースの名前が DEMO になります。

デフォルトでは、アナリティック・ワークスペースは、データベース・ユーザー ID のスキーマに作成されます。たとえば、ユーザー SCOTT が DEMO アナリティック・ワークスペースを作成した場合、アナリティック・ワークスペースのフルネームは SCOTT.DEMO になります。他のユーザーのスキーマに存在するアナリティック・ワークスペースへのアクセス権を持っている場合は、アナリティック・ワークスペースをアタッチする際にフルネームを指定できます。たとえば、SCOTT は、次のコマンドを使用して、SUSAN が所有するスキーマに存在する REPORTS という名前のアナリティック・ワークスペースをアタッチできます。

```
aw attach susan.reports
```

ほぼすべての DML コマンドで、アナリティック・ワークスペースのフルネーム（SCOTT.DEMO など）を指定できます。アナリティック・ワークスペースが自分のスキーマに存在する場合は、名前のみ（DEMO など）を指定できます。オプションで、割り当てられたアナリティック・ワークスペースの別名を使用してアナリティック・ワークスペースを参照できます。

アナリティック・ワークスペースの別名

アナリティック・ワークスペースの別名は、アタッチ済のアナリティック・ワークスペースの代替名です。AW ALIASLIST コマンドを使用すると、別名の割当てまたは削除を行うことができます。

別名は、その別名が割り当てられてから、アナリティック・ワークスペースがデタッチされるまで（または別名が削除されるまで）有効になります。そのため、アタッチされていないアナリティック・ワークスペースをアタッチするたびに、別名を再度割り当てる必要があります。

別名を割り当てる理由の 1 つは、他のユーザーのスキーマに含まれるアナリティック・ワークスペースを簡単に参照できるようにするためです。たとえば、他のユーザーのスキーマに含まれるアナリティック・ワークスペースを参照する修飾オブジェクト名およびコマンドに別名を使用できます。別名を割り当てる別の理由は、アナリティック・ワークスペースへの参照を含むがその名前をハードコードしない汎用コードを記述するためです。一般参照を提供する別名を使用すると、別名を割り当て、異なる時点で異なるアナリティック・ワークスペースに対してコードを実行できます。

アナリティック・ワークスペースの変更の保存

通常、セッションの終了時にアナリティック・ワークスペースを保存して、セッション中に加えた変更を保存します。セッション中にアナリティック・ワークスペースを定期的に保存し、処理の経過に沿って変更を保存することもできます。

アナリティック・ワークスペースに読込み / 書込みアクセスを行っている場合は、変更を保存できます。アナリティック・ワークスペースに読込み専用アクセスを行っている場合は、アナリティック・ワークスペースは変更できますが、これらの変更は保存できません。

アナリティック・ワークスペースへの変更を保存するには、2 つのコマンド UPDATE および COMMIT を同時に使用します。

UPDATE コマンド

UPDATE コマンドを実行すると、アナリティック・ワークスペースの変更が一時領域からアナリティック・ワークスペースが格納されているデータベース表に移動されます。OLAP DML または SQL のいずれかで COMMIT コマンドを実行するまで、変更は保存されません。

COMMIT コマンドを実行する際に、アナリティック・ワークスペースでの変更をコミットする場合、最初に UPDATE コマンドを使用して、アナリティック・ワークスペースを更新する必要があります。表に移動されていない変更は、コミットされません。

UPDATE コマンドの単純な構文は次のとおりです。

```
UPDATE [awname1 [awname2 . . .]]
```

awname 引数には、セッションにアタッチされた読込み / 書込みアナリティック・ワークスペースの名前を指定します。アナリティック・ワークスペース名を指定しない場合、アタッチ済のすべての読込み / 書込みアナリティック・ワークスペースが更新されます。

たとえば、次のコマンドを発行すると、アタッチ済のすべてのアナリティック・ワークスペースでの変更を、一時領域からアナリティック・ワークスペースが格納されているデータベース表に移動できます。

```
UPDATE
```

COMMIT コマンド

COMMIT コマンドを実行すると、SQL COMMIT コマンドが実行されます。セッション中に Oracle OLAP またはその他のアクセス方法（SQL など）で加えられたデータベースへのすべての変更がコミットされます。

たとえば、次の 2 つのコマンドを発行すると、データベースでのすべてのアナリティック・ワークスペースの変更を保存できます。

```
UPDATE
COMMIT
```

多くのユーザーは、SQL*Plus または OLAP Worksheet を使用して DML コマンドを実行します。これらのツールでは、いずれもセッションの終了時に COMMIT コマンドが自動的に実行されます。ただし、アナリティック・ワークスペースの変更を保存するには、最初に UPDATE コマンドを実行する必要があります。

共有アナリティック・ワークスペースをアタッチし、別のユーザーが読込み / 書込みアクセスを行った場合、そのユーザーが実行した COMMIT コマンドは、アナリティック・ワークスペースのビューには影響を及ぼしません。データのビューは、アナリティック・ワークスペースのアタッチ時と同じです。変更されたデータにアクセスする場合は、アナリティック・ワークスペースをデタッチしてから再度アタッチする必要があります。

ROLLBACK コマンドの影響

OLAP DML では、ROLLBACK コマンドを発行する方法は提供されません。ただし、Oracle OLAP 以外のセッションでは（たとえば、PL/SQL を介して）このコマンドを実行できます。セッション中に ROLLBACK コマンドが実行されると、Oracle OLAP によって、アタッチ済のアナリティック・ワークスペースにコミットされていない更新が存在するかどうかが確認されます。

- コミットされていない更新が存在する場合（変更を加えて UPDATE コマンドを実行しているが、COMMIT コマンドを実行していない場合）、Oracle OLAP によって変更が廃棄され、アナリティック・ワークスペースがデタッチされます。
- コミットされていない更新が存在しない場合、ROLLBACK コマンドを実行しても Oracle OLAP による操作は行われません。これは、最後に COMMIT コマンドを発行した後に UPDATE コマンドを発行していない場合、Oracle OLAP による操作は行われず、セッション中のすべての変更がアナリティック・ワークスペースに残ることを意味します。

セーブポイント以降に実行されたコミットされていない更新が存在する場合にそのセーブポイントにロールバックすると、Oracle OLAP によってこれらの更新が廃棄され、アナリティック・ワークスペースがデタッチされます。セーブポイント以前に実行されたコミットされていない更新はアナリティック・ワークスペースに残り、同じセッションでアナリティック・ワークスペースを再度アタッチした場合にこれらの更新を参照できます。

アナリティック・ワークスペースの拡張の最小化

アナリティック・ワークスペースを排他的にアタッチした際、頻繁に更新することによって、アナリティック・ワークスペースの拡張を最小限にできます。アナリティック・ワークスペース内のすべてのオブジェクトをエクスポートし、新しいアナリティック・ワークスペースにインポートすることによって、アナリティック・ワークスペース・ファイルを再編成できます。新しいアナリティック・ワークスペースは、大幅に小さくなる場合があります。

アナリティック・ワークスペース・オブジェクトのエクスポートおよびインポートによってアナリティック・ワークスペースを再編成するには、次の手順を実行します。

1. 元のアナリティック・ワークスペースに対して ALLSTAT コマンドを発行します。
2. EXPORT コマンドを ALL キーワードとともに使用して、元のアナリティック・ワークスペースのすべてのデータを EIF ファイルに格納します。
3. 元のアナリティック・ワークスペースと異なる名前で、新しいアナリティック・ワークスペースを作成します。
4. IMPORT コマンドを使用して、EIF ファイルを新しいアナリティック・ワークスペースにインポートします。
5. UPDATE および COMMIT コマンドを使用して、新しいアナリティック・ワークスペースを保存します。
6. オブジェクトが新しいアナリティック・ワークスペースに正しく移動されたことを確認してから、元のアナリティック・ワークスペースを削除します。

指定されたアナリティック・ワークスペースを参照するプログラムは、別名によってアナリティック・ワークスペースを参照できます。この方法では、異なる名前を持つアナリティック・ワークスペースに何回でもインポートすることができます。インポートするたびに、該当するアナリティック・ワークスペースに別名を割り当てることができます。

プログラムの自動実行

アナリティック・ワークスペースのアタッチ時に自動的に実行されるプログラムを作成できます。AW ATTACH コマンドを AUTOGO キーワードとともに使用してアナリティック・ワークスペースをアタッチすると、アナリティック・ワークスペースで AUTOGO という名前のプログラムが検索されます。このプログラムが存在する場合、コマンドが受け入れられる前にそのプログラムが実行されます。AUTOGO キーワードを指定しない場合、または NOAUTOGO キーワードを指定した場合、プログラムは自動的に実行されません。

プログラムの名前

現在アタッチされている複数のアナリティック・ワークスペースに AUTOGO という名前のプログラムが存在する場合（および同じ名前を持つ複数のプログラムが存在する場合）、それらのプログラム編集する際には、修飾オブジェクト名を使用して、適切なプログラムにアクセスしていることを確認する必要があります。

AUTOGO キーワードを指定する際にプログラムが自動的に実行されるようにするには、プログラム名を AUTOGO に指定する必要はありません。かわりに、実行するプログラムの名前を AUTOGO キーワードで指定することができます。アナリティック・ワークスペースに AUTOGO という名前のプログラムが存在する場合でも、Oracle OLAP では AUTOGO キーワードで指定したプログラムが実行されます。

AUTOGO プログラムの例

売上データ用の 2 つのアナリティック・ワークスペース（支出と収入に 1 つずつ）が存在すると想定します。また、データを分析するプログラムを含む analysis という名前の 3 つ目のアナリティック・ワークスペースが存在するとします。analysis アナリティック・ワークスペースでは、AUTOGO プログラムを使用できます。このプログラムには、他の 2 つのアナリティック・ワークスペースをアタッチするための、次に示す 2 行のコードが含まれています。

```
AW ATTACH expense AFTER analysis
AW ATTACH revenue AFTER analysis
```

次のコマンドを使用して analysis アナリティック・ワークスペースをアタッチする場合、このアナリティック・ワークスペースの AUTOGO プログラムは自動的に実行され、他の 2 つのアナリティック・ワークスペースがアタッチされます。

```
AW ATTACH analysis AUTOGO
```

プログラムの名前を AUTOGO ではなく ATTACHDATA に指定した場合、次のコマンドを使用して analysis アナリティック・ワークスペースをアタッチします。

```
AW ATTACH analysis AUTOGO attachdata
```

権限プログラムは AUTOGO プログラムの実行前に実行されることに注意してください。

参照：

- 権限プログラムの詳細は、2-12 ページの「アナリティック・ワークスペースへのセキュリティの追加」を参照してください。
- プログラムを作成する方法の詳細は、第 7 章「プログラムの開発」を参照してください。

アナリティック・ワークスペースへのセキュリティの追加

エンティティとしてのアナリティック・ワークスペースは、データベースに組み込まれたすべてのセキュリティ機能によって保護されます。さらに、権限プログラムを使用して、特定のアナリティック・ワークスペース・オブジェクトまたはアナリティック・ワークスペース全体へのアクセスを制限できます。

権限プログラム

ユーザーがアナリティック・ワークスペースをアタッチする場合、permit_read および permit_write という名前の権限プログラムがアナリティック・ワークスペースに含まれているかどうかが確認されます。これらのプログラムを作成する必要はありません。これらのプログラムが存在する場合、ユーザーがアナリティック・ワークスペースをアタッチする際に、適切なプログラムが自動的に実行されます。

ユーザーがアナリティック・ワークスペースをアタッチする際のアクセス方法	実行されるプログラム（存在する場合）
読込み専用アクセス	permit_read プログラム
読込み / 書込みアクセス	permit_write プログラム

権限プログラムは、AUTOGO プログラムの実行前に実行されます。ユーザーがアナリティック・ワークスペースのアタッチ時にパスワードを指定した場合、パスワードが引数として権限プログラムに渡され、処理されます。権限プログラムは、指定されたパスワードに基づいて、アナリティック・ワークスペース全体または個々のオブジェクトへのアクセス権を付与したり、アクセスを制限することができます。たとえば、次の AW コマンドを実行すると、パスワード goldfinch で、sales アナリティック・ワークスペースがアタッチされます。

```
AW ATTACH sales PASSWORD goldfinch
```

権限プログラムの作成および設計

権限プログラムを作成するには、permit_read および permit_write という名前の 2 つのプログラムを定義します。これらのプログラムでは、PERMIT コマンドを指定して、個々のアナリティック・ワークスペース・オブジェクトへのアクセス権を付与したり、アクセスを制限することができます。また、これらのプログラムはブール値を戻すユーザー定義ファ

ンクションとしても作成できます。戻り値は、ユーザーがアナリティック・ワークスペースをアタッチする権限を持つかどうかを Oracle OLAP に示します。

プログラムからの戻り値	アナリティック・ワークスペースをアタッチ可能かどうか
YES	アタッチ可能
NO	アタッチ不可

権限プログラムを使用すると、このプログラムが格納されているアナリティック・ワークスペースへの 2 つのレベルのアクセスを制御できます。

アクセスのタイプ	説明
アナリティック・ワークスペース・レベル	アナリティック・ワークスペース全体に対するアクセス権が付与されるかどうかは、権限プログラムの戻り値によって異なります。
オブジェクト・レベル	特定のオブジェクトまたは一連のオブジェクト値へのアクセス権が付与されるかどうかは、権限プログラムの PERMIT コマンドによって異なります。 指定した権限プログラムで参照されるすべてのオブジェクトは、同じアナリティック・ワークスペースに存在する必要があります。

たとえば、PERMIT コマンドを使用すると、あるユーザー・グループによる salary 変数へのアクセスを拒否したり、別のユーザー・グループによる tenure 変数へのアクセスを拒否することができます。特定のユーザーが、salary 変数のセルのサブセットにアクセスできないように指定することもできます。現在アタッチされている複数のアナリティック・ワークスペースに権限プログラムが存在する場合（および同じ名前を持つ複数のプログラムが存在する場合）、それらのプログラムを編集する際には、修飾オブジェクト名を使用して、適切なプログラムにアクセスしていることを確認する必要があります。

参照： プログラムを作成する方法の詳細は、第 7 章「プログラムの開発」を参照してください。

アナリティック・ワークスペース・オブジェクトのインポートおよびエクスポート

アナリティック・ワークスペース全体、複数のアナリティック・ワークスペース・オブジェクト、1 つのアナリティック・ワークスペース・オブジェクトまたはアナリティック・ワークスペース・オブジェクトの一部を固有フォーマットの EIF ファイルにエクスポートできます。その後、同じ Oracle データベースまたは異なる Oracle データベース内の異なるアナリティック・ワークスペースにその情報をインポートできます。

エクスポートおよびインポートを行う目的の 1 つは、データを新しい場所に移動することです。別の目的は、多数のオブジェクトまたはディメンション値を追加して削除した後に、アナリティック・ワークスペースから不要な領域を削除することです。これを行うには、EXPORT コマンドを使用してすべてのデータを EIF ファイルに格納し、異なる名前を持つ別のアナリティック・ワークスペースを作成し、次に IMPORT コマンドを使用して EIF ファイルを新しいアナリティック・ワークスペースにインポートします。同じデータベースにインポートした場合、古いアナリティック・ワークスペースを削除し、古いアナリティック・ワークスペースに使用していた別名と同じ別名で新しいアナリティック・ワークスペースを参照できます。

次のコマンドを実行すると、すべてのデータおよび定義が現行のアナリティック・ワークスペースから mydir という名前のディレクトリ別名に存在する reorg.eif という名前の EIF ファイルにコピーされます。

```
export all to eif file 'mydir/reorg.eif'
```

ディレクトリ別名は、データベースで定義され、ディレクトリへのアクセスを制御します。CDA コマンドを使用すると、カレント・ディレクトリ別名を指定できます。この場合、Oracle OLAP はファイルをカレント・ディレクトリ別名で作成する必要があると想定するため、EXPORT コマンドでディレクトリ別名を指定する必要はありません。データベース・ユーザー名でファイルの読込みおよび書込みを実行できるディレクトリ別名に対するアクセス権については、Oracle DBA に連絡してください。

次のコマンドを実行すると、データおよび定義が EIF ファイルから新しいアナリティック・ワークスペースにコピーされます。

```
aw create new
import all from eif file 'reorg.eif'
update
commit
```

アナリティック・ワークスペースの情報の取得

AWDESCRIBE プログラムを使用すると、次の情報を含むアナリティック・ワークスペースの完全な説明を表示できます。

- 最後に更新された日時、各タイプのアナリティック・ワークスペース・オブジェクトの数など、アナリティック・ワークスペースの一般情報を示す目次。
- アルファベット順にソートされたアナリティック・ワークスペース・オブジェクトのリスト。
- オブジェクトのタイプによってソートされ、各タイプ内でアルファベット順に名前がソートされた、すべてのアナリティック・ワークスペース・オブジェクトの詳細な説明。各オブジェクトには、このオブジェクトを使用したり、またはそれが使用する他のオブジェクトのクロス・リファレンスのリストが存在します。

通常、AWDESCRIBE からの出力は非常に長いため、OUTFILE コマンドを使用してこの出力をファイルに直接送ることもできます。

```
OUTFILE 'diralias/filename'
AWDESCRIBE
OUTFILE EOF
```

diralias は、ディレクトリ別名の名前です。*filename* は、情報が書き込まれるファイルの名前です。

読込み / 書込み権限を持つディレクトリ別名の名前については、DBA に連絡してください。

アナリティック・ワークスペースの一般情報の取得

AW ファンクションを使用すると、アタッチ済のアナリティック・ワークスペースに関する様々な情報を戻すことができます。たとえば、AW ファンクションを使用して、現行のアナリティック・ワークスペースの名前、またはこのアナリティック・ワークスペースへの読込み / 書込み権限を持っているかどうかを確認できます。

AW ファンクションの単純な構文は次のとおりです。

```
AW(choice [workspace])
```

choice に指定するキーワードによって、AW ファンクションから戻される情報のタイプが決まります。キーワードには、ATTACHED、NAME、RO、RW などを指定できます。

たとえば、次のコマンドを実行すると、アクティブ・アナリティック・ワークスペースが確認されるため、プログラムは適切なデータを選択してレポートできます。

```
IF AW(NAME) EQ 'mysales'
     THEN REPORT sales.m
     ELSE REPORT gensales
```

アナリティック・ワークスペースのオブジェクトの表示

LISTNAMES プログラムを使用すると、アナリティック・ワークスペース内のオブジェクトのリストを取得できます。このプログラムは、アナリティック・ワークスペース内のすべてのオブジェクトを、オブジェクトのタイプによってグループ化し、アルファベット順にソートしたリストで表示します。LISTNAMES は、各タイプのオブジェクト（ディメンションや変数など）の総数を示します。

任意のディメンションによってディメンション化されたすべてのオブジェクト、または任意のディメンションに関連するすべてのオブジェクトのリストを取得するには、LISTBY コマンドを使用します。

たとえば、month によってディメンション化されている、またはこれに関連するオブジェクトを検索するには、次のコマンドを使用します。

```
LISTBY month
```

次のリストが表示されます。

```
14 objects dimensioned by or related to MONTH in DEMO
-------------------------------------------------------------
ACTUAL          ADVERTISING        BUDGET
EXPENSE         FCST               NATIONAL.SALES
PRICE           PRODUCT.MEMO       SALES
SALES.FORECAST  SALES.PLAN         SHARE
UNITS           UNITS.M
```

1 つ以上のオブジェクトの定義を表示するには、DESCRIBE コマンドを使用します。たとえば、次のコマンドを発行できます。

```
DESCRIBE price
```

このコマンドによって生成される出力は次のとおりです。

```
DEFINE PRICE VARIABLE DECIMAL <MONTH PRODUCT>
LD Wholesale Unit Selling Price
```

オブジェクト名を指定せずに DESCRIBE コマンドを実行すると、NAME ディメンションの現行のステータス・リストに含まれるすべてのオブジェクトが表示されます。NAME ディメンションには、アナリティック・ワークスペースに定義されたすべてのオブジェクトの名前が含まれています。

REPORT コマンドを実行すると、変数、ディメンション、リレーションなど、多数のアナリティック・ワークスペース・オブジェクトの値を表示できます。たとえば、次のコマンドを実行すると、costs という名前の変数の値が表示されます。

```
report costs
```

このコマンドによって生成される出力は次のとおりです。

```
                      -----------------------COSTS-----------------------------
                      ---------------------GEOGRAPHY---------------------------

DIVISION     EAST       WEST      BOSTON   SAN FRANCISCO      SEATTLE
--------- ---------- ---------- ---------- -------------    ----------
DIVA       27,600.00  50,000.00  27,600.00  10,000.00       40,000.00
DIVB       30,000.00  62,000.00  30,000.00  12,000.00       50,000.00
```

オブジェクトの情報の取得

アナリティック・ワークスペース・オブジェクトの情報を取得するには、OBJ ファンクションを使用します。

たとえば、次のコマンドを使用すると、units 変数のディメンション数が取得されます。この出力は、コマンドの次に示します。

```
SHOW OBJ(NUMDIMS 'units')
3
```

次のコマンドを実行すると、units 変数のデータ型が取得されます。この出力は、コマンドの次に示します。

```
show obj(data 'units')
INTEGER
```

多くの場合、OBJ ファンクションを LIMIT コマンドおよび NAME ディメンションとともに使用して、オブジェクト・グループの情報を取得します。LIMIT コマンドでは、ディメンションのステータスを設定します。これによって、ディメンション値へのアクセスを制限し、このディメンション値によってディメンション化された任意の変数またはリレーションに対応する制限を設定します。

LIMIT コマンドを OBJ ファンクションとともに使用すると、特定の特性を持つオブジェクト・グループを識別できます。その後、STATUS コマンドを使用して、そのグループのオブジェクトのリストを表示できます。

次のコマンドを実行すると、month と product の両方によってディメンション化されているオブジェクトのリストが表示されます。

```
LIMIT NAME TO OBJ(ISBY 'month') AND OBJ(ISBY 'product')
STATUS NAME
```

これらのコマンドの出力は次のとおりです。

```
The current status of NAME is:
ADVERTISING, EXPENSE, NATIONAL.SALES, PRICE, PRODUCT.MEMO, SALES, SALES.FORECAST,
SALES.PLAN, SHARE, UNITS, UNITS.M
```

> **参照：** LIMIT コマンドの使用方法の詳細は、第 6 章「データの選択」を参照してください。

3

データ・オブジェクトの定義

この章では、多次元データ構造の概要を示します。オブジェクトを定義する方法およびこれらのオブジェクトの定義を変更する方法について説明します。この章では、次の項目について説明します。

アナリティック・ワークスペース・オブジェクトの定義の概要

オブジェクトの定義とオブジェクトのデータの区別について理解することは重要です。オブジェクト定義とは、アナリティック・ワークスペースに含まれるオブジェクトの説明です。オブジェクトのデータとは、その定義に関連付けられた値です。すべてのオブジェクトは定義を持ちます。ただし、すべてのオブジェクトがデータを持つわけではありません。

たとえば、month、product および district によってディメンション化されている sales 変数には、変数オブジェクトとしてのこの変数の定義が存在します。sales 変数は、3 つのディメンションの定義とも関連付けられます。ただし、sales、month、product および district の値は、定義には含まれません。

プログラム、フォーミュラなどの他のオブジェクトには、データは含まれません。

アナリティック・ワークスペースの作成後、アナリティック・ワークスペース・オブジェクトの定義を開始できます。任意の OLAP DML オブジェクトを定義するには、DEFINE コマンドを使用します。DEFINE コマンドの単純な構文は次のとおりです。

```
DEFINE name object-type attributes
```

name 引数には、新しい定義の名前を指定します。

注意： 各アナリティック・ワークスペースには固有のアナリティック・ワークスペース・オブジェクトのリストが存在するため、複数のアナリティック・ワークスペースに同じ名前を持つオブジェクトを定義できます。ただし、予期しない結果を回避するため、修飾オブジェクト名を使用する場合を除き、同時にアクティブになる別々のアナリティック・ワークスペースのオブジェクトには一意の名前を指定する必要があります（第 2 章「アナリティック・ワークスペースの定義および処理」を参照）。

---

object-type 引数には、定義するオブジェクトのタイプを指定します。デフォルトは VARIABLE です。任意の有効なオブジェクト・タイプを指定できます（3-3 ページの「定義可能なアナリティック・ワークスペース・オブジェクト」を参照）。

attributes 引数には、オブジェクトのプロパティを指定します。属性は、オブジェクトのタイプによって異なります。属性は、各オブジェクト・タイプのエントリに示されます。

定義可能なアナリティック・ワークスペース・オブジェクト

次の表に、DEFINE コマンドを使用して定義する OLAP DML データ・オブジェクト・タイプの概要を示します。

オブジェクト・タイプ	説明
DIMENSION	データのカテゴリを提供する値のリストを含みます。ディメンションは、変数の値を識別する索引として使用します。ディメンションは、リレーショナル・データベースのキーに類似しています。
RELATION	任意のディメンション値を、アナリティック・ワークスペースの同じディメンションの値または別のディメンションの値に関連付けます。リレーションは、リレーショナル・データベースの外部キーに類似しています。
VARIABLE	データを格納します。変数のデータ型は、その変数に含まれるデータの型を示します。変数は、リレーショナル・データベースの表に類似しています。
COMPOSITE	ディメンション値の組合せの名前付きリストです。任意の組合せには、複合ディメンションのベースとなる各ディメンションからの 1 つの値が含まれます。 **注意 :** いくつかのディメンションを疎密として指定して変数を定義すると、名前なし複合ディメンションが自動的に作成されます。名前なし複合ディメンションは内部オブジェクトであるため、OLAP DML オブジェクトとはみなされません。
SURROGATE	単純なディメンションの値のサロゲートとして使用される値のリストを含みます。サロゲートは、LIMIT コマンド、モデル、修飾データ参照およびデータのロードでディメンションのかわりに使用できます。
FORMULA	値を生成する格納済の計算、式またはプロシージャを表します。フォーミュラは、リレーショナル・データベースのビューに類似しています。
MODEL	データを計算して変数またはディメンション値に割り当てる場合に使用する、相互に関連した一連の方程式を含みます。通常、モデルは、財務データを処理する場合に使用します。
PROGRAM	一連の OLAP DML コマンドを含みます。プログラムによって、関連した一連のコマンドが実行されます。プログラムは、SQL のストアド・プロシージャに類似しています。
VALUESET	特定のディメンションのディメンション値のリストを含みます。
AGGMAP	集計マップを作成します。集計マップには、集計またはアロケーションを行う必要がある変数のデータおよび操作の実行方法を指定するコマンドが含まれます。AGGMAP コマンドでは、AGGREGATE コマンドによって使用されるコマンドを指定できます。ALLOCMAP コマンドでは、ALLOCATE コマンドによって使用されるコマンドを指定できます。

データ型

アナリティック・ワークスペースのデータ型は、基本データ型という複数のカテゴリに分類されます。次の表に、これらの基本データ型を示します。

基本データ型	固有のデータ型
数値	INTEGER、SHORTINTEGER、LONGINTEGER、DECIMAL、SHORTDECIMAL、NUMBER
テキスト	TEXT、NTEXT、ID
ブール	BOOLEAN
日付	DATETIME、DATE

異なるオブジェクトでは、その値に異なるデータ型を使用できます。

- 変数に格納されるデータ値のような、ほとんどのデータ値では、INTEGER、SHORTINTEGER、DECIMAL、SHORTDECIMAL、NUMBER、TEXT、ID、NTEXT、BOOLEAN、DATETIME および DATE データ型がサポートされています。
- ディメンション値では、INTEGER、NUMBER、TEXT、ID および NTEXT データ型がサポートされています

数値データ型

次の数値データ型がサポートされています。

データ型	データ値
INTEGER	(-2**31) ～ (2**31)-1 の整数
SHORTINTEGER	(-2**15) ～ (2**15)-1 の整数
LONGINTEGER	(-2**63) ～ (2**63)-1 の整数
DECIMAL	小数（最大 15 桁の有効数字）
SHORTDECIMAL	小数（最大 7 桁の有効数字）
NUMBER	小数（最大 38 桁の有効数字）

データを入力する際、これらのいずれのデータ型でも、値はプラス記号（+）またはマイナス記号（-）で始めることができ、カンマを含めることはできません。また、小数値には小数点を含めることができます。データの表示では、桁区切りおよび小数点は、NLS_NUMERIC_CHARACTERS オプションによって制御されます。

アナリティック・ワークスペースの NUMBER データ型は、データベースの NUMBER データ型と完全な互換性があります。このデータ型は、テキスト・データ型または整数データ型が適切でない場合に、ディメンションおよびサロゲートに使用されます。通常、計算（予測、集計など）に使用されない変数に割り当てられ、データベースの丸め動作に一致する必要があるか、または高い精度を必要とする変数に使用されます。変数に NUMBER データ型を割り当てるかどうかを判断する場合、パフォーマンスを最適化するために次のことに注意してください。

- アナリティック・ワークスペースでの NUMBER 変数の計算は、DECIMAL などの他の数値型での計算より低速になります。
- アナリティック・ワークスペースから NUMBER データ型が含まれるリレーショナル列にデータがフェッチされる場合、アナリティック・ワークスペースでデータにすでに NUMBER データ型が含まれている場合には変換手順が必要ないため、パフォーマンスが最適化されます。

リテラル数値の例

リテラル数値の例を次に示します。

```
-1
256000
+2147483647
10000000000.0009
```

テキスト・データ型

次のテキスト・データ型がサポートされています。

データ型	データ値
TEXT	データベース・キャラクタ・セットで 1 行に最大 4000 バイト。このデータ型は、データベースの CHAR データ型および VARCHAR2 データ型に相当します。
NTEXT	UTF-8 キャラクタ・コードで 1 行に最大 4000 バイト。このデータ型は、データベースの NCHAR データ型および NVARCHAR2 データ型に相当します。
ID	データベース・キャラクタ・セットで 1 行に最大 8 文字。

データを入力する際、テキスト・リテラルは一重引用符で囲む必要があります。一重引用符で囲まれていない場合、OLAP DML コマンド・プロセッサによって、アナリティック・ワークスペース・オブジェクトがその名前で検索されます。

エスケープ・シーケンス

テキスト・データに、印刷できない値が含まれることがあります。そのような値を入力および表示するには、エスケープ・シーケンスを使用します。エスケープ・シーケンスは、円記号で始まる一連の英数字です。

次の表に、認識されるエスケープ・シーケンスを示します。

エスケープ・シーケンス	意味
¥b	バックスペース。
¥f	改ページ。
¥n	ライン・フィード（改行）。
¥r	キャリッジ・リターン（改行）。
¥t	水平タブ。
¥"	二重引用符。
¥'	一重引用符。
¥¥	円記号。
¥d*nnn*	ASCII コード *nnn*（10 進）の文字。¥*d* は 10 進のエスケープを示し、nnn は文字の 10 進値です。
¥x*nn*	ASCII コード *nn*（16 進）の文字。¥x は 16 進のエスケープを示し、*nn* は文字の 16 進値です。
¥U*nnnn*	Unicode *nnnn* の文字。¥U は Unicode のエスケープを示し、*nnnn* は値 U+*nnnn* で Unicode のコード・ポイントを表す 4 桁の 16 進の整数です。U は大文字にする必要があります。

リテラル・テキスト値の例

リテラル・テキスト値の例を次に示します。

```
'Raoul D¥'Allesandro'
'NONE'
'January 2002'
```

ブール・データ型

論理値を表すブール・データ型を使用できます。コードでは、次の値のいずれか（大文字および小文字の任意の組合せ）を使用してブール値を示すことができます。

- YES、TRUE、ON
- NO、FALSE、OFF

使用される値は、NLS_LANGUAGE オプションによって指定される言語によって決まります。読込み専用の NOSPELL オプションおよび YESSPELL オプションを使用して、値を取得できます。

ブール式の使用の詳細は、4-18 ページの「ブール式」を参照してください。

日付データ型

次の日付データ型がサポートされています。

データ型	データ値
DATETIME	紀元前 4712 年 1 月 1 日から西暦 9999 年 12 月 31 日までの日付と、時間、分および秒
DATE	西暦 1000 年 1 月 1 日から西暦 9999 年 12 月 31 日までの日付

DATETIME 値の書式および言語は、NLS_DATE_FORMAT オプションおよび NLS_DATE_LANGUAGE オプションによって制御されます。DATETIME データ型は、Oracle の標準的なライブラリによってサポートされ、データベースでの動作と同様に動作するため、DATE データ型より優先的に使用してください。DATE 値の書式設定を制御する DATEORDER オプション、DATEFORMAT オプションおよび MONTHNAMES オプションは、DATETIME 値には影響を及ぼしません。ただし、DATETIME 値および DATE 値は、ほとんどの DML コマンドで区別なく使用できます。

ディメンションの定義

ディメンションとは、1 つ以上の変数の編成を指定する値のリストが含まれるオブジェクトです。ディメンション値は、リレーショナル表のキーに類似しています。単独で使用する場合でも他のディメンション値と併用する場合でも、ディメンション値はデータ値を一意に識別します。たとえば、月ごとの売上高が別々に示されている売上データの場合、データには month ディメンションが設定されており、データは月ごとに編成されています。feb02、mar02、apr02 などのディメンション値を追加できます。

単純なディメンションには、同じデータ型を持つ値のみのリストが含まれます。OLAP DML では、単純なフラット・ディメンションと単純な階層ディメンションの両方がサポートされています。

- ディメンション内のすべての値が同じレベルに含まれている場合、**フラット・ディメンション**が存在します。いずれの値も、別の値の子または親ではありません。
- 値が 1 対多（親対子）の関係にある場合、**階層ディメンション**が存在します。階層ディメンションは、単一のディメンション内でこのようなデータを編成および構成するための手段です。階層ディメンションを使用して、すべてのレベルのデータが含まれる変数をディメンション化できます。ディメンションに複数の階層が含まれる場合もありま

す。セルフ・リレーションを作成することによって、ディメンション値の親子関係を指定します。

これらの単純なベース・ディメンションから複合ディメンションおよび結合ディメンションを導出し、多次元形式で疎密なデータをより効率的に格納できます。

連結ディメンションは、単純なディメンションまたは結合ディメンションに基づくことができます。ベース・ディメンション中に 2 つ以上の単純なフラット・ディメンションが存在する連結ディメンションを使用して、階層を表すことができます。連結ディメンションを使用すると、アナリティック・ワークスペースのディメンションをリレーショナル表の列に簡単にマッピングできるため、リレーショナル構造からアナリティック・ワークスペース構造へのデータのロードの効率を改善できます。連結ディメンションのベース・ディメンションは、様々なデータ型を持つ場合があります。

参照：

- 複合ディメンションおよび結合ディメンションの詳細は、3-17 ページの「疎密なデータを効率的に処理する変数の定義」を参照してください。
- 階層ディメンションの詳細は、3-21 ページの「階層ディメンションおよび階層ディメンションを使用する変数の定義」を参照してください。
- 連結ディメンションの詳細は、3-24 ページの「連結ディメンションおよび連結ディメンションを使用する変数の定義」を参照してください。

定義するディメンションの決定

アナリティック・ワークスペースにフラット・ディメンションのみを含める場合、ユーザーがアクセスするデータの各ディテール・レベルのディメンションを定義する必要があります。

たとえば、会社が複数の営業地区に分割されており、地区ごとに複数の店舗の収支が処理されている場合、各店舗の売上高が必要か、または各地区の売上高のみが必要かを判断する必要があります。次の表に示すとおり、この判断によって、アナリティック・ワークスペースの構造が決定されます。

条件	結果
店舗データが必要な場合	store ディメンションを定義できます。
それぞれの地区全体のデータを常に参照する場合	district ディメンションのみが必要です。
両方のデータを参照する場合	store ディメンションと district ディメンションの両方を作成し、これらのディメンション間にリレーションを作成することによって、店舗ごとにデータを編成し、地区ごとのデータの集計を参照できます。

1 つの変数内に様々な集計レベルのデータを格納する場合もあります。このような格納を使用すると、データを参照するユーザーに対する応答時間が短縮されるためです。この場合、階層のすべての値が含まれる 1 つの階層ディメンションを定義するか、または単純なフラット・ディメンションに基づく連結ディメンションを定義できます。たとえば、各フラット・ディメンションに、階層のいずれかのレベルの値を含めることができます。

たとえば、store ディメンションと district ディメンションの両方を定義する前述の方法を使用せずに両方の方法でデータを参照する場合、1 つの階層ディメンションを定義できます。この階層ディメンションには、店舗および地区に関するすべての値を含めます。この階層ディメンションによって変数をディメンション化すると、1 つの変数内に様々な集計レベルのデータを格納できます。店舗データおよび地区データを個別に参照することもできます。

ベース・ディメンションとして store ディメンションおよび district ディメンションを持つ連結ディメンションを定義して、同様の結果を得ることもできます。連結ディメンションにも、店舗および地区に関するすべての値が含まれます。階層ディメンションの場合と同様に、この連結ディメンションによって変数をディメンション化すると、その変数内に様々な集計レベルのデータを保持することができ、店舗データおよび地区データを個別に参照することもできます。

アナリティック・ワークスペースにすでに単純なフラット・ディメンションが含まれている場合、またはリレーショナル・ディメンションの列に簡単にマッピングできるように単純なフラット・ディメンションを作成する場合、階層ディメンションではなく連結ディメンションを使用します。単純な階層ディメンションではなく連結ディメンションを使用するもう 1 つの理由は、単純なディメンションのすべての値は一意である必要があるのに対して、連結ディメンションでは連結される 2 つ以上のベース・ディメンションに同じ値を含めることができるためです。

単純なフラット・ディメンションのデータを格納する方法

単純なフラット・ディメンションのデータは、1 次元の配列に格納されます。ディメンションに値を追加すると、新しい値は配列の末尾に格納されます。

product ディメンションが TEXT データ型として定義されていると想定します。ディメンションに追加される最初の 3 つの値は、TENTS、CANOES および RACQUETS です。この時点で、ディメンションのレポートには次の値が表示されます。

```
PRODUCT
--------------
TENTS
CANOES
RACQUETS
```

product ディメンションの値は、実際には次のとおり格納されます。

位置	1	2	3
値	TENTS	CANOES	RACQUETS

その後、値 SPORTSWEAR および FOOTWEAR を追加します。この時点で、ディメンションのレポートには次の値が表示されます。

```
PRODUCT
--------------
TENTS
CANOES
RACQUETS
SPORTSWEAR
FOOTWEAR
```

product ディメンションの配列は、次のようになります。

位置	1	2	3	4	5
値	TENTS	CANOES	RACQUETS	SPORTSWEAR	FOOTWEAR

参照： ディメンション値を追加する方法の詳細は、第 5 章「アナリティック・ワークスペースの データ・オブジェクトの移入」を参照してください。

ディメンション・サロゲートの定義

ディメンション・サロゲートは、別の方法でディメンションの位置を指定するためのオブジェクトです。ディメンションの各値は、ディメンション内の位置によって識別されます（3-10 ページの「単純なフラット・ディメンションのデータを格納する方法」を参照）。位置は、整数で指定します。INTEGER 型のディメンションの場合、値と位置は同じになります。LIMIT コマンドまたは修飾データ参照（QDR）では、ディメンションの値またはディメンションでの値の位置を使用できます。たとえば、次の両方のコマンドを実行すると、product ディメンションのステータスが同じ値に設定されます。

```
LIMIT product TO 'TENTS'
LIMIT product TO 1
```

リレーショナル表の主キー列には、数値が含まれる場合があります。リレーショナル構造からアナリティック・ワークスペースにデータを効率的にロードするために、NUMBER ディメンションを定義して主キーの値を含めることができます。NUMBER ディメンションは、他の型のディメンションとは異なり、ディメンションの値をディメンションでの位置によって指定できません。ただし、NUMBER ディメンションに INTEGER 型のディメンション・サロゲートを定義すると、LIMIT コマンド、モデル、QDR およびデータのロードで、NUMBER ディメンションからの主キーの値を使用するかわりに、サロゲートの値を使用できます。

単純なディメンションおよび結合ディメンションにはディメンション・サロゲートを定義できますが、連結ディメンションまたは複合ディメンションにはディメンション・サロゲートを定義できません。たとえば、結合ディメンションが必要で、結合ディメンションの各値を指定する 1 つのテキスト値も必要な場合があります。このような場合、結合ディメンションに TEXT ディメンション・サロゲートを作成します。結合ディメンションにディメンション・サロゲートを定義する場合、結合ディメンションを複合ディメンションに変換することはできません。

ディメンションには任意の数のディメンション・サロゲートを定義できます。ディメンション・サロゲートのタイプは、ディメンションのタイプと同じである必要はありません。ディメンション・サロゲートは、DAY、WEEK、MONTH、QUARTER または YEAR 時間型を除く、任意のディメンション・タイプに定義できます。これらの時間型は、旧バージョンとの互換性のためのみに提供されます。これらの時間型を使用しないことをお薦めします。

ディメンションとディメンション・サロゲートの相違点

ディメンション・サロゲートによってオブジェクトをディメンション化することはできません。ただし、変数などのオブジェクトをディメンションによってディメンション化し、そのディメンションにディメンション・サロゲートを定義して、LIMIT コマンド、モデル、QDR およびデータのロードでディメンションのかわりにサロゲートの値を使用できます。

ディメンション・サロゲートに値セットを定義することはできません。ただし、ディメンションに値セットを定義し、そのディメンションにディメンション・サロゲートを定義して、LIMIT コマンドでサロゲートの値を使用すると、値セットに値を指定できます。

ディメンション・サロゲートにリレーションを定義することはできません。ただし、リレーションをディメンション化するディメンションにディメンション・サロゲートを定義して、LIMIT コマンドまたは QDR でサロゲートの値を使用できます。

プログラムまたはフォーミュラのデータ型としてサロゲートを使用することはできません。

ディメンション・サロゲートに新しい位置を直接追加することはできません。ただし、MAINTAIN コマンドを使用して、サロゲートが定義されているディメンションに値を追加できます。サロゲートには、ディメンションに追加した各値に対する新しい位置が自動的に設定されます。

ディメンション・サロゲートは、固有のステータスを持ちません。ディメンション・サロゲートは、ディメンションのステータスを共有します。LIMIT コマンドまたは QDR でディメンションまたはディメンション・サロゲートの値または位置を使用して、ディメンションおよびディメンション・サロゲートのステータスを設定できます。

ディメンションにディメンション・サロゲートが存在する場合、そのディメンションを削除することはできません。ただし、ディメンション・サロゲートを削除してもディメンションに影響を及ぼしません。

ディメンション・サロゲートに PERMIT コマンドを使用することはできません。サロゲートには、ディメンションに設定されている権限が設定されます。

ACROSS キーワードまたは DOWN キーワードを使用するコマンドでディメンション・サロゲートを使用して、指定されたディメンションのループ処理、合計またはレポートを行うことはできません。このような場合、サロゲートではなくディメンションを指定する必要があります。

ディメンション・サロゲートに CHDGFN コマンドまたは MAINTAIN コマンドを使用することはできません。ただし、MAINTAIN コマンドでディメンション・サロゲートの値を使用して、ディメンションの値を指定できます。

リレーションの定義

リレーションとは、任意のディメンション値を、アナリティック・ワークスペースの同じディメンションの値または別のディメンションの値に関連付けるオブジェクトです。リレーションの構造は、変数の構造に類似しています。ただし、リレーションのセルには実際のデータ値は含まれません。リレーションの各セルには、ディメンションの値の索引が含まれます。

1 対多（親子）の関係に含まれる 2 つのディメンション間にリレーションを作成することによって、子ディメンションごとにデータを編成し、親ディメンションごとにデータの集計を表示できます。たとえば、store ディメンションと district ディメンション、およびこれらのディメンション間のリレーションを定義する場合、store ごとにデータを編成し、district ごとにデータの集計を表示できます。

2 つ以上のディメンション間のリレーション、一連のディメンション間の複数のリレーションまたは自身のディメンションに対するリレーション（**セルフ・リレーション**）を明示的に定義できます。

リレーションをディメンション化する方法

リレーションは、ディメンション化された配列です。リレーションは、多数の値または少数の値を持つディメンションによってディメンション化できます。

通常、リレーションは多数の値を持つディメンション（下位の集計ディメンションまたは子ディメンション）によってディメンション化され、関連ディメンションは少数の値を持つディメンション（上位の集計ディメンションまたは親ディメンション）になります。たとえば、`state.city` というリレーションを作成して、各市をその市が存在する州に関連付けることができます。リレーションは、`city` によってディメンション化され、関連ディメンションは `state` になります。各市に州を割り当てます。

一般的ではありませんが、少数の値を持つディメンション（上位の集計ディメンションまたは親ディメンション）によってリレーションがディメンション化される場合もあります。この場合、他のディメンションのすべての値が関連付けられるわけではありません。たとえば、州とその州都の間に `city.state` という名前の関係を作成すると想定します。リレーションは、`state` によってディメンション化され、関連ディメンションは `city` になります。州都のみが州に割り当てられます。

リレーション・データを格納する方法

リレーションのディメンションを定義する順序によって、データの格納方法およびアクセス方法が決まります。ディメンションは、定義に示された順序で、最初のディメンションが最も速く変化し、最後のディメンションが最も遅く変化します。速く変化するディメンションおよび遅く変化するディメンションの詳細は、3-16 ページの「変数データを格納する方法」を参照してください。

リレーションに格納されるデータ値は、関連ディメンションの索引になります。索引は、ディメンションでの値の位置を示します。

たとえば、`state.city` というリレーション（`city` によってディメンション化され、`state` という関連ディメンションを持つ）では、各市に州が割り当てられます。この関係を実装するために、`city` ディメンションの各値（索引）に `state` ディメンションの索引が格納されます。次の表に、`state` ディメンションの各位置に割り当てられる `city` ディメンションの位置を示します。この表には、ディメンションのこれらの位置での値も示します。

市の位置（索引）	この位置での市の値	州の位置（索引）	この位置での州の値
1	Atlanta	1	Georgia
2	Chicago	2	Illinois
3	Springfield	2	Illinois

参照 :

- リレーションへ値を追加する方法の詳細は、第 5 章「アナリティック・ワークスペースの データ・オブジェクトの移入」を参照してください。
- 式でリレーションを使用する方法の詳細は、第 4 章「式の使用」を参照してください。

例 : 2 つのディメンション間のリレーション

ほとんどのリレーションは、あるディメンションの値を別のディメンションの値と関連付ける 1 次元配列です。たとえば、state および city という 2 つの単純なディメンションを定義し、これらのディメンション間に state.city というリレーションを定義して各市をその市が存在する州に関連付けることができます。

state.city リレーションが次のコマンドを使用して定義されると想定します。

```
DEFINE state.city RELATION state <city>
```

次に示すとおり、state ディメンションに 2 つの値が含まれ、city ディメンションに 3 つの値が含まれると想定します。

```
STATE
--------------
GEORGIA
ILLINOIS

CITY
--------------
ATLANTA
CHICAGO
SPRINGFIELD
```

state.city リレーションは city によってディメンション化され、関連ディメンションは state になります。次に示すとおり、state.city リレーションによって、各市に州が割り当てられます。

```
CITY           STATE.CITY
-------------- ---------------
ATLANTA        GEORGIA
CHICAGO        ILLINOIS
SPRINGFIELD    ILLINOIS
```

例 : セルフ・リレーション

1 つのディメンションに対してセルフ・リレーションを定義できます。たとえば、会社の管理構造を追跡するには、employee ディメンションに対して emp.emp リレーションを定義できます。

emp.emp リレーションが次のコマンドを使用して定義されると想定します。

```
DEFINE emp.emp RELATION employee <employee>
```

employee ディメンションに次に示す値が含まれていると想定します。

```
EMPLOYEE
--------------
ANN LOGAN
MICHAEL ARON
LUCY BATES
RALPH BURNS
```

emp.emp セルフ・リレーションは employee ディメンションによってディメンション化され、関連ディメンションも employee ディメンションになります。次に示すとおり、emp.emp リレーションによって、各従業員に管理者が割り当てられます。

```
EMPLOYEE         EMP.EMP
-------------- ----------
ANN LOGAN      NA
MICHAEL ARON   ANN LOGAN
LUCY BATES     ANN LOGAN
RALPH BURNS    LUCY BATES
```

この例では、社長の Ann Logan を管理する従業員は存在せず、従業員の Lucy Bates および Michael Aron は社長の Ann Logan によって直接管理され、従業員の Ralph Burns は従業員の Lucy Bates によって管理されます。

参照 :

- 階層ディメンションでセルフ・リレーションを使用する方法の詳細は、3-21 ページの「階層ディメンションおよび階層ディメンションを使用する変数の定義」を参照してください。
- 連結ディメンションでセルフ・リレーションを使用する方法の詳細は、3-24 ページの「連結ディメンションおよび連結ディメンションを使用する変数の定義」を参照してください。

変数の定義

変数は、データを格納するオブジェクトです。1 つの変数のすべてのデータは、同じデータ型の同じ測定単位を表します。ビジネスで（ドル、個、パーセントなどで測定される）複数のカテゴリのトランザクションを使用する場合、各カテゴリを固有の変数に格納します。たとえば、売上データをドル（sales 変数）および個数（units 変数）で記録できます。

通常、変数を使用して、ビジネスの特定の側面を定量化するデータ値を含めます。

変数のタイプ

変数は、ディメンション化変数または非ディメンション化変数のいずれかです。

- ディメンション化変数 : 変数がディメンションの配列である場合、これらのディメンションによってデータが編成されます。ディメンション値の各組合せに対して 1 つのセルが存在します。このような変数は、**ディメンション化変数**といいます。1 つの変数は、最大 32 のディメンションによってディメンション化できます。
- 非ディメンション化変数 : 変数にディメンションが含まれない場合、その変数は**スカラー**（単一セル変数）であり、1 つのデータ値が含まれます。

アナリティック・ワークスペースに定義する変数は、永続変数または一時変数です。プログラムに変数を定義することもできます（7-5 ページの「ローカル変数の定義」を参照）。

永続変数とは、変数値と定義の両方がアナリティック・ワークスペースに格納される変数です。

一時変数とは、現行のセッション中のみ値を持つ変数です。アナリティック・ワークスペースを更新およびコミットすると、一時変数の定義のみが保存されます。アナリティック・ワークスペースからの連結を解除すると、データ値は廃棄されます。

変数データを格納する方法

変数定義にディメンションを示す順序によって、その変数のデータの格納方法およびアクセス方法が決定されます。変数定義に示す最初のディメンションは**最も速く変化するディメンション**で、最後のディメンションは**最も遅く変化するディメンション**です。

たとえば、アナリティック・ワークスペースに、オフィスが存在する各市の月ごとの経費が含まれる opcosts 変数が存在すると想定します。次に示す opcosts 変数の定義では、month は最も速く変化するディメンションで、city は最も遅く変化するディメンションです。

```
DEFINE opcosts VARIABLE DECIMAL <month city>
```

多次元変数のデータは、値の線形ストリームとして格納されます。最も速く変化するディメンションの値は、クラスタ化されます。たとえば、opcosts 変数の場合、すべての月の Boston の値が順に格納され、次にすべての月の Chicago の値が順に格納されます（以後、

同様に続きます)。そのため、次の表に示すとおり、opcosts 変数では month の値が最も速く変化します。

ディメンション値	JAN97 BOSTON	FEB97 BOSTON	. . . . .	DEC97 BOSTON	JAN97 CHICAGO	. . . . .
変数値	16000.77	16000.28	. . .	16000.98	19000.24	. . .

変数および他のディメンション化されたオブジェクトを定義する場合、およびネストしたループで多次元式をループ実行するプログラムを作成する場合、常に、最も速く変化するディメンションと内部ループを一致させて、パフォーマンスを最適化する必要があります。

疎密なデータを効率的に処理する変数の定義

疎密なデータが含まれる変数とは、実際のデータが含まれないセルの割合が比較的高い変数です。このような空（NA）の値は、アナリティック・ワークスペースの記憶域を占有します。

疎密性には次の 2 つのタイプがあります。

- **制御された疎密性**: 1 つ以上のディメンション値の特定範囲にデータが含まれない場合に発生します。たとえば、過去のデータが存在しない month によってディメンション化された新しい変数で発生します。month ディメンションには定義済の過去の月が含まれるため、セルは存在しますが、空になります。
- **ランダムな疎密性**: NA 値がデータ変数の中に散在するときに発生します。通常、ディメンション値の一部の組合せにデータが含まれないために発生します。たとえば、ある地域で、特定の製品のみが販売され、他の製品のデータが存在しない場合があります。他の地区では、異なる製品群が販売されている場合があります。

定義 : 複合ディメンション

複合ディメンションは、疎密なデータを集約して変数に格納するために使用される内部オブジェクトです。複合ディメンションは、ディメンション値の組合せのリストです。組合せには、複合ディメンションのベースとなる各ディメンションからの 1 つの値が含まれます。

複合ディメンションには、名前付き複合ディメンションおよび名前なし複合ディメンションが存在します。

- **名前なし複合ディメンション**はアナリティック・ワークスペース・オブジェクトではなく、内部構造です。変数を定義する際に SPARSE キーワードを使用して、名前なし複合ディメンションが自動的に作成されるように要求します。

- **名前付き複合ディメンション**は、DEFINE COMPOSITE コマンドを使用して定義するオブジェクトです。後で変数を定義したり、または変数にアクセスする際に、他のディメンションの名前とともに、この複合ディメンションを名前で指定することができます。

複合ディメンションの値は自動的に保持されるため、アナリティック・ワークスペースの疎密性を処理するには、複合ディメンションを使用することをお薦めします。

複合ディメンションを使用することは、疎密性を管理するための最も重要な手順の 1 つです。これによって、アナリティック・ワークスペースが最小サイズに保たれ、パフォーマンスが向上します。

名前付き複合ディメンションを使用するメリット

名前付き複合ディメンションを使用すると、同じ複合ディメンションを共有する変数を簡単に追跡できます。変数のディメンションのリストに名前付き複合ディメンションを含めると、Oracle OLAP で、この変数では名前付き複合ディメンション内のディメンションが疎密であること、およびこの複合ディメンションを同じ疎密性のパターンを持つ他の変数のみと共有することを指定できます。

一方、名前なし複合ディメンションを使用して定義されており、同じディメンションを同じ順序で持つすべての変数は、この名前なし複合ディメンションを自動的に共有します。これらの変数で疎密性のパターンが異なる場合、パフォーマンスが低下します。

結合ディメンションを使用して任意の変数にデータが含まれるディメンション値の組合せを保持することによって、疎密性を管理することもできます。ただし、複合ディメンションの値は自動的に保持されるため、複合ディメンションを使用してアナリティック・ワークスペースの疎密性を処理することをお薦めします。

複合ディメンションを使用する方法

多次元変数を定義する場合、ディメンションのリストに複合ディメンションを指定できます。

最初に、DEFINE COMPOSITE コマンドを使用して、名前付き複合ディメンションを定義します。その後、次の構文を使用して各変数のディメンション・リストに名前付き複合ディメンションを含め、変数を定義します。

composite-name <*dims*>

たとえば、次のコマンドに示すとおり、proddist という名前の複合ディメンションを定義して、そのディメンションに product および district を含めると想定します。

DEFINE *proddist* COMPOSITE <*product district*>

次のコマンドに示すとおり、sales 変数を定義して、time を最も速く変化するディメンションとし、proddist 複合ディメンションを最も遅く変化するディメンションとすると想定します。

```
DEFINE sales <time proddist<product district>>
```

複合ディメンションとともに SPARSE キーワードを使用することはできません。SPARSE キーワードのかわりに複合ディメンションの名前を使用します。

参照：

- 複合ディメンションの使用方法の詳細は、4-11 ページの「式での複合ディメンションの使用」を参照してください。
- 疎密なデータの処理方法の詳細は、4-28 ページの「NA 値の処理」を参照してください。

複合ディメンションの命名、名前変更、および名前なし複合ディメンションへの変更

RENAME コマンドを使用すると、次の操作を実行できます。

- 名前なし複合ディメンションの命名
- 名前付き複合ディメンションの名前の変更
- 名前付き複合ディメンションから名前なし複合ディメンションへの変更

複合ディメンションを使用する変数へのデータの追加

多次元変数を定義する場合、ベース・ディメンションのかわりに複合ディメンションを使用してデータをディメンション化するように指定できます。その後、複合ディメンションを定義した変数のディメンションに値を追加すると、次の操作が実行されます。

- 複合ディメンションにディメンション値の組合せが設定されます。
- 変数のデータが、ベース・ディメンションの構造ではなく複合ディメンションの構造を使用して格納されます。

複合ディメンションを使用する変数の場合、複合ディメンションのディメンション値の組合せで使用されるディメンション値のみに対してセルが作成されます。ベース・ディメンションのすべての値に対して変数のセルが作成されるわけではありません。変数のデータは、ディメンション値の各組合せに対して、セルごとに順に格納されます。データの格納の観点から見ると、複合ディメンションに含まれるベース・ディメンション値の各組合せは、通常のディメンションの値と同様に処理されます。これは、1 つの通常のディメンションおよび 1 つの複合ディメンションで変数を定義する場合、2 次元の変数と同様に格納されることを意味します。

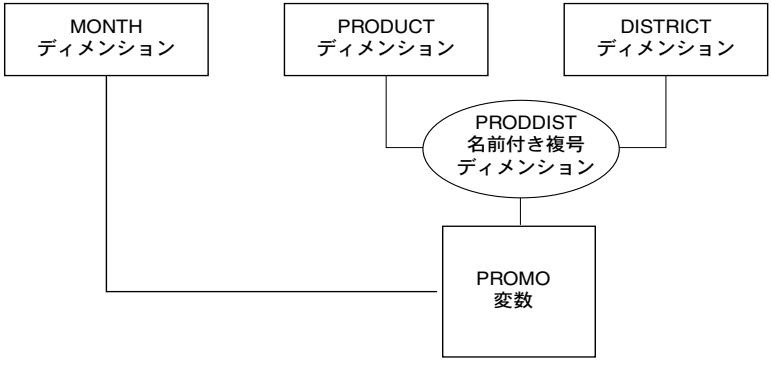
例 3-1 名前付き複合ディメンションを使用する変数の定義

一部の地域で特定の製品に関するプロモーションを行っている場合、変数データの product ディメンションおよび district ディメンションが疎密になります。そのため、ベース・ディメンションが product および district である proddist という名前の複合ディメンションを定義すると想定します。データが存在する値のみに対して、複合ディメンションにディメンション値の組合せが存在します。たとえば、テントについてプロモーションを行い、カヌーについては行わない場合、複合ディメンションにはテントと市の組合せが含まれ、カヌーと市の組合せは含まれません。

次のコマンドを実行すると、month によってディメンション化されている promo という名前の変数、およびベース・ディメンションが product および district である proddist という名前の複合ディメンションが作成されます。

```
DEFINE promo INTEGER <month proddist<product district>>
```

次の概念図に、このコマンドによって作成される promo 変数、month、ベース・ディメンション product と district、ベース・ディメンション product と district から作成された名前付き複合ディメンション proddist、およびベース・ディメンション（product と district）と proddist 複合ディメンションの間に作成された内部リレーションを示します。



次の表に、promo 変数のデータが格納されるシーケンスの例を示します。

TENTS BOSTON JAN95	TENTS BOSTON FEB95	TENTS BOSTON MAR95	. . . . . .	RACQUETS CHICAGO JAN95	RACQUETS CHICAGO FEB95	. . . . . .
257	379	428	. . .	635	192	. . .

単一の複合ディメンションを持つ変数の定義

変数定義で 1 つのみのディメンションに複合ディメンションを指定する場合、単一の複合ディメンションが作成されます。この単一の複合ディメンションの値は、ベース・ディメンションの値のサブセットになります。

他の変数と同じディメンションを共有する変数が特定の 1 つのディメンションに含まれるいくつかの値のデータのみを持つ場合、単一の複合ディメンションの使用が有効です。

line、division および month ディメンションを使用して actual という名前の変数を定義済であると想定します。actual 変数には、NA 値は含まれません。actual より必要とする詳細が少ない budget という名前の変数を定義する必要があります。たとえば、budget では line ディメンション値の 10% が必要ですが、actual ではこのすべてのディメンション値が必要です。

疎密性に関する設定を行わずに budget を定義すると、すべての month および division に対して line ディメンション値が含まれます。ただし、line ディメンションのセルの 90% は、NA 値になります。

この場合に疎密なデータを処理するには、次に示すとおり、line ディメンションのみに対する名前なし複合ディメンションを使用して budget を定義します。

```
DEFINE budget DECIMAL <SPARSE <line> division month>
```

階層ディメンションおよび階層ディメンションを使用する変数の定義

階層ディメンションは、単一ディメンション内で親子（1 対多）データを編成および構成し、セルフ・リレーションを使用して階層ディメンションの値をグループに編成するための方法です。ディメンション内の値がレベルに配置され、各レベルが下位レベルからのデータの集計の合計を表す場合、階層が存在します。複数の階層が存在するディメンションもあります。

階層ディメンションを使用すると、1 つの変数内に様々な集計レベルのデータを格納できます。このような格納を使用すると、特に変数が大きい場合に、データを参照するユーザーに対する応答時間が短縮されます。

市および地域に対して 2 つの個別のディメンションを定義するのではなく、市と地域の両方の値が含まれる geography という名前の階層ディメンションを定義できます。

```
GEOGRAPHY
--------------
EAST
WEST
BOSTON
SAN FRANCISCO
SEATTLE
```

階層ディメンションを持つ変数の定義

階層ディメンションを使用すると、1 つの変数内に様々な集計レベルのデータを含む変数を定義できます。このような格納を使用すると、特に変数が大きい場合に、データを参照するユーザーに対する応答時間が短縮されます。

多くの場合、階層ディメンションの上位レベルの値に対応する変数のセルには、下位レベルのディメンション値に対応する変数のセルに含まれる値の合計が含まれます。たとえば、時間を表す階層ディメンションで定義された sales 変数では、各四半期の値のセルが、その四半期のすべての月の総売上高を表します。

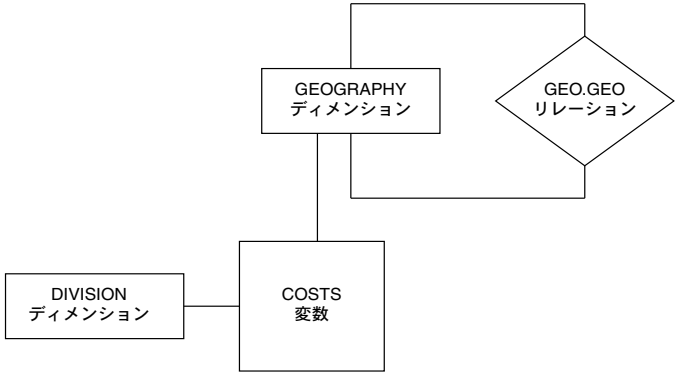
階層ディメンションで変数を定義した後、階層の最下位レベルに変数データを追加し、階層の上位レベルの値を計算または**集計**できます。逆に、階層の上位レベルから下位レベルにデータを分散または**割り当て**ることができます。

参照 :

- データを割り当てる方法の詳細は、第 9 章「データのアロケーション」を参照してください。
- データを集計する方法の詳細は、第 12 章「データの集計」を参照してください。

例 : 階層ディメンションを持つ変数

次の概念図に、市と地域の両方の値が含まれる geography ディメンション、市と地域の間の関係を定義する geo.geo リレーション、部門のリストが含まれる division ディメンション、および市ごとの各 division の支出および地域ごとの合計が含まれる costs 変数を示します。



division ディメンションおよび geography ディメンションには、次の値が含まれます。

```
DIVISION
--------------
DIVA
DIVB

GEOGRAPHY
--------------
EAST
WEST
BOSTON
SAN FRANCISCO
SEATTLE
```

geo.geo リレーションが次のコマンドを使用して定義されると想定します。

```
define geo.geo relation geography <geography>
```

geo.geo セルフ・リレーションで地域の値が市の値に割り当てられた後、geo.geo のレポートには次の値が表示されます。

```
GEOGRAPHY       GEO.GEO
--------------- ----------
EAST            NA
WEST            NA
BOSTON          EAST
SAN FRANCISCO   WEST
SEATTLE         WEST
```

costs の最下位レベル（市レベル）にデータ値を入力すると、cost に次の値が含まれます。

```
          ------------------------COSTS-----------------------------
          ----------------------GEOGRAPHY---------------------------

DIVISION     EAST       WEST      BOSTON   SAN FRANCISCO      SEATTLE
--------- ---------- ---------- ---------- -------------    ----------
DIVA           NA       NA       27,600.00  10,000.00       40,000.00
DIVB           NA       NA       30,000.00  12,000.00       50,000.00
```

データを集計すると、costs 変数では、EAST 地域および WEST 地域の合計値のセルを含むすべてのセルに値が含まれます。

```
          ------------------------COSTS-----------------------------
          ----------------------GEOGRAPHY---------------------------

DIVISION     EAST       WEST      BOSTON   SAN FRANCISCO      SEATTLE
--------- ---------- ---------- ---------- -------------    ----------
DIVA       27,600.00  50,000.00  27,600.00  10,000.00       40,000.00
DIVB       30,000.00  62,000.00  30,000.00  12,000.00       50,000.00
```

連結ディメンションおよび連結ディメンションを使用する変数の定義

連結ディメンションでは、2 つ以上のベース・ディメンションが結合されて、1 つのディメンションになります。ディメンション内に親子データを編成および構成する別の方法として、単純な階層ディメンションのかわりに連結ディメンションを使用することができます。セルフ・リレーションを使用して、連結ディメンションの値を階層のレベルごとにグループに編成します。

リレーショナル・ディメンション表に、市の名前を値に持つ地区の列、および地域の列が含まれると想定します。アナリティック・ワークスペースに district ディメンションおよび region ディメンションを定義して、これらのディメンションにリレーショナル列の値をロードできます。これらのディメンションの値は次のとおりです。

```
DISTRICT
----------
BOSTON
SAN FRANCISCO
SEATTLE

REGION
-------
EAST
WEST
```

これらの単純なフラット・ディメンションに基づく reg.dist.ccdim という名前の連結ディメンションを定義できます。連結ディメンションには、両方のディメンションの値が含まれます。

```
REG.DIST.CCDIM
-------------------------
<REGION: EAST>
<REGION: WEST>
<DISTRICT: BOSTON>
<DISTRICT: SAN FRANCISCO>
<DISTRICT: SEATTLE>
```

次に、連結ディメンションの値を階層レベルにグループ化するセルフ・リレーションを定義できます。単純な階層ディメンションの場合と同様に、連結ディメンションを使用して、様々な集計レベルが含まれる変数を定義できます。

例 : 連結ディメンションを持つ変数

次に示すとおり、3-22 ページの「例 : 階層ディメンションを持つ変数」に示す reg.dist.ccdim 連結ディメンションおよび division ディメンションによってディメンション化された変数を定義できます。

```
DEFINE costs VARIABLE DECIMAL <reg.dist.ccdim division>
```

次に示すとおり、地区 / 地域階層の親子関係を識別する reg.dist.ccdim 連結ディメンションのセルフ・リレーションを定義できます。

```
DEFINE rdccdim.rdccdim RELATION reg.dist.ccdim <reg.dist.ccdim>
limit district to 'BOSTON'
rdccdim.rdccdim(REG.DIST.CCDIM district) = reg.dist.ccdim(REGION 'EAST')
limit district to 'DENVER' 'SEATTLE'
rdccdim.rdccdim(REG.DIST.CCDIM district) = reg.dist.ccdim(REGION 'WEST')
```

最下位レベル（district ディメンション・レベル）にデータを入力すると、costs 変数に次の値が含まれます。

```
          ------------------------COSTS-----------------------------
          --------------------REG.DIST.CCDIM-------------------------
           <REGION:   <REGION:  <DISTRICT: <DISTRICT:      <DISTRICT:
DIVISION    EAST>      WEST>     BOSTON>    SAN FRANCISCO>  SEATTLE>
--------- ---------- ---------- ---------- --------------- ----------
DIVA         NA         NA         27,600.00  10,000.00       40,000.00
DIVB         NA         NA         30,000.00  12,000.00       50,000.00
```

集計マップを作成して AGGREGATE コマンドを実行すると、costs 変数のデータを集計できます。データを集計すると、costs 変数では、EAST 地域および WEST 地域の合計値のセルを含むすべてのセルに値が含まれます。

```
          ------------------------COSTS-----------------------------
          --------------------REG.DIST.CCDIM-------------------------

           <REGION:   <REGION:  <DISTRICT: <DISTRICT:      <DISTRICT:
DIVISION    EAST>      WEST>     BOSTON>    SAN FRANCISCO>  SEATTLE>
--------- ---------- ---------- ---------- --------------- ----------
DIVA       27,600.00  50,000.00  27,600.00  10,000.00       40,000.00
DIVB       30,000.00  62,000.00  30,000.00  12,000.00       50,000.00
```

オブジェクトの定義の変更

アナリティック・ワークスペースに最後に定義したオブジェクトの定義が、現行の定義になります。現行の定義には、説明、プロパティ、権限などの特性を追加できます。現行ではない定義に特性を追加する場合、CONSIDER コマンドを使用してその定義を現行の定義にすることができます。

次の表に、オブジェクト定義に特性を追加できる OLAP DML コマンドを示します。

コマンド	説明
AGGMAP	AGGMAP タイプの新しい集計マップまたは既存の集計マップに新しい内容を指定して、AGGREGATE コマンドを使用できます。
ALLOCMAP	ALLOCMAP タイプの新しい集計マップまたは既存の集計マップに新しい内容を指定して、ALLOCATE コマンドを使用できます。
EQ	定義済のフォーミュラに対して計算する式を指定できます。
LD	オブジェクト定義に長い説明を割り当てます。
MODEL	新しいモデルまたは既存のモデルの新しい内容を指定できます。
PERMIT	オブジェクト定義にアクセス権を割り当てます。
PROGRAM	新しいプログラムまたは既存のプログラムの新しい内容を指定できます。
PROPERTY	オブジェクト定義にプロパティを割り当てます。

onplan という名前のブール変数を定義したと想定します。変数の定義に説明を追加します。

次に示すとおり、ONPLAN 変数の定義を変更するには、最初に ONPLAN を現行の定義に設定して、定義に説明を追加します。

```
CONSIDER onplan
LD Are these districts tracked on a special plan?
```

CHGDFN コマンドを実行すると、変数定義のいくつかの特性を再定義できます。次の例では、sales 変数のセグメント・サイズが変更されます。

```
CHGDFN sales SEGWIDTH 150 1000
```

DEFINE および CHGDFN コマンドの詳細は、Oracle9*i* OLAP DML Reference ヘルプのこれらのコマンドの項を参照してください。

4

式の使用

式は、OLAP DML の構文でデータ値を表します。この章では、式を作成および使用する方法について説明します。この章では、次の項目について説明します。

- 式の概要
- 式の次元性
- 式のディメンションに対する 1 つの値の指定
- 式でのワークスペース・オブジェクトの使用
- 数式
- テキスト式
- ブール式
- 条件式
- 置換式
- NA 値の処理

式の概要

式は、OLAP DML の構文でデータ値を表します。式は、コマンドまたはファンクションの引数として使用したり、オプションの値として使用することができます。式では、多くの場合、数学的または論理的な操作が実行されます。式では、常にアナリティック・ワークスペースのデータ型のいずれかに結果が評価されます。

式には、次のものが含まれます。

- 1 つのリテラル値（10 や EAST など）
- 複数の値を含む変数またはフォーミュラ（sales など）
- 1 つ以上の値を戻すファンクション（TOTAL や JOINLINES など）
- リテラル値、ディメンション、変数、フォーミュラまたはファンクションを、演算子と組み合せた計算（inflation*1.02 や actual gt 20000 など）

式はデータ型を持ちます。また、ディメンションを持つ場合もあります。式のデータ型およびディメンションは、式で使用する値によって異なります。

式のデータ型

式のデータ型は、次の基本的な型のいずれかになります。

- 数値型
- テキスト型
- 日付型（日付値に評価する）
- ブール型（YES 値または NO 値に評価する）

これらのデータ型の定義については、3-4 ページの「データ型」を参照してください。

式のデータ型の決定

式のデータ型が、結果値のデータ型になります。これは、式を構成するデータ・オブジェクトのデータ型と同じであるとはかぎりません。データおよび関連する演算子とファンクションによって異なります。

また、IF... THEN. . . ELSE 演算子で示される条件式もサポートされます。条件式で戻される値のデータ型は、IF 句の（ブールである必要がある）式ではなく、THEN および ELSE 句の式によって決定されます。

注意： 条件式と、IF コマンドを混同しないでください。IF コマンドは、構文が類似していても別の目的で使用されます。また、IF コマンドはデータ型を持たず、式のように評価されることはありません。

式のデータ型の変更

式のデータ型は、CONVERT ファンクションを使用して変更できます。たとえば、数値をテキストに変換したり、数字で構成されるテキスト文字列を数値に変換することができます。

ただし、データを同じ基本的なカテゴリ内の別の型に変換する必要はありません。これらの変換は自動的に行われます。通常、テキストを必要とする場所であれば TEXT、NTEXT または ID データを使用することができ、整数と小数を入れ替えて使用することもできます。

OLAP DML のデータ型については、3-4 ページの「データ型」を参照してください。

演算子

演算子は、値を変換したり、なんらかの方法で値を別の値と組み合せるための記号です。次の表に、OLAP DML 演算子のカテゴリを示します。

表 4-1 OLAP DML 演算子

カテゴリ	説明
算術	数式で数値データと使用できる演算子。結果は数値になります。一部の算術演算子は、結果を日付か数値のいずれかで戻す、日付データと数値データが混在した日付式でも使用できます。算術演算子の詳細は、表 4-2「算術演算子」を参照してください。
代入	式の結果をオブジェクトへ格納する代入文の作成に使用する演算子。代入文の使用方法の詳細は、5-10 ページの「代入文でのオブジェクトの使用」を参照してください。
比較	基本型（数値型、テキスト型、日付型またはブール型。ただし、ブール型になることはまれ）が同じである 2 つの値を比較し、結果をブール値で戻す演算子。比較演算子の詳細は、表 4-3「ブール演算子」を参照してください。
条件	ブール条件に基づいて、2 つの値のいずれかを選択する演算子。条件演算子の詳細は、4-25 ページの「条件式」を参照してください。
論理	論理演算によってブール値を変換し、ブール値を戻す演算子。論理演算子の詳細は、表 4-3「ブール演算子」を参照してください。
置換	式を評価し、その結果の値を置換する演算子。置換演算子の詳細は、4-27 ページの「置換式」を参照してください。

式の保存

式は、フォーミュラに保存できます。通常、フォーミュラを定義して、複雑な式または頻繁に使用される式を保存します。フォーミュラはオブジェクトであり、DEFINE FORMULA コマンドによって命名し、定義します。

たとえば、次に示すとおりフォーミュラを定義して売上（ドル）を計算できます。

```
DEFINE dollar.sales FORMULA units * price
```

フォーミュラを使用するたびに、それが表す式が評価されます。結果は格納されません。

式の次元性

式は、その式のすべての変数、ディメンション、リレーション、フォーミュラ、修飾データ参照およびファンクションのディメンションの集合によってディメンション化されます。

項目	ディメンション化元	コメント
変数 リレーション フォーミュラ	オブジェクトの定義に指定されたディメンション	**例 1:** price 変数が month および product によってディメンション化されている場合、式 price * 1.2 も month および product によってディメンション化されます。 **例 2:** units 変数が month、product および district によってディメンション化されている場合、（price 変数のディメンションが month および product のみであっても）式 units * price も month、product および district によってディメンション化されます。
修飾データ参照	修飾されたディメンションを除く、関連付けられたオブジェクトのすべてのディメンション	修飾データ参照については、4-5 ページの「式のディメンションに対する 1 つの値の指定」を参照してください。
ファンクション	多くの場合、その入力引数のディメンションの集合	Oracle9*i* OLAP DML Reference ヘルプで特に記載されていないかぎり、集計ファンクションで分割ディメンションまたは分割リレーションを指定すると、式の次元性が変更されます。分割ディメンションとして指定する最初のディメンションは最も遅く変化するディメンションで、最後のディメンションは最も速く変化するディメンションです。

式のディメンションの判断

式のディメンションは、PARSE コマンドおよび INFO ファンクションを使用して判断できます。PARSE によって式のテキストが評価され、INFO ファンクションによって式を解釈する方法が示されます。

この例では、INFO ファンクションで DIMENSION キーワードを使用して、PARSE コマンドによって分析された式のディメンションを取得する方法を示します。次のコマンドを実行すると、次に示す出力が生成されます。

```
PARSE 'TOTAL(sales region)'

SHOW INFO(PARSE DIMENSION)
REGION
```

ディメンションのステータスによる式の結果への影響

式によって生成される値の数は、式のディメンションおよびそのディメンションのステータスによって異なります。現行のステータス内のディメンション値の各組合せに対して、1 つのデータ値が算出されます。たとえば、month のステータスに 3 つ、product のステータスに 2 つのディメンション値が存在する場合、price gt 100 という式の結果は 6 つの値（3 × 2）となります。

したがって、目的の結果を得るには、式のディメンションを、考慮するデータ範囲に制限する必要があります。また、PERMIT コマンドを使用して、データのディメンションへのアクセスを制限することを考慮する必要があります。

> **参照：** ディメンションのステータスを設定する方法の詳細は、第 6 章「データの選択」を参照してください。

式のディメンションに対する 1 つの値の指定

修飾データ参照（QDR）は、式の 1 つ以上のディメンションを 1 つの値に制限するための方法です。QDR は、現行のステータスを変更せずに、1 つの値を指定する場合に有効です。QDR を使用すると、ディメンションを修飾したり（これによって式にディメンション値を 1 つ指定できます）、変数またはリレーションの 1 つ以上のディメンションを修飾することができます。

修飾データ参照の形式は次のとおりです。

expression(dimname1 dimexp1 [, dimname2 dimexp2. . .])

dimname 引数は、式のいずれかのディメンション（またはそのディメンションのディメンション・サロゲート）の名前であり、*dimexp* 引数は次のいずれかです。

- *dimname* の値
- 結果が *dimname* の値であるテキスト式
- 結果が *dimname* の値の論理位置である数式
- *dimname* のリレーション

注意： 複雑な式を修飾するには、QUAL ファンクションを使用してください。

変数の修飾

次のいずれかの方法を使用して、変数の任意またはすべてのディメンションを修飾できます。

- 指定したディメンション値を 1 つ選択すると、QDR によって一時的に変数のディメンションを制限できます。この値は、現行のステータスには含まれません。
- 修飾子として適切なリレーションの名前を指定すると、QDR によって、変数のディメンションをより集計度が低い関連ディメンションに置換できます。ディメンションは、リレーションのディメンションによって一時的に置換されます。

たとえば、変数 sales には、month、product および district という 3 つのディメンションが含まれています。Boston の総売上を全都市の総売上と比較する必要があり、1 つのコマンドで、district を次の 2 つの異なる値に制限すると想定します。

- 式の分子では、district のステータスを BOSTON にします。
- 式の分母では、district のステータスを ALL にします。

次のコマンドを実行すると、QDR を使用してこの結果を計算できます。

```
SHOW sales(district 'BOSTON')/TOTAL(sales)
```

複数の変数ディメンションを修飾できます。たとえば、各ディメンションの 1 つのディメンション値を指定して、sales 変数のすべてのディメンションを修飾すると、sales を 1 つのセル値に限定することができます。

JUN02、TENTS および SEATTLE の売上をフェッチするには、次の QDR を使用します。

```
SHOW sales(month 'JUN02', product 'TENTS', district 'SEATTLE')
```

このコマンドでは、1 つの値がフェッチされます。

= コマンドのターゲット式には修飾データ参照を使用できます。これによって、データ・オブジェクトの特定のセルに値を代入できます。

次の例では、修飾データ参照に指定されている sales 複合ディメンションのデータ・セルに、値 10200 が代入されます。sales という名前の複合ディメンションに BOSTON と TENTS の組合せの値が存在しない場合、この値の組合せが複合ディメンションに追加され、データ・セルも追加されます。

```
sales(market 'BOSTON' product 'TENTS' month 'JAN99')= 10200
```

変数のディメンションの置換

QDR の修飾子としてリレーションを使用する場合は、変数のディメンションをリレーションの（複数の）ディメンションと置換します。リレーションは、修飾するディメンションに関連し、置換ディメンションによってディメンション化される必要があります。

例 4-1 変数のディメンションの置換

2 つの変数 sales および quota が存在し、それらが month、product および district によってディメンション化されていると想定します。3 番目の変数 division.mgr は、month および division によってディメンション化されています。また、division と product の間には、division.product という名前のリレーションが存在します。これらのオブジェクトの定義は次のとおりです。

```
DEFINE SALES VARIABLE DECIMAL <MONTH PRODUCT DISTRICT>
LD Sales Revenue
DEFINE QUOTA VARIABLE DECIMAL <MONTH PRODUCT DISTRICT>
DEFINE DIVISION.MGR VARIABLE TEXT <MONTH DIVISION>
DEFINE DIVISION.PRODUCT RELATION DIVISION <PRODUCT>
LD DIVISION for each PRODUCT
```

次のコマンドを実行すると、次のレポートが生成されます。

```
REPORT division.mgr

------------------DIVISION.MGR----------------------
          ----------------------MONTH--------------------------
DIVISION  JAN02    FEB02    MAR02    APR02    MAY02    JUN02
--------  -------- -------- -------- -------- -------- --------
CAMPING   Hawley   Hawley   Jones    Jones    Jones    Jones
SPORTING  Carey    Carey    Carey    Carey    Carey    Musgrave
CLOTHING  Musgrave Musgrave Musgrave Musgrave Musgrave Wong
```

売上が割当て制限を超える部分を示すレポートを取得し、該当する部門マネージャ名も各製品別にレポートすると想定します。division に関連し、product によってディメンション化される division.product リレーションを修飾子として使用すると、各製品の部門マネージャを示すことができます。QDR によって division ディメンションが product に置換されます。これによって、「sales / quota」レポートの他の式と同じディメンションが含まれます。次のコマンドを実行すると、次のレポートが生成されます。

```
REPORT DOWN month sales W 6 sales/quota W 8 HEADING -
   'MANAGER' division.mgr(division division.product)

DISTRICT: BOSTON
        ------------------------------PRODUCT-------------------------------------
        ----TENTS---- ---CANOES---- --RACQUETS---  --SPORTSWEAR-- ---FOOTWEAR---
        SALES/        SALES/        SALES/         SALES/         SALES/
MONTH   QUOTA MANAGER QUOTA MANAGER QUOTA MANAGER  QUOTA MANAGER  QUOTA  MANAGER
------  ----- ------- ----- ------- ----- -------  ----- -------- ----- --------
JAN02   1.00  Hawley  0.82  Hawley  1.02  Carey    0.91  Musgrave 0.92  Musgrave
FEB02   0.84  Hawley  0.96  Hawley  1.00  Carey    0.80  Musgrave 1.07  Musgrave
MAR02   0.87  Jones   0.95  Jones   0.87  Carey    0.88  Musgrave 0.91  Musgrave
APR02   0.91  Jones   0.93  Jones   0.99  Carey    0.94  Musgrave 0.95  Musgrave
.
.
.
```

リレーションの修飾

QDR を使用して、リレーション（特殊な変数）を修飾することもできます。

region.district リレーションが district でディメンション化されていると想定します。値 SEATTLE で district を修飾する場合、式の値は SEATTLE のリレーション値となります。QDR によって指定される district の値は 1 つであるため、式の結果は単一セルになります。

region.district の定義は次のとおりです。

```
DEFINE REGION.DISTRICT RELATION REGION <DISTRICT>
LD The region for each district
```

次のコマンドを実行すると、値 WEST が表示されます。

```
SHOW region.district(district 'SEATTLE')
```

ディメンションの修飾

QDR を使用すると、ディメンション自体を修飾し、それによって式のディメンション値を 1 つ指定できます。次の式では、district の値（単一セルの変数 mydistrict に含まれる値）を 1 つ指定します。

```
district(district mydistrict)
```

連結ディメンションの場合、QDR を使用して、その連結ディメンションの 1 つのベース・ディメンションの値を指定することによってディメンションを修飾できます。次の式では、reg.dist.ccdim の値（ベース・ディメンションとして region および district を持つ連結ディメンション）を 1 つ指定します。costs 変数は、division ディメンションおよび reg.dist.ccdim ディメンションによってディメンション化されています。

```
show reg.dist.ccdim(district 'BOSTON')
```

この式によって生成される結果は次のとおりです。

```
<DISTRICT: BOSTON>
```

QDR でのアンパサンド置換の使用

式の先頭にアンパサンド文字（&）を指定すると、コマンドまたはファンクション内で、その式の値がその式自体のかわりに使用されます。アンパサンドを QDR とともに使用し、置換の前に変数を修飾する場合、式全体をカッコで囲む必要があります。

reptype によってディメンション化され、変数の名前を含む myvar という名前のテキスト変数が存在すると想定します。reptype によってディメンション化されるのは myvar であり、myvar によって指定された変数ではないことに注意してください。このため、myvar が修飾され、結果の値が REPORT コマンドで使用されるように、カッコを使用する必要があります。

```
REPORT &(myvar(reptype 'ACTUAL'))
```

カッコを使用せずに、myvar に指定した変数が sales であると想定すると、sales は reptype によってディメンション化されていないことを示すエラー・メッセージが表示されます。

QUAL ファンクションを使用した QDR の指定

QDR の構文があいまいであるため、誤って解釈されたり、構文エラーの原因になる場合があります。この場合、QUAL ファンクションを使用して、修飾データ参照（QDR）を明示的に指定できます。

例 4-2 QUAL ファンクションの使用

次の例では、ディメンションを制限してデータを表示する方法を最初に示し、QUAL を使用して表示する方法を次に示します。

次のコマンドを実行すると、次に示すレポートが生成されます。

```
LIMIT month TO 'JAN96' TO 'JUN96'
LIMIT line TO 'COGS'
LIMIT division TO 'SPORTING'
REPORT DOWN month W 11 MAX(actual,budget) W 11 actual W 11 budget
```

```
DIVISION: SPORTING
                ---------------LINE----------------
                ---------------COGS----------------
                MAX(ACTUAL,
MONTH             BUDGET)      ACTUAL      BUDGET
------------- ------------ ----------- -----------
JAN96            287,557.87  287,557.87  279,773.01
FEB96            323,981.56  315,298.82  323,981.56
MAR96            326,184.87  326,184.87  302,177.88
APR96            394,544.27  394,544.27  386,100.82
MAY96            449,862.25  449,862.25  433,997.89
JUN96            457,347.55  457,347.55  448,042.45
```

ここでは、line または division のステータスを変更せずに、MAX(actual,budget) に同じ数字を表示する方法を示します。

```
ALLSTAT
LIMIT month TO 'JAN96' TO 'JUN96'
REPORT HEADING 'For Cogs in Sporting Division' DOWN month -
   W 11 HEADING 'MAX(actual,budget)'-
   QUAL(MAX(actual,budget), line 'COGS', division 'SPORTING')

For Cogs in
Sporting        MAX(ACTUAL,
Division          BUDGET)
------------- ------------
JAN96            287,557.87
FEB96            323,981.56
MAR96            326,184.87
APR96            394,544.27
MAY96            449,862.25
JUN96            457,347.55
```

標準の QDR 構文で同じレポートを生成しようとすると、エラーが通知されます。

```
REPORT HEADING 'For Cogs in Sporting Division' DOWN month -
   W 11 HEADING 'MAX(actual,budget)'-
   MAX(actual,budget) (line cogs, division sporting)
```

次のエラー・メッセージが表示されます。

```
ERROR: A right parenthesis or an operator is expected after LINE.
```

式でのワークスペース・オブジェクトの使用

オブジェクトは、次のように式で使用できます。

- ディメンション、ディメンション・サロゲート、リレーションまたは変数 : そのオブジェクトの名前を指定することによって、データの配列として使用できます。
- フォーミュラまたはファンクション : その名前を指定することによって、コマンドまたはファンクションの副式または式として使用できます。
- 値セット : その名前を指定することによって、ディメンション値のリストとして式で使用できます。
- 様々なデータ・オブジェクト : 代入文でターゲット式またはソース式として使用できます。

式でのディメンションまたはディメンション・サロゲートの使用

式では、ディメンションまたはディメンション・サロゲートは 1 次元の配列として参照されます。

ディメンションまたはディメンション・サロゲートのデータ型が TEXT であるとき、多くの場合、それらの値はテキスト値として参照されます。NUMBER ディメンション値は、常に値自体によって参照されます。

ただし、NUMBER 以外のディメンションでは、次のいずれかの操作を実行すると、値はディメンション配列内の位置番号で参照されます。

- データ型が TEXT のディメンションを数式で使用します。
- ディメンションのある値を同じディメンションの別の値と比較します。

この場合、整数の位置番号は、現行のステータスではなく、デフォルトのステータス・リストに基づきます。

式での複合ディメンションの使用

式では、複合ディメンションがディメンションとほぼ同様に動作します。通常、式でディメンションを使用可能なすべての場所で、複合ディメンションを使用できます。

- 複合ディメンションが名前付きである場合、その名前を指定します。
- 複合ディメンションが名前なしである場合、SPARSE <*dimensions...*> を指定します。

式での変数の使用

式では、指定されたデータ型の値を含む配列として、変数が参照されます。

変数に値を代入する場合、または変数のディメンションをループ処理する REPORT あるいは他のコマンドやファンクションを使用する場合、変数の最も速く変化するディメンションが最初に変化します。たとえば、month および city によってディメンション化されている opcosts 変数の場合、REPORT コマンドの出力として変数を表示すると、2 番目の都市に関するデータの前に、最初の都市に関するすべての月のデータが表示されます。この場合、month は、その値が city の値の前に変化するため、最も速く変化するディメンションとなります。多次元変数をこのようにループ処理させるプログラムを作成する場合、最も速く変化するディメンションを内部ループと一致させて、パフォーマンスを最適化します。

注意： 変数をモデルのソリューション変数として使用する際、ソリューション変数の定義でのディメンションの順序が、モデルの DIMENSION コマンドのディメンションの順序と一致すると、モデルは最も効率的に実行されます。

修飾データ参照（QDR）を使用して変数の各ディメンションから 1 つの値を指定することによって、多次元変数内の任意のデータ項目を、一意かつ完全に選択できます。

たとえば、opcosts 変数が month および city によってディメンション化されている場合、month ディメンションに 'JAN02'、city ディメンションに 'BOSTON' を指定すると、変数の単一のセルが一意に指定されます。

複合ディメンションで定義した変数の式での使用

多くの場合、複合ディメンションで定義される変数をファンクションおよびコマンドに含めると、これらの変数は、ファンクションおよびコマンドによって、ベース・ディメンションで定義された変数と同様に処理されます。

- いずれかのベース・ディメンション値を要求することで、複合ディメンションによってディメンション化された変数にアクセスできます。
- **ステータス内**の複合ディメンションの値は、複合ディメンションのベース・ディメンションのステータスによって決定されます。複合ディメンションはディメンションではないため、固有のステータスは存在しません。

変数をループ処理するコマンドのデフォルトの動作

REPORT コマンド、または複合ディメンションを使用する変数をループ処理させる他の任意のコマンドを使用する場合、デフォルトの動作では、ステータスに含まれる複合ディメンションのベース・ディメンション値のすべての組合せが評価されます。複合ディメンションに存在しない組合せの場合、それに関連するデータは NA で表示されます。

たとえば、次のコマンドを実行すると、2002 年の 1 月から 3 月までにスポーツウェア向けに発行されたクーポン券の数を示す、東地域に関するレポートが作成されます。2003 年 3 月にはクーポンが発行されなかったため、レポート内の該当する列には NA 値が表示されます。

```
LIMIT month TO 'JAN02' 'FEB02' 'MAR02'
LIMIT market TO 'EAST'
LIMIT product TO 'SPORTSWEAR'
REPORT coupons

MARKET: EAST
                ------------COUPONS-------------
                ------------MONTH--------------
PRODUCT           JAN02      FEB02      MAR02
--------------- ---------- ---------- ----------
SPORTSWEAR          1,000      1,000         NA
```

ただし、パフォーマンスを向上するために、REPORT、ROW、= などのコマンドのデフォルトのループ動作を変更し、それらのコマンドが、すべてのベース・ディメンション値ではなく、複合ディメンションの値をループ処理するようにできます。

式でのリレーションの使用

リレーションは、多くの点で、特別なタイプの変数です。リレーションには、通常のデータ値ではなく、関連ディメンションの値が含まれます。したがって、式では、リレーションはある意味では変数のように動作し、ある意味ではディメンションのように動作します。

- テキスト式でリレーションを使用する場合、リレーション値はテキスト値として参照されます。リレーションに含まれる関連ディメンションの値がテキストに変換され、これらの値を式で使用できます。テキスト・リテラルをリレーションと比較することもできます。
- 数式でリレーションを使用する場合、リレーション値はその関連ディメンションの配列におけるその位置（整数）で参照されます。この数値を式で使用できます。位置番号は、ディメンションの現行のステータス・リストではなく、ディメンションのデフォルトのステータス・リストに基づきます。

式でのファンクションの使用

ファンクションは、値を戻す事前定義済の計算です。次のような多くの組込みファンクションが提供されます。

- 数値ファンクション : データの計算および分析に使用できます。
- 日付ファンクション : 日付の操作に使用できます。
- テキストファンクション : 文字または行の結合、文字グループの検索または抽出、テキストの長さの計算に使用できます。

数式

数式は、任意の数値データ型（INTEGER、SHORTINTEGER、DECIMAL、SHORTDECIMAL および NUMBER）のデータに評価されます。数式のデータは、次のいずれかの組合せになります。

- 数値リテラル
- 数値変数またはフォーミュラ
- ディメンション
- 数値結果を生成するファンクション
- 日付リテラル、変数、フォーミュラまたはファンクション

さらに、より複雑な数式では、算術演算子とこれらの 3 つの部分を持つ任意の式を結合できます。算術演算子は、数値データを含む数式で使用して、結果を数値で戻すことができます。一部の算術演算子は、結果を日付か数値のいずれかで取得する、日付データと数値データが混在した日付式でも使用できます。

算術演算子

次の表に、OLAP DML の算術演算子を示します。数式で 2 つ以上の演算子を使用する場合、標準の算術規則に従って式が評価されます。「優先順位」とタイトルの付いた列は、演算子を評価する順序を示します。優先順位が同じ演算子は左から右に評価されます。

表 4-2 算術演算子

演算子	操作	優先順位
-	記号反転	1
**	指数	2
* および /	乗算および除算	3
+ および -	加算および減算	4

注意： 別の数式の後に続く負数の前には、カンマが必要です。カンマを付けないと、マイナス記号は減算演算子と解釈されます。たとえば、`intvar,-4` とします。

数値データ型の混在

1 つの数式には、すべての数値データ型を含めることができます。

次の規則に従って、結果のデータ型が決定されます。

条件	結果
式のすべてのデータが INTEGER または SHORTINTEGER で、加算、減算および乗算操作のみ実行する場合	INTEGER
いずれかのデータが NUMBER である場合	NUMBER
いずれかのデータが DECIMAL または SHORTDECIMAL で、NUMBER のデータが存在しない場合	DECIMAL
除算または指数操作を実行する場合	DECIMAL

数値データ型の自動変換

数値は、次の規則に従って異なるデータ型に変換されます。

操作	結果
SHORTINTEGER または SHORTDECIMAL データ型の値を式で使用する場合	値は長い方に変換されて使用されます。 **注意 :** SHORTDECIMAL と DECIMAL データ型を比較式で混在させる場合に発生する可能性がある問題については、4-18 ページの「ブール式」を参照してください。
計算結果を SHORTINTEGER データ型の値で保存する場合	結果が SHORTINTEGER の範囲（-32768 ～ 32767）外である場合は、NA が格納されます。
DECIMAL 式の値を INTEGER データ型のオブジェクトに代入する場合	値は丸められてから、格納または使用されます。 **注意 :** 小数値が INTEGER の範囲外（約± 20 億）である場合、NA が格納されます。
INTEGER データ型の値が必要な場合に小数値を使用する場合	値は丸められてから、格納または使用されます。 **注意 :** 小数値が INTEGER の範囲外（約± 20 億）である場合、NA が格納されます。
DECIMAL 式の値を SHORTDECIMAL データ型の変数に代入する場合	有効桁の最初の 7 桁のみが格納されます。
NUMBER 値を他の数値データ型と組み合せる場合	すべての値が NUMBER に変換されます。

目的の変換が行われない場合、CONVERT、TO_CHAR、TO_NCHAR、TO_NUMBER または TO_DATE ファンクションを使用して異なる結果を得ることができます。

算術式でのディメンションの使用

数式でデータ型が TEXT のディメンションを使用する場合、ディメンション値は位置（整数）として処理され、数値として使用されます。位置番号は、現行のステータスではなく、デフォルトのステータス・リストに基づきます。

算術式での日付の使用

算術式で日付を使用すると、その結果は数値または日付になります。次の表に、日付の有効な操作および結果のデータ型を示します。

操作	結果
日付に数値を足す、または日付から数値を引く場合	将来または過去の日付。
日付から日付を引く場合	その間の日数。
日付に日付を足す場合	エラー。これは無効な操作です。
ある期間に数値を足す、またはその期間から数値を引く場合	LEAD または LAG ファンクションでの処理と同様に、将来または過去の適切な期間。結果に対応するディメンション値が存在しない場合、結果は NA となります。ディメンションのデフォルトのステータス・リストにおける値の位置に基づいて計算されます。

浮動小数点の計算の制限

すべての小数値データは浮動小数点形式に変換され、格納および計算されます。浮動小数点形式では、数値は仮数および指数によって表されます。仮数および指数は、2 進数として格納されます。仮数とは、2 と等価の数値によって指数計算される場合、元の小数値と等価またはほぼ等しい数値を生成する 2 進部分です。

3 分の 1 が小数で正確に表現できないように、小数値の端数部分は常に正確に 2 進表現されるとはかぎりません。浮動小数点数の算術演算子の算出する結果はさらに概算となり、操作数が増えるとともに不正確さが増加します。概算要因のみではなく、使用可能な有効桁数が、結果の正確性に影響します。

これらのすべての理由から、TOTAL、AVERAGE、またはその他の DECIMAL や SHORTDECIMAL 変数の集計ファンクションで算出した結果と、手動で計算した結果を比較すると、最後の有効数字が異なる場合があります。SHORTDECIMAL データ型は最大 7 桁の有効数字しか提供できないため、このような SHORTDECIMAL データとの違いはさらに大きくなります。そのため、通貨の金額を含む変数など、計算速度より精度が重要である場合は、NUMBER データ型の使用が必要な場合があります。

また、2 進小数部で正確に表現できない端数小数値については、有効数字の桁数が多い DECIMAL データ型を使用することをお薦めします。このデータ型は、SHORTDECIMAL データ型とは違った、より近似の概算を提供できます。このような理由から、SHORTDECIMAL データ型と DECIMAL データ型が比較式で混在する場合に問題が発生します。比較を処理する方法の詳細は、4-18 ページの「ブール式」を参照してください。

計算過程におけるエラーの制御

次のタイプのエラーを制御できます。

- 0（ゼロ）による除算 : NA 値を 0 で割ると、その結果は NA となり、エラーは発生しません。NA 以外の値を 0 で割ると、通常はエラーになります。ディメンション化されたデータに対して計算している場合に 0 による除算のエラーが発生すると、部分的な結果しか戻されない場合があります。REPORT または = コマンドを使用すると、値は計算されるたびにレポートまたは格納されます。そのため、0 で割った場合、ループはすべての値を計算する前に停止します。

  0 による除算エラーを防ぐには、DIVIDEBYZERO オプションの値を YES に変更します。これによって、0 による除算のすべての結果は NA となり、エラーは発生しません。これによって、ディメンション化された式のその他の値が継続して計算されます。

- 負数のルート計算 : 負数のルート計算を行う（または、数値を整数以外の値で累乗する）と、通常はエラーになります。エラーを防ぎ、継続して式の負数値以外のルートを計算できるようにするには、ROOTOFNEGATIVE オプションを YES に設定します。

- オーバーフロー・エラー : DECIMALOVERFLOW オプションは、DIVIDEBYZERO と同様に動作します。これによって、計算で処理できないほど大きい小数値結果が生成された場合、エラーを生成するかどうかを制御できます。

テキスト式

テキスト式は、TEXT、NTEXT または ID のいずれかのデータ型のデータに評価されます。テキスト式では、次のいずれかの組合せが可能です。

- テキスト・リテラル（BOSTON や Current Sales Report など）。
- テキスト・ディメンション（district や month など）。
- テキスト変数またはフォーミュラ（product.name など）。
- 結果がテキストとなるファンクション（ JOINLINES('Product: 'product.name) など）。

textvar は、その値がディメンションの名前 geog である変数であると想定します。textvar という用語を引用符で囲むかどうかによって、次の OBJ ファンクションで VARIABLE（オブジェクト・タイプが textvar）という用語が戻されるか、DIMENSION（オブジェクト・タイプが geog）という用語が戻されるかが決まります。

```
SHOW OBJ(TYPE 'textvar')
VARIABLE

SHOW OBJ(TYPE textvar)
DIMENSION
```

テキスト式での日付の処理

テキスト値（TEXT、NTEXT または ID）が必要な位置で DATETIME 値を使用した場合、または、テキスト変数に DATETIME 値を格納した場合、DATETIME 値がテキスト値に自動的に変換されます。

DATETIME 値の形式は、NLS_DATE_FORMAT オプションで制御できます。DATETIME 値をテキスト値で格納すると、NLS_DATE_FORMAT 設定では制御できなくなります。

NTEXT データの処理

TEXT データと NTEXT データは、多くの場合、入れ替えて使用できます。ただし、NTEXT 値が TEXT 変数に代入された場合などは、明示的な変換が実行される場合があります。TEXT が NTEXT に変換される場合、NTEXT データ型の UTF-8 キャラクタ・コードには他のほぼすべてのデータ型が含まれるため、データが消失することはありません。ただし、NTEXT が TEXT に変換される場合、アナリティック・ワークスペースのキャラクタ・セットに NTEXT が含まれていない場合、データは消失します。

TEXT 値と NTEXT 値を同時に使用する場合（JOINCHARS ファンクションへのコール内など）、TEXT 値は NTEXT に変換され、NTEXT 値が戻されます。

ブール式

ブール式は、true または false の論理文です。ブール式を使用する場合、比較する式の基本データ型が同じであれば、すべての型のデータを比較できます。データをテストして、他方のデータと等価か、より大きいか、またはより小さいかを調べることができます。

ブール式は、次のようなブール・データで構成できます。

- ブール値（YES と NO、そのシノニム ON と OFF、TRUE と FALSE）
- ブール変数またはフォーミュラ
- ブール結果を生成するファンクション
- 比較演算子によって計算されるブール値

たとえば、次のようなブール式では、変数 actual の各値が定数 20,000 と比較されます。値が 20,000 より大きい場合、文は true となり、値が 20,000 以下の場合、文は false となります。

```
actual GT 20000
```

ブール値を指定する場合、真の値には yes、on、true のいずれか、偽の値には no、off、false のいずれかを指定できます。ブール計算の結果が生成される場合、デフォルトは、NLS_LANGUAGE オプションで指定した言語での yes および no です。読込み専用の YESSPELL および NOSPELL オプションを指定すると、yes 値および no 値を記録できます。

次の表に、比較演算子および論理演算子を示します。これらの演算子を使用すると、算術演算子とほぼ同じ方法で式を作成できます。「優先順位」とタイトルの付いた列は、演算子を評価する順序を示します。

表 4-3 ブール演算子

演算子	操作	例	優先順位
NOT	ブール式の反対値を戻す	NOT(yes) = no	1
EQ	等しい	4 EQ 4 = yes	2
NE	等しくない	5 NE 2 = yes	2
GT	より大きい	5 GT 7 = no	2
LT	より小さい	5 LT 7 = yes	2
GE	以上	8 GE 8 = yes	2
LE	以下	8 LE 9 = yes	2
IN	日付が期間に含まれているかどうか	'1JAN02' IN W1.02 = yes	2
LIKE	テキスト値が、指定したテキスト・パターンに一致するかどうか	'FINANCE' LIKE '%NAN%' = yes	2
AND	どちらの式も true	8 GE 8 AND 5 LT 7 = yes	3
OR	どちらか 1 つの式が true	8 GE 8 OR 5 GT 7 = yes	4

演算子にはそれぞれ優先順位があり、それに従って評価順序が決定されます。演算子の優先順位が等しい場合は、左から右に評価されます。ただし、カッコが含まれている場合は評価の順序は変わります。また、真の値が途中で決定されると評価は停止します。たとえば、次の式では最初の句で式全体が true であることが判断されるため、TOTAL ファンクションは実行されません。

```
yes EQ yes OR TOTAL(sales) GT 20000
```

ブール式の作成

ブール式は、比較演算子で区切られた 2 つの比較項目で構成される、3 つの部分を持つ句です。AND および OR 論理演算子によって、3 つの部分を持つ式を結合して、より複雑なブール式を作成できます。AND または OR によって結合されているそれぞれの式は、それ自体完全なブール式である必要があります。そのためには、同じ変数を複数回指定する必要がある場合もあります。

たとえば、次の式は 2 番目の部分が不完全なため、有効ではありません。

```
sales GT 50000 AND LE 20000
```

次の式では、両部分とも完全であるため、式は有効になります。

```
sales GT 50000 AND sales LE 20000
```

複数のブール式を組み合せる際、真の値が式の最初の部分で決定される場合も、式全体として有効である必要があります。式は、その全体がコンパイルされた後に評価されます。そのため、未定義の変数がブール式の 2 番目の部分に含まれている場合、エラーが発生します。

式をカッコで囲み、NOT 演算子を使用すると、ブール式の意味を逆転させることができます。

次の 2 つの式は等価です。

```
district NE 'BOSTON'
NOT(district EQ 'BOSTON')
```

例 4-3 ブール値比較の使用

次の例では、Boston での各製品の売上がリテラル量より大きかったかどうかを表示するレポートを示します。

```
LIMIT time TO FIRST 2
LIMIT geography TO 'BOSTON'
REPORT DOWN product ACROSS time: f.sales GT 7500
```

この REPORT コマンドを実行すると、次のデータが戻されます。

```
CHANNEL: TOTALCHANNEL
GEOGRAPHY: BOSTON
                ---F.SALES GT 7500---
                --------TIME---------
PRODUCT           JAN02      FEB02
--------------- ---------- ----------
PORTAUDIO              no         no
AUDIOCOMP             yes        yes
TV                     no         no
VCR                    no         no
CAMCORDER             yes        yes
AUDIOTAPE              no         no
VIDEOTAPE             yes        yes
```

ブール式での NA 値の比較

ブール式で比較するデータに NA 値が存在する場合、それが意味を持つ場合には YES または NO が結果として戻されます。たとえば、NA 値が NA 以外の値と等価であるかどうかをテストすると、結果は NO となります。ただし、結果が紛らわしい場合は、NA が戻されます。たとえば、NA 値が NA 以外の値より小さいか大きいかをテストすると、NA が結果として戻されます。

次の表に、NA 値を含み、NA 以外の値を生成するブール式の結果を示します。

式	結果
NA EQ NA	YES
NA NE NA	NO
NA EQ non-NA	NO
NA NE non-NA	YES
NA AND NO	NO
NA OR YES	YES

数値データを比較する際のエラーの制御

数値データの比較で予期しない結果が生成された場合、いくつかの原因が考えられます。

- 比較している数値の 1 つに、DECIMALS オプションが設定されているため出力に表示されない小数値下位部分が存在する。
- 2 つの浮動小数点数を比較し、1 つ以上の数が算術操作の結果である。
- 比較において SHORTDECIMAL データ型と DECIMAL データ型を混在させた。

ABS および ROUND ファンクションを使用して近似等価テストを行い、予期しない比較障害の 3 つすべての原因を回避することをお薦めします。ABS または ROUND を使用する場合、絶対差または四捨五入因数を、アプリケーションに合わせて適切な値に調整できます。計算の速度を重視する場合は、ROUND ファンクションではなく ABS ファンクションを使用することをお薦めします。

数値精度によるエラーの制御

expense が小数値変数で、その値が計算によって設定されると想定します。計算結果が 100.000001 で、小数位が 2 に設定されている場合、出力される値は 100.00 です。ただし、次のコマンドの出力としては NO が戻されます。

```
SHOW expense EQ 100.00
```

ABS または ROUND ファンクションを使用すると、これらのわずかな違いを無視して比較できます。

浮動小数点数を比較する際のエラーの制御

コンピュータ言語において浮動小数点数を使用する場合、比較する 2 つの浮動小数点数のいずれかが算術操作の結果で得られた値である際に、比較には正確な均一性が得られないという標準制限があります。たとえば、次のコマンドでは YES が予想されますが、システムによっては NO が戻されます。

```
SHOW .1 + .2 EQ .3
```

小数値データを使用する場合、前述のような直接的な比較のコードを作成しないでください。かわりに、ABS または ROUND ファンクションを使用すると、おおよその均一性を認められる範囲の結果が得られます。たとえば、次の 2 つのコマンドのどちらを実行しても、目的の YES が生成されます。

```
SHOW ABS((.1 + .2) - .3) LT .00001
SHOW ROUND(.1 + .2) EQ ROUND(.3, .00001)
```

異なる数値データ型を比較する際のエラーの制御

DECIMAL および NUMBER データ型は、正確に表現できない端数部分を近似する有効桁数が SHORTDECIMAL より多いため、端数部分を含む小数値については、SHORTDECIMAL と DECIMAL または NUMBER の表現の間には正確な均一性が期待できません。

変数を SHORTDECIMAL データ型で定義し、それを端数小数値に設定して、SHORTDECIMAL 数値をその端数小数値と比較すると想定します。

```
DEFINE sdvar SHORTDECIMAL
sdvar = 1.3
SHOW sdvar EQ 1.3
```

この比較では、通常、NO が戻されます。この場合、リテラルが DECIMAL として自動的に分類され、SHORTDECIMAL 変数 sdvar が DECIMAL に変換されます。これによって、小数位が 0（ゼロ）で拡張されます。その後、ビット単位の比較が行われますが、これは失敗します。DECIMAL または NUMBER データ型の変数を使用した同様の比較では、通常、YES が戻されます。

このような比較エラーは、通常、次の 2 つの方法で回避できます。

- 比較において SHORTDECIMAL データ型と DECIMAL または NUMBER データ型を混在させない。これらの 2 つのデータ型を混在させないようにするには、通常、小数部分を含む変数を SHORTDECIMAL として定義しないようにします。
- ABS または ROUND ファンクションを使用して、おおよその均一性を認めるようにする。次の両方のコマンドでは、YES が生成されます。

  ```
  SHOW ABS(sdvar - 1.3) LT .00001
  SHOW ROUND(sdvar, .00001) EQ ROUND(.3, .00001)
  ```

ディメンション値の比較

同じディメンション内の値は、テキスト値に基づいて比較されることはありません。かわりに、Oracle OLAP は、ディメンションのデフォルト・ステータスにおける値の位置を比較します。このため、次のようなコマンドを指定できます。

```
REPORT district LT 'SEATTLE'
```

コマンドは、次のようなプロセスで解析されます。

1. テキスト・リテラル 'SEATTLE' は、district ディメンションのデフォルトのステータス・リスト内での位置に変換されます。
2. その位置が、district ディメンション内の他のすべての値の位置と比較されます。
3. 次のレポートに示すとおり、district ディメンションのデフォルトのステータス・リスト内の SEATTLE の位置より前に含まれる地区には値 YES が戻され、SEATTLE 自体には NO が戻されます。

```
REPORT 22 WIDTH district LT 'SEATTLE'

DISTRICT       DISTRICT LT 'SEATTLE'
-------------- ----------------------
BOSTON                            YES
ATLANTA                           YES
CHICAGO                           YES
DALLAS                            YES
DENVER                            YES
SEATTLE                            NO
```

より複雑な例では、最初の 6 か月に代入された初期値に基づいて、変数 quota に増分値を代入します。比較は、month ディメンションにおける値の位置に基づいて行われます。これは時間ディメンションであるため、値は時系列になります。

```
quota = IF month LE 'JUN02' THEN 100 ELSE LAG(quota, 1, month)* 1.15
```

ただし、region lt district 式のように異なるディメンションの値を比較する場合、共通の分母は TEXT のみであるため、ディメンションの位置ではなくテキスト値が比較されます。

> **参照：** IF...THEN...ELSE 構文の詳細は、4-25 ページの「条件式」を参照してください。

日付の比較

2 つの日付は、任意のブール比較演算子を使用して比較できます。日付の場合、「より少ない」は「前」を、「より大きい」は「後」を意味します。比較する式には、4-14 ページの「数式」に示すどの日付計算も含めることができます。たとえば、請求アプリケーションで、未払い請求書の支払い勧告を送付するために、請求日付から 60 日が経過したかどうかを判断することができます。

```
bill.date + 60 LE SYSDATE
```

日付には数値も含まれます。TO_NUMBER ファンクションによって日付を整数に、および TO_DATE ファンクションによって整数を日付に変換して比較できます。

テキスト・データの比較

テキスト・データを比較する場合、句読点、空白、大文字、小文字などを含めて、テキストを正確に指定する必要があります。テキスト・リテラルは、一重引用符で囲む必要があります。たとえば、次の式では、各従業員名の最初の文字が文字「M」より後かどうかがテストされます。

```
EXTCHARS(employee.name, 1, 1) GT 'M'
```

TEXT と ID 値は比較できますが、これらの値が同じ長さの場合のみ等価になります。あるテキスト値が他のテキスト値より後か前かをテストする場合、NLS_SORT オプションの設定に基づいた順序が使用されます。

数値とテキストを比較するには、まず数値をテキストに変換します。順序は、文字の値に基づきます。この比較では、テキストが左から右に評価されるため、予期しない結果が生成される場合があります。たとえば、テキスト・リテラル '1234' と '100,999.00' の場合、前者の 2 番目の文字 '2' が後者の 2 番目の文字 '0' より大きいため、'1234' の方が大きいという結果が生成されます。

name.label が、'3-Person' という値を持つ ID 変数であり、name.desc が、'3-Person Tents' という値を持つ TEXT 変数であると想定します。

次の SHOW コマンドの結果は、NO になります。

```
SHOW name.desc EQ name.label
```

次のコマンドの結果は、YES になります。

```
name.desc = '3-Person'
SHOW name.desc EQ name.label
```

テキスト・パターンとテキスト値の比較

ブール演算子 LIKE は、テキスト値とテキスト・パターンとの比較用に設計されています。対応する文字が一致する場合、一方のテキスト値は他方のテキスト値またはテキスト・パターンと部分一致すると評価されます。

LIKE では、リテラル一致に加えて、ワイルドカード文字を使用した、文字列内の複数の文字の一致も検索できます。

- パターン内のアンダースコア（_）文字は、すべての単一文字に一致します。
- パターン内のパーセント（%）文字は、最初のすべての文字（存在する場合）に一致します。

たとえば、%AT_ というパターンでは、先頭に任意の文字が含まれ、その次に AT という文字、その次に任意の数の文字が含まれるすべてのテキストが一致します。LIKE を使用して、%AT_ というパターンと 'DATA' および 'ERRATA' を比較した場合、その両方に対して YES が戻されます。

LIKE 演算子を使用した式の結果は、LIKECASE および LIKENL オプションの設定によって影響されます。これらのオプションによって LIKE 演算子が受ける影響の例と、LIKE 演算子の一般的な使用方法の例については、Oracle9*i* OLAP DML Reference ヘルプを参照してください。

LIKE には、対応する否定演算子が存在しません。否定演算を実行するには、式全体を否定する必要があります。たとえば、次のコマンドの結果は NO になります。

```
SHOW NOT ('BOSTON' LIKE 'BO%')
```

テキスト・リテラルとリレーションの比較

テキスト・リテラルをリレーションと比較することもできます。リレーションには、関連付けられたディメンションの値が含まれています。テキスト・リテラルは、そのディメンションの値と比較されます。たとえば、region.district には region の値が含まれているため、次の比較を行うことができます。

```
region.district EQ 'WEST'
```

条件式

条件式を使用すると、ブール条件に基づいて 2 つの値から 1 つの値を選択できます。条件式には、条件演算子 IF...THEN...ELSE が含まれます。形式は次のとおりです。

```
IF Boolean-expression THEN expression1 ELSE expression2
```

条件式は、データ型が適切である場合、他の式の一部として指定できます。

注意： 条件式と、IF コマンドを混同しないでください。IF コマンドは、構文が類似していても別の目的で使用されます。また、IF コマンドはデータ型を持たず、式のように評価されることはありません。

条件式の処理では、まずブール式が評価されます。その後、次のとおり処理が実行されます。

- ブール式の結果が TRUE の場合、*expression1* が評価され、その値が戻されます。
- ブール式の結果が式のデータ型が FALSE の場合、*expression2* が評価され、その値が戻されます。

expression1 および *expression2* 引数には、同じ基本データ型に評価されるすべての有効な OLAP DML の式を指定できます。ただし、いずれかの値のデータ型が DATE である場合、他方の値に数値データ型またはテキスト・データ型を指定できます。両方のデータ型が DATE である必要があるため、指定した数値またはテキスト値は DATE に変換されます。式全体のデータ型は、これらの 2 つの式と同じです。

ブール式の結果が NA の場合、NA が戻されます。

例 4-4 条件式を使用したレポート

この例では、売上ボーナスのレポートを示します。このボーナスは、売上から予算額を引いた金額の 5% です。その地区の売上が予算未満である場合、ボーナスは 0（ゼロ）です。

```
LIMIT month TO 'JAN02' TO 'JUN02'
LIMIT product TO 'TENTS'
REPORT DOWN district IF sales-sales.plan LT 0 THEN 0
       ELSE .05*(sales-sales.plan)

PRODUCT: TENTS
        ---IF SALES-SALES.PLAN LT 0 THEN 0 ELSE .05*(SALES-SALES.PLAN)---
          ----------------------MONTH-----------------------------
DISTRICT   JAN02    FEB02    MAR02     APR02    MAY02    JUN02
--------- -------- -------- -------- ------- --------- ----------
BOSTON      229.53     0.00     0.00    0.00    584.51     749.13
ATLANTA       0.00     0.00     0.00  190.34    837.62   1,154.87
CHICAGO       0.00     0.00     0.00   84.06    504.95     786.81
.
.
.
```

置換式

置換式を使用すると、コマンドまたはファンクション内で、式自体のかわりにその式の値を指定できます。

置換式を作成するには、式の先頭にアンパサンド文字（&）を指定します。このようにアンパサンド（置換演算子）を使用する置換は、アンパサンド置換といいます。アンパサンドを指定すると、アンパサンドによって式が評価され、結果の値がアンパサンドと置換されてから、その式の残りの部分が評価されます。

アンパサンド置換では、式を指定する際に間接的な処理のレベルを指定できます。たとえば、アンパサンドの後に、他の変数の名前を含む変数を指定すると、その式の値は、その名前を持つ変数に含まれているデータになります。アンパサンド置換を使用すると、プログラムの実行時に選択されたデータを操作可能な、より一般的なプログラムを作成できます。

アンパサンド置換は、モデル方程式内では使用できません。

注意： アンパサンド置換を使用すると、様々な変数やデータを処理可能な一般的なプログラムを作成できますが、アンパサンド置換を指定した行の実行効率は低下します。アンパサンド置換を含む行はコンパイルされず、プログラムの実行時に解析されます。アンパサンド置換を使用するかわりに、IF または SWITCH コマンドを使用できます。

参照： 条件付きコマンドを作成する方法の詳細は、7-12 ページの「実行フローの制御」を参照してください。

例 4-5 アンパサンド置換の使用

アナリティック・ワークスペース（product）内のディメンションの名前を含む curname という名前の変数が存在すると想定します。次のコマンドを実行すると、REPORT によって 1 つの値 product が生成されます。この値は、次に示すとおり §curname に格納されている実際の値です。

```
report curname

CURNAME
----------
PRODUCT
```

ただし、次のコマンドを実行すると、次に示すとおり、REPORT によってディメンション product の値が生成されます。

```
report &curname

PRODUCT
--------------
TENTS
CANOES
RACQUETS
SPORTSWEAR
FOOTWEAR
```

NA 値の処理

指定した操作に使用可能なデータが存在しない場合があります。たとえば、ファンクションの戻り値、または算術演算子を含む式の値として、変数の指定したセルに適切な値が存在しない場合があります。この場合、NA（使用不可）値が自動的に指定されます。

NA は、特定のデータ値が代入されていない、またはデータを計算できない任意のセルの値です。NA 値は、特定のデータ型を持ちません。

特定のファンクション（集計ファンクションなど）は、そのファンクションによって要求された情報が使用できないかまたは計算できないときに、NA 値を戻します。同様に、ある式で値が計算できなかった場合、値は NA になります。

変数またはリレーションの値を NA に設定するには、次の例に示すとおり、= コマンドを使用します。

```
sales = NA
```

sales がディメンション化された変数である場合、= コマンドによって sales のすべての値がループ処理され、それらの値が NA に設定されます。

NA 値の処理方法の制御

次のオプションおよびファンクションを使用すると、式での NA 値の処理方法を制御できます。

- PROPERTY コマンドを使用すると、ディメンション化された変数の NATRIGGER プロパティの値を設定し、NA 値を含む変数のセルが読み込まれたときに、NA 値が NATRIGGER 式の値に置換されるようにできます。この置換によって、特定の種類の計算効率が向上します。また、いくつかのフォーミュラ・オブジェクトを省略できます。
- 次のオプションを使用すると、集計ファンクション、および加算（+）操作や減算（-）操作を含む算術操作での NA 値の処理方法を制御できます。
  - NASKIP オプションは、集計ファンクションでの NA 値の処理方法を制御します。

  - NASKIP2 オプションは、加算（+）操作や減算（-）操作を含む算術操作での NA 値の処理方法を制御します。
- NAFILL ファンクションは、ソース式の値に、指定された充填表現として NA 値を埋め込んで戻します。このファンクションを式に含めて、その値の形式を制御することができます。

NATRIGGER プロパティの処理

NATRIGGER プロパティ式は、NAFILL ファンクション、あるいは NASKIP オプション、NASKIP2 オプションまたは NASPELL オプションが適用される前に評価されます。NATRIGGER 式が NA である場合、NAFILL ファンクションおよび NA オプションは影響を受けます。また、NATRIGGER プロパティによって、NA 値をより柔軟に処理できるようになります。

- 値式内にトリガー・オブジェクトを指定して、NA トリガーを再帰的または相互に再帰的にすることができます。フォーミュラ、プログラム、または他の NATRIGGER 式の実行中にトリガー式が実行されるようにするには、RECURSIVE オプションを yes に設定する必要があります。同時に実行可能なトリガーの数を制限する方法の詳細は、TRIGGERMAXDEPTH オプションを参照してください。
- 変数のセル内の NA 値は、TRIGGERSTOREOK オプションを yes、およびその変数の STORETRIGGERVAL プロパティを yes に設定することによって、NATRIGGER 式の値に置き換えることができます。

ROLLUP と AGGREGATE コマンドおよび AGGREGATE ファンクションは、ロールアップ操作または集計操作中は、変数の NATRIGGER プロパティの設定を無視します。また、変数の NATRIGGER プロパティ式は、その変数が EXPORT TO EIF コマンドによって単にエクスポートされた場合は評価されません。NATRIGGER プロパティ式は、その変数が、エクスポート操作中に計算された式の一部である場合にのみ評価されます。

NASKIP の使用

NASKIP オプションは、集計ファンクションでの NA 値の処理方法を制御します。

- デフォルトでは、NASKIP オプションは YES に設定され、NA 値は集計ファンクションでは無視されます。実際の値を含む式のみが、計算に使用されます。
- NASKIP オプションを NO に設定した場合、NA 値は集計ファンクションへの入力として処理されます。処理される値のいずれかが NA である場合、ファンクションは、その値に対して NA を戻します。

NASKIP を NO に設定すると、データ内に NA 値が存在するため計算自体が無効になる場合などに有効です。たとえば、MOVINGMAX ファンクションを使用する場合、最大値を選択する範囲を設定します。

- NASKIP が YES（デフォルト）である場合、範囲内のすべての値が NA である場合にのみ、MOVINGMAX は NA を戻します。

- NASKIP が NO であり、範囲内のいずれかの値が NA である場合、MOVINGMAX は NA を戻します。

NASKIP2 の使用

NASKIP2 オプションは、加算（+）操作や減算（-）操作を含む算術操作での NA 値の処理方法を制御します。

- デフォルトでは、NASKIP2 オプションの値は NO に設定されています。NA 値は、加算（+）操作や減算（-）操作を含む算術操作では NA として処理されます。処理されるオペランドのいずれかが NA である場合、その算術操作は NA に評価されます。たとえば、デフォルトでは、2+NA の結果は NA になります。
- NASKIP2 オプションの値を YES に設定した場合、加算（+）操作や減算（-）操作を含む算術操作では、NA 値のかわりに 0（ゼロ）が使用されます。NA+NA および NA-NA の場合は特別で、いずれも結果は NA になります。

NAFILL の使用

NASKIP と NASKIP2 は、データを変更することはありません。これらのオプションでは、データの計算結果のみが影響を受けます。任意の式に影響を限定し、データを実際に変更する必要がある場合は、NAFILL を使用します。

NAFILL ファンクションの影響は、指定した 1 つの式のみに制限されます。ファンクションや加算（+）操作または減算（-）操作のみでなく、すべての式に指定できます。また、NAFILL を使用すると、その式内の NA のかわりに、0（ゼロ）のみでなく様々な要素を使用できます。さらに、代入文を使用すると、NAFILL を使用して、データ内の NA を永続的に置換できます。

NAFILL では、指定した式の値が戻されます。ただし、その値が NA である場合、NAFILL によって、指定した置換値が戻されます。

次のコマンドでは、NAFILL を使用して sales 変数内の NA 値を数値 1 に置換し、それらの値をその変数に代入します。これによって、その置換値がデータ内で永続的に使用されるようになります。

```
sales = NAFILL(sales, 1)
```

次のコマンドは、より特別な目的のための NAFILL の使用を示しています。この例の NAFILL では、NA 値の代入値として 0（ゼロ）を指定することで、AVERAGE ファンクションが平均値を求める値の数を計算するときに NA 値も含まれるように強制的に指定しています。この置換は一時的であり、このコマンドの継続期間中のみ有効です。

```
SHOW AVERAGE(NAFILL(sales 0.0) district)
```

5

アナリティック・ワークスペースのデータ・オブジェクトの移入

この章では、ソース・データを含むアナリティック・ワークスペースのデータ・オブジェクトを移入する方法、および変数に計算値を移入する方法について説明します。この章では、次の項目について説明します。

- アナリティック・ワークスペースの移入の概要
- ディメンションおよび複合ディメンションのメンテナンス
- データ・オブジェクトへの値の代入
- データの計算および分析

アナリティック・ワークスペースの移入の概要

アナリティック・ワークスペースを使用するには、その中にデータが存在する必要があります。データには、ファクト・データとディメンションという 2 つの基本的なタイプが存在します。ファクト・データは **variable** アナリティック・ワークスペース・オブジェクトに格納され、ディメンション値を含むディメンションは **dimension** アナリティック・ワークスペース・オジェクトに格納されます。

変数とディメンションは、次の操作によって移入できます。

- リレーショナル表からのデータのロード。たとえば、売上ファクト表から売上ファクト・データ、時間ディメンション表から時間ディメンション値、顧客ディメンション表から顧客ディメンション値、製品ディメンション表から製品ディメンション値を、それぞれ変数にロードできます。
- 計算の結果として。たとえば、売上予測変数に、予測ファンクションの結果を移入できます。
- OLAP DML を介して制御されるデータ・ローダーを使用した、フラット・ファイルからのデータのロード。
- 手動。ただし、この方法は通常、少数の値の入力のみに使用します。

データ・オブジェクトをアナリティック・ワークスペースへ明示的に移入するには、次の手順を実行します。

1. 各ディメンションに値を指定します。これらの値は、実際のデータへの索引として使用されます。これは、アナリティック・ワークスペース変数に格納されます。
2. 各リレーションに値を指定します。これらの値は、ディメンション間の関係を示します。
3. アプリケーションにソース・データを提供する変数には、実際のデータ値を指定します。

アナリティック・ワークスペースは、SQL コマンドおよびデータ・ロード・コマンドを使用して作成されたプログラムを使用して移入できます。次の表に、データ・オブジェクトの移入に一般的に使用する OLAP DML のコマンドを示します。

コマンド	説明
= または SET	式の結果を変数、オプションまたはリレーションに代入します。詳細は、5-10 ページの「データ・オブジェクトへの値の代入」および 8-2 ページの「モデルを使用したデータの計算」を参照してください。
MAINTAIN	ディメンションの値の追加、削除、名前の変更、移動またはマージ、および複合ディメンションの値の追加、削除およびマージを行います。詳細は、5-3 ページの「ディメンションおよび複合ディメンションのメンテナンス」を参照してください。
FILEREAD	入力ファイルからディメンション、複合ディメンション、リレーションまたは変数に読み込まれたデータを格納します。詳細は、第 11 章「ファイルからのデータの読込み」を参照してください。
SQL	リレーショナル表からデータを取得し、ディメンションまたは変数に挿入します。詳細は、第 10 章「リレーショナル表の処理」を参照してください。
IMPORT	アナリティック・ワークスペースのデータおよび定義を EIF ファイルからコピーします。

ディメンションおよび複合ディメンションのメンテナンス

アナリティック・ワークスペースを移入する最初の手順は、アナリティック・ワークスペースのディメンションに値を格納することです。格納されたディメンション値のリストを、そのディメンションのデフォルトのステータス・リストといいます。アナリティック・ワークスペースを初めてアタッチする場合、デフォルトのステータス・リストが各ディメンションの現行のステータス・リストになります。

MAINTAIN コマンドを使用すると、単純なディメンション、複合ディメンションまたは結合ディメンションの値の追加、削除、マージ、位置の変更、変更、および連結ディメンション値の位置の変更を行うことができます。ディメンションの値を格納および操作することを、ディメンションのメンテナンスといいます。

ディメンションのメンテナンスによるディメンションのステータスへの影響

次の表に示すとおり、MAINTAIN コマンドを使用すると、ディメンションのステータスが影響を受ける場合があります。

MAINTAIN コマンドを使用する条件	結果
ADD、DELETE、MERGE または MOVE キーワードを使用し、ディメンションの現行のステータスが ALL ではない場合	要求されたメンテナンスをコマンドが実行する前に、ディメンションのステータスが ALL にリセットされます。
ディメンションに送信されたステータス・リスト（PUSH コマンドを使用して作成されたステータス・リスト）が存在する場合	ディメンションの送信されたステータス・リストが消去され、そのディメンションが影響を受けなかったことが表示されます。

ディメンションのステータスを表示および送信する方法の詳細は、6-2 ページの「ディメンションのステータスの概要」を参照してください。

遅延メンテナンスの回避

ディメンションをメンテナンスする場合、そのディメンションによってディメンション化されているオブジェクトを変更する必要があります。これらのオブジェクトは、メモリー内に存在する場合は即時に変更されますが、メモリー内に存在しない場合は、メモリーにロードされるまでメンテナンスが遅延されます。

多くのディメンションのメンテナンスを行い、最後に大規模な更新を行う必要がある場合は、遅延メンテナンスによってエラーが発生する場合があります。例として、MAINTAIN DELETE ALL コマンドの発行や、大量の値がディメンションに追加されるデータのロードの実行などがあります。このようなプロジェクトを開始する前に、対象のディメンションを使用するオブジェクトをメモリーにロードし、遅延メンテナンスが実行されないようにしておくことをお薦めします。これを行うには、次のようなコマンドを実行します。サンプル・ディメンションは product です。

```
LIMIT NAME TO OBJ(ISBY product)
LOAD &values(NAME)
MAINTAIN product ADD ...
```

ディメンションへの値の追加

ディメンションまたは複合ディメンションの最後に新しい値を追加するには、MAINTAIN コマンドとともに ADD キーワードを使用します。実際に値が追加される方法および使用する引数は、値の追加先がディメンションか複合ディメンションかによって異なります。

連結ディメンションには、値を直接追加する必要はありません。連結ディメンションのベース・ディメンションに値を追加すると、Oracle OLAP によって自動的にその値が連結ディメンションに追加されます。同様に、ディメンション・サロゲートにも値を追加する必要はあ

りません。ただし、ディメンション・サロゲートのディメンションに値を追加した場合、その新しい値に対するサロゲートをディメンション・サロゲートに追加できます。

MAINTAIN コマンドとともに MERGE キーワードを使用すると、様々なリスト上のすべてのディメンション値がディメンションに含まれていることを簡単に確認できます。この構文を使用すると、リストの新しい値は自動的に追加され、重複している値は無視されます。この方法でディメンション値を入力すると、大量の値の入力にかかる時間を大幅に短縮できます。

MAINTAIN コマンドとともに ADD キーワードを使用することによって、次の方法でディメンションに値を追加できます。

- 追加する値のみを指定します。この場合、値はディメンション値のリストの末尾に追加されます。
- 追加する値およびその位置を指定します。

例 5-1 ディメンションへの値の追加

次のコマンドを実行すると、都市のリストの先頭に ATLANTA が追加され、OMAHA の後に PEORIA が挿入されます。

```
MAINTAIN city ADD 'ATLANTA' FIRST, 'PEORIA' AFTER 'OMAHA'
```

city ディメンションのデフォルトのステータス・リストを表示すると、新しい値がリスト内の適切な位置に追加されていることを確認できます。

```
SHOW VALUES(city NOSTATUS)
ATLANTA
CONCORD
LINCOLN
NEW YORK
OMAHA
PEORIA
SEATTLE
```

新しい値のマージ時のリレーションの更新

値をディメンションにマージする場合、そのディメンションに関連するすべてのリレーションを更新することをお薦めします。

- 次に示す単純な MAINTAIN コマンドの構文を使用して、値をディメンションにマージすると同時にリレーションも更新できる場合があります。

  MAINTAIN *dimension* MERGE [*exp* [RELATE *relation*]]

  exp 引数には、ディメンションにマージする値を含む、ディメンション化された式（たとえば、ディメンション値を含む、ディメンション化されたテキスト変数）を指定します。

RELATE *relation* 句には、更新するリレーションの名前を指定します。

注意：*exp* 引数はディメンション化されている必要があります。また、これらのディメンションの 1 つ以上が、RELATE *relation* 句に指定したリレーションの定義に含まれている必要があります。

- 前述以外の場合、そのディメンションに関連するすべてのリレーションを明示的に更新する必要があります。

リレーションを明示的に更新する方法の詳細は、5-10 ページの「データ・オブジェクトへの値の代入」を参照してください。

product ディメンションの最初の 3 つの値と district ディメンションの最初の 5 つの値のすべての組合せで構成される、comp_proddist という名前の複合ディメンションを定義すると想定します。次のコマンドを使用して、15 のすべての値を効率的に含めることができます。

```
DEFINE comp_proddist COMPOSITE <product district>
LIMIT product TO FIRST 3
LIMIT district TO FIRST 5
MAINTAIN comp_proddist MERGE <product district>
```

この方法は、結合ディメンションにも有効です。

ディメンションからの値の削除

MAINTAIN コマンドとともに DELETE キーワードを使用すると、ディメンションから値を削除できます。削除する値は、LIMIT コマンドを使用して値を選択する場合と同様に選択します。次の値を選択して削除できます。

- 1 つの値、値のリスト、値の範囲またはすべての値
- 指定した関連ディメンションの値のリストに一致する値
- ディメンション内の最初、最後、または指定した位置に存在する値
- ブール条件に一致する値
- 指定した基準に従ってディメンション値がソートされた後の、そのディメンションの上位または下位 *n* 個の値（あるいは上位または下位 *n*% の実行者）
- 階層ディメンションの場合、階層内で特定の関係を持つ値
- 値セット内の値に一致するディメンション値

例 5-2 ディメンションからの値の削除

city から、人口が 75,000 人未満のすべての都市を削除すると想定します。コマンド実行前の city ディメンションのデフォルトのステータス・リストには、次に示す 6 つの値が含まれています。

```
SHOW VALUES (city NOSTATUS)
ATLANTA
CONCORD
LINCOLN
COLUMBUS
PEORIA
SEATTLE
```

各都市の人口を含む変数 population.c を使用します。

```
MAINTAIN city DELETE population.c LT 75000
```

人口が 75,000 未満であるのは LINCOLN および PEORIA のみであると想定すると、city ディメンションのデフォルトのステータス・リストには、次の値が含まれています。

```
SHOW VALUES (city NOSTATUS)
ATLANTA
CONCORD
COLUMBUS
SEATTLE
```

結合ディメンションからの値の削除

MAINTAIN コマンドとともに DELETE キーワードを使用すると、結合ディメンションから値を削除できます。

また、結合ディメンションのベース・ディメンションに直接 MAINTAIN コマンドを使用しても、結合ディメンションから値を削除できます。ベース・ディメンションから値を削除すると、そのベース・ディメンション値に関連付けられていたすべての値がその結合ディメンションから削除されます。

product および district をベース・ディメンションとする prod_dist という名前の結合ディメンションが存在すると想定します。その結合ディメンションから値 <'SNOWSHOES' 'ATLANTA'> を削除するには、次のコマンドを使用します。

```
MAINTAIN prod_dist DELETE <'SNOWSHOES' 'ATLANTA'>
```

ディメンション値の位置の変更

MAINTAIN コマンドとともに MOVE キーワードを使用すると、ディメンション・リスト内の 1 つ以上の値の位置を変更できます。時間ディメンションまたは複合ディメンション内の値の位置は変更できません。

ディメンション値をアルファベット順に格納する必要がある場合、最初に SORT コマンドを使用して値を一時的にソートし、次に MAINTAIN コマンドを使用して、ソート順序で値を格納します。

SEATTLE を都市リストの末尾に移動させるには、TEXT 変数 textvar を使用します。

```
textvar = 'SEATTLE'
MAINTAIN city MOVE textvar LAST
```

ソート順序でのディメンション値の格納

ディメンションの値をソート順序で格納するには、次の手順を実行します。

1. ディメンションを、そのすべての値に制限します。

   ```
   LIMIT dimension TO ALL
   ```

2. 希望のソート基準に従って、ディメンション値をソートします。

   ```
   SORT dimension A sort-criterion
   ```

   値をアルファベット順にソートするには、ディメンション自体を基準にしてソートします。

3. ディメンション値をソート順序で格納します。

   ```
   MAINTAIN dimension MOVE VALUES(dimension) FIRST
   ```

city ディメンションのデフォルトのステータス・リストに、次の値が含まれていると想定します。

```
SHOW VALUES (city NOSTATUS)
ATLANTA
CONCORD
LINCOLN
COLUMBUS
PEORIA
SEATTLE
```

次のコマンドを実行すると、city の値がアルファベット順にソートされ、その順序で値が格納されます。

```
SORT city A city
MAINTAIN city MOVE VALUES(city) FIRST
```

city のデフォルトのステータス・リストに、新しいソート順序が反映されます。

```
SHOW VALUES (city NOSTATUS)
ATLANTA
COLUMBUS
CONCORD
LINCOLN
PEORIA
SEATTLE
```

複合ディメンションおよび結合ディメンションのメンテナンス

複合ディメンションと結合ディメンションは、両方とも、ディメンションと値の組合せのリストです。この組合せには、複合ディメンションまたは結合ディメンションのベースとなる各ディメンションからの 1 つの値が含まれます。複合ディメンションと結合ディメンションは、メンテナンスの方法が異なります。

複合ディメンションのメンテナンス

複合ディメンションは、自動的にメンテナンスされる内部構造です。そのため、複合ディメンションの最も単純なメンテナンス方法は、ベース・ディメンションのみをメンテナンスすることです。これによって、複合ディメンション内の値が自動的にメンテナンスされます。

多くの場合、複合ディメンションのメンテナンスを特に行う必要はありません。ただし、詳細に制御する必要がある場合、複合ディメンションを明示的にメンテナンスする必要がある場合があります。その場合、MAINTAIN コマンドを使用して値を追加、削除およびマージします。

結合ディメンションのメンテナンス

結合ディメンションは、複合ディメンションとは異なり、MAINTAIN コマンドを使用して明示的にメンテナンスする必要がある実際のディメンションです。

連結ディメンションのメンテナンス

MAINTAIN コマンドを使用して、連結ディメンションの値の順序を変更できます。連結ディメンションのコンポーネントである単純なディメンションに MAINTAIN MOVE コマンドを実行すると、その連結ディメンションの値の位置は影響を受けません。

連結ディメンション値の追加、削除または名前の変更、あるいは他のディメンションの値から連結ディメンションの値へのマージは、MAINTAIN コマンドを使用して行うことができません。

MAINTAIN コマンドを使用して、連結ディメンションのコンポーネントである単純なディメンションへ値を追加すると、Oracle OLAP によって、その値はコンポーネントであるディメンションの値として連結ディメンションに追加されます。単純なディメンションの値を、コンポーネントである単純なディメンションとマージすると、Oracle OLAP によって、それら

の値はコンポーネントであるディメンションの値として連結ディメンションに追加されます。

連結ディメンションのコンポーネントである単純なディメンションの値の削除または名前の変更を行うと、Oracle OLAP によって、連結ディメンション内のその値の削除または名前の変更が行われます。MAINTAIN コマンドを使用して、連結ディメンションのコンポーネントである単純なディメンションの値を追加、マージまたは削除すると、連結ディメンションのステータスが自動的に ALL に設定されます。

データ・オブジェクトへの値の代入

式では、一時データが作成されます。この結果の値は表示できますが、アナリティック・ワークスペースに自動的に保存されることはありません。式の結果を保存するには、式と同じデータ型とディメンションを持つオブジェクトにそれを格納します。式の結果の値をオブジェクトに格納するには、代入文を使用します。

代入文は、OLAP DML の = 演算子の前後（左側と右側）に式を指定して作成します。

target-expression = *source-expression*

代入文を使用すると、ターゲット式の値をソース式の結果と同じ値に設定できます。

> **参照：** データ・オブジェクトへデータを格納する方法の詳細は、第 3 章「データ・オブジェクトの定義」を参照してください。

代入文でのオブジェクトの使用

次の表に、代入文で使用可能なオブジェクトを示し、ターゲット式およびソース式で使用できるかどうかを示します。

オブジェクト	ターゲット式	ソース式
変数	使用可能	使用可能
リレーション	使用可能	使用可能
ディメンション	モデルでのみ使用可能	使用可能
サロゲート	使用不可	使用可能
複合ディメンション	使用不可	使用可能
ワークシート	使用可能	使用可能
ファンクション	使用不可	使用可能
フォーミュラ	使用不可	使用可能
値セット	使用不可	使用可能

= 演算子を使用して単一セル式の値を単一セルに代入すると、1 つの値が格納されます。ただし、= 演算子を使用して 1 つ以上のディメンションを持つターゲット変数に単一セル式の値を代入すると、ターゲット変数の各ディメンションのステータス内の値に対して代入がループ実行され、その変数の対応するセルにデータ値が代入されます。

例 5-3 変数への値の代入

choicedesc 変数は、choice によってディメンション化されています。データ入力前のこの変数のセルには、NA 値のみが含まれています。

```
CHOICE           CHOICEDESC
-------------- --------------------
REPORT         NA
GRAPH          NA
ANALYZE        NA
DATA           NA
QUIT           NA
```

次のコマンドを使用して、choicedesc 変数を初期化すると想定します。

```
choicedesc =  JOINCHARS ('Description for ' choice)
```

この変数のすべての choicedesc セルに、適切な値が含まれています。

```
CHOICE             CHOICEDESC
-------------- -------------------------
REPORT         Description for REPORT
GRAPH          Description for GRAPH
ANALYZE        Description for ANALYZE
DATA           Description for DATA
QUIT           Description for QUIT
```

次の例では、time、product および district によってディメンション化され、新しい変数に代入される式を示します。この式は、2001 年の単位売上に基づいて 2002 年の売上計画を計算します。

```
DEFINE units.plan INTEGER <month product district>
LIMIT month TO 'DEC02'
units.plan = LAG(units 12 month) * 1.15
```

複合ディメンションを含む変数に値を代入する方法

複合ディメンションを含む変数にデータを代入する場合、ターゲット変数に現在セルが存在しない疎密なディメンションの組合せを含め、ターゲット変数のステータス内のディメンション値の各組合せに対してソース式が評価されます。現在ターゲットにセルが存在しないこれらの組合せに対するソース式が NA ではない場合、新しいセルが作成され、そのセルにデータが代入されます。

= コマンドを使用して複合ディメンションを含むターゲット変数に値を代入する場合、このコマンドによって、次の動作が自動的に実行されます。

- NA 以外の値が代入されている、ターゲット変数の欠落したセルが作成されます。
- これらの新しいセルに対応するディメンションと値のすべての組合せが複合ディメンションに追加されます。

このため、代入後は、ターゲット変数と複合ディメンションの両方が大きくなる場合があります。ターゲット変数にすでに存在するセルのみに値を代入する場合、= コマンドに ACROSS キーワードを追加します。

OLAP DML を使用すると、複合ディメンションを含む変数へのデータの代入に、異なる評価動作を指定できます。代入文のデフォルトの評価動作を変更して、現在ターゲット変数にセルが存在する、ステータス内のディメンション値の組合せに対してのみソース式が評価されるようにできます。

これを行うには、次の構文を使用します。これは、疎密なディメンションの複合ディメンションによって、どの疎密なディメンションの組合せにデータ・セルが含まれているかを追跡できるためです。

varname = *expression* ACROSS *composite*

varname 引数は変数の名前です。これは、データの代入先のターゲットです。

expression 引数は、ターゲット変数に代入するデータを含むソース式です。

ACROSS は、デフォルトの評価動作を変更し、ターゲット変数の複合ディメンションを評価するように指定するキーワードです。

composite 引数は、ターゲット変数の疎密なディメンション用の複合ディメンションです。変数が名前付き複合ディメンションで定義されている場合は、複合ディメンションの名前を指定します。変数が名前なし複合ディメンションで定義されている場合は、SPARSE キーワードを使用して名前なし複合ディメンションを参照します（SPARSE <MARKET PRODUCT> など）。

例 5-4 複合ディメンションを含む変数への値の代入

sales のデータを、関連付けられたディメンション値がステータス内に存在する sparse_sales の既存のデータ・セルにのみ代入するには、次のコマンドを使用します。

```
sparse_sales = sales ACROSS SPARSE<product market>
```

ソース式が 1 つの値の場合、ACROSS キーワードが特に有効です。sparse_sales のディメンションに制限がない場合、ソース式が NA であることはないため、次のような代入コマンドを実行すると、ディメンション値のすべての組合せに対してセルが作成されます。

```
sparse_sales = 0
```

これは疎密な変数の用途に反しています。

これとは異なり、次のコマンドでは sparse_sales の既存のセルにのみ 0（ゼロ）が設定されます。

```
sparse_sales = 0 ACROSS SPARSE<product market>
```

リレーションへの値の代入

代入文を使用して、リレーションに値を代入できます。代入文を実行すると、ターゲット・リレーションのステータス内の値に対して代入がループ実行され、ターゲット・リレーションの対応するセルにデータ値が代入されます。

次のいずれかの値を代入することによって、テキスト・ディメンションを含むリレーションに値を代入することができます。

- ディメンションのテキスト値
- ディメンションのデフォルトのステータス・リストに含まれるディメンション値の位置を示す整数

ディメンションへの値の代入

多くの場合、ディメンションへの値の代入には代入文を使用できません。ただし、モデル方程式では、計算の結果が数値である場合、= 演算子を使用してその結果をディメンション値に代入できます。 ただし、モデル方程式（式）は、他のコンテキストで使用される式とはいくつかの点で異なります。

> **参照：** モデルの使用方法の詳細は、第 8 章「モデルの使用」を参照してください。

データ・オブジェクトの特定のセルへの値の代入

代入文のターゲットとともに QDR を使用できます。これによって、変数またはリレーションの特定のセルに値を代入することができます。

次の例では、修飾データ参照に指定されている sales 変数のデータ・セルに、値 10200 が割り当てられます。sales という名前の変数の BOSTON、TENTS および JAN99 に関連付けられたセルに値が存在しない場合、このセルに値が代入され、変数に追加されます。セルに値が存在する場合、値 10200 によってその値が上書きされます。

```
sales(market 'BOSTON' product 'TENTS' month 'JAN99')= 10200
```

> **参照：** QDR の詳細は、4-5 ページの「式のディメンションに対する 1 つの値の指定」を参照してください。

データの計算および分析

OLAP DML では、通常、次の方法でデータを計算および分析します。

- 一般的な計算は、組込みファンクションを使用して実行します（組込みファンクションの詳細は、Oracle9*i* OLAP DML Reference ヘルプを参照）。
- 1 つ以上の階層ディメンションによってディメンション化された変数のデータを集計（ロールアップ）します（第 12 章「データの集計」を参照）。
- ベース・オブジェクトのデータに基づいて、ソース・オブジェクトから変数にデータを割り当てます（第 9 章「データのアロケーション」を参照）。
- MODEL オブジェクトを使用して移入ソリューション変数を作成します（第 8 章「モデルの使用」を参照）。
- 傾向の分析に基づいてデータを予測します（Oracle9*i* OLAP DML Reference ヘルプの FCSET、FCOPEN、FCEXEC、FCCLOSE および FCQUERY コマンドの項を参照）。

OLAP DML には、数値分析用の組込みファンクションが含まれています。次の表に、これらのファンクションのカテゴリを示します。

カテゴリ	説明
セルごとの数値	式または変数の各セルを操作します。
時系列	以前または将来の期間から値を取得し、それらの値を操作します。
統計	統計分析用の計算を実行します。
財務	財務分析用の計算を実行します。
集計	通常、入力式の複数の値に対する 1 つの値で構成される、集計値を戻します。

参照：

- ファンクションのカテゴリのリストは、Oracle9*i* OLAP DML Reference ヘルプを参照してください。
- 数値式の使用方法の詳細は、4-14 ページの「数式」を参照してください。

6

データの選択

この章では、ディメンションのステータスの概要を示し、LIMIT コマンドを使用してアナリティック・ワークスペース内のデータ・ビューを一時的に変更する方法について説明します。LIMIT コマンドは、SQL SELECT 文の WHERE 句に相当します。

この章では、次の項目について説明します。

ディメンションのステータスの概要

ディメンションの**現行のステータス・リスト**は、そのディメンションの現在アクセス可能な値を順序付けしたリストです。ディメンションの現行のステータス・リストに存在する値を、「ステータス内に存在する」といいます。ディメンションの現行のステータス・リストによって、そのディメンションによってディメンション化されるすべてのオブジェクトから選択されるデータが決定します。

ディメンションの場合、現行のステータス・リスト内のディメンション値のみがアクセスされます。ディメンション化されたオブジェクトの場合、現行のステータス・リスト内のディメンション値によって索引付けされたデータ値のみがアクセスされます。

ディメンション化されたオブジェクトに対してループが実行される場合、現行のステータス・リスト内のディメンション値の順序で、オブジェクトの値がアクセスされます。

ディメンション値がステータス・リスト内に存在するかどうかは、任意のセッション中の値の表示方法のみに影響を及ぼします。アナリティック・ワークスペースに格納されている値に永続的に影響を及ぼすことはありません。

アナリティック・ワークスペースを初めてアタッチすると、各ディメンションの現行のステータス・リストには、読込み権限を持つディメンションのすべての値が、格納された順序で含まれます。この値のリストを、そのディメンションの**デフォルトのステータス・リスト**といいます。

ディメンション・サロゲートのステータス・リストは、そのディメンションのステータス・リストと同じです。サロゲートには、そのディメンションのものと異なる現行またはデフォルトのステータス・リストは存在しません。

現行のステータス・リストの変更

ディメンションの現行のステータス・リストは、次のコマンドを使用して変更できます。

- LIMIT コマンド: ディメンションの現行のステータス・リスト内の値とその順序を変更できます。
- SORT コマンド: ディメンションの現行のステータス・リスト内の値の順序を調整できます。

デフォルトのステータス・リストの変更

ディメンションのデフォルトのステータス・リストは、次のコマンドを使用して変更できます。

- MAINTAIN コマンド: ディメンションの値の追加、削除、移動、マージおよび名前の変更を行うことができます。ただし、連結ディメンションの場合は、ディメンション内で値の順序を変更する場合にのみ MAINTAIN コマンドを使用できます。
- PERMIT コマンドまたは PERMITRESET コマンド: ディメンションに関連付けられた値の読込み権限を変更できます。

参照：

- ディメンション値を格納およびメンテナンスする方法の詳細は、5-3 ページの「ディメンションおよび複合ディメンションのメンテナンス」を参照してください。
- ワークスペース・オブジェクトへ権限を設定する方法の詳細は、2-12 ページの「アナリティック・ワークスペースへのセキュリティの追加」を参照してください。

ステータス・リストの識別および取得

ディメンション値のステータスを識別および取得するには、次のコマンドおよびファンクションを使用します。

コマンドまたはファンクション	説明
INSTAT ファンクション	ディメンション値がディメンションの現行のステータス・リスト内に存在するかどうかを確認します。
STATFIRST ファンクション	ディメンションの現行のステータス・リストに含まれる最初の値を取得します。
STATLAST ファンクション	ディメンションの現行のステータス・リストに含まれる最後の値を取得します。
STATUS コマンド	ディメンションの 1 つ以上の値のステータス、またはアナリティック・ワークスペース内のすべてのディメンションのステータスを現行の送信ファイルに送信します。
VALUES ファンクション	指定したキーワードに対応する様々な値を取得します。 ■ NOSTATUS キーワードを指定すると、ディメンション・リストのデフォルトのステータス・リストが取得されます。 ■ STATUS キーワードを指定すると、ディメンションの現行のステータス・リストが取得されます。 ■ INTEGER キーワードを指定すると、行ごとに 1 つのディメンション値を含む複数行のテキスト値が戻されます。このキーワードを指定しないと、ディメンション値の位置番号が整数として戻されます。

ディメンション・ステータスの保存およびリストア

ディメンションの現行のステータスは、次の方法で保存できます。

- 現行のステータス（またはディメンション値）を保存して任意のセッションで使用する場合、名前付き値セットを使用します。値セットを定義するには、DEFINE VALUESET コマンドを使用します。
- 現行のステータス（またはディメンション値、値セット、オプション、単一セル変数）を保存して現行のプログラムで使用する場合、PUSHLEVEL および PUSH コマンドを使用します。現行のステータス値をリストアするには、POPLEVEL および POP コマンドを使用します。
- 現行のステータス（またはディメンション値、値セット、オプション、単一セル変数、単一セル・リレーション）を保存、アクセスまたは更新して現行のセッションで使用する場合、名前付きコンテキストを使用します。コンテキストを定義するには、CONTEXT コマンドを使用します。

コンテキストは、アナリティック・ワークスペースで使用するオブジェクト値を保存するための、最も高度な方法です。コンテキストを使用すると、保存したオブジェクト値に対してアクセス、更新およびコミットを行うことができます。一方、PUSH および POP を使用すると、値を保存およびリストアできます。通常、PUSH および POP コマンドは、プログラム実行中にのみ適用される変更を行うために、プログラム内のみで使用します。

> **参照：** 環境設定を保存する方法の詳細は、7-17 ページの「セッション環境の保持」を参照してください。

単純な値のリストへの制限

データを選択する一般的な方法は、ディメンションを 1 つの値または値のリストに制限することです。ディメンション値を制限する場合、ディメンションのかわりにディメンション・サロゲートを使用できます。この方法で LIMIT コマンドを使用する単純な構文を次に示します。

```
LIMIT dimension TO values
```

values 引数には、次のどの組合せでも指定できます。

- カンマで区切られたリテラル値、または各行にディメンションの値を含む複数行テキストとして表現されたディメンション値
- *value1* TO *value2* として表現されたディメンション値の範囲
- カンマで区切られた整数として表現された、ディメンション値の論理位置を表す整数値
- *value1* TO *value2* として表現された、ディメンション値の論理位置を表す整数値の範囲
- 値セット

1995 年の 1 月から 3 月までの Boston における Footwear（履物）の売上のレポートが必要であると想定します。次のコマンドを実行すると、適切なディメンションが制限され、レポートが要求されます。

```
LIMIT month TO 'JAN95' 'FEB95' 'MAR95'
LIMIT product TO 'FOOTWEAR'
LIMIT district TO 'BOSTON'
REPORT sales
```

レポートの出力は、次のように表示されます。

```
DISTRICT: BOSTON
                ------------SALES--------------
                ------------MONTH--------------
PRODUCT           JAN95      FEB95      MAR95
--------------- ---------- ---------- ----------
FOOTWEAR         91,406.82  86,827.32 100,199.46
```

代替ディメンションを使用してディメンション値を制限する例として、識別番号が値として格納されている storeid という名前の NUMBER ディメンションが存在すると想定します。storeid の値は、10、20、30、100、110、120 および 200 です。storeid に対して、storenum という名前の INTEGER ディメンション・サロゲートが存在します。これには、storeid の値の各位置に対する整数が含まれています。storenum の値は、1 ～ 7 の整数です。storeid と storenum の両方の現行のステータス・リストを同じ値セットに制限するには、次のどのコマンドでも使用できます。

```
LIMIT storeid TO 10, 100
LIMIT storenum TO 1, 4
LIMIT storenum TO storeid 10, 100
LIMIT storenum TO storenum 1, 4
```

ブール式の使用の制限

LIMIT コマンドを使用して、ブール式の結果に従ってディメンションを制限できます。この方法で LIMIT コマンドを使用する単純な構文を次に示します。

```
LIMIT dimension TO Boolean-expression
```

この形式で LIMIT コマンドを使用する場合、現在ステータス内に存在する値は、ブール式で true となったディメンション値に置換されます。

ブール式を作成する場合は、次のことに注意してください。

- ブール式は、ステータスを設定済のディメンションによってディメンション化されている必要があります。
- ブール式で比較する式のデータ型は、類似している必要があります。

たとえば、次のブール式のブール演算子 GT の両側のデータ型は類似しています。

```
LIMIT market TO units.m GT 50000
```

次の例では、TOTAL ファンクションの値が product によって分割され、リテラル（数値 12000000）と比較されます。LIMIT コマンドによって、product ディメンションのステータス内に現在存在する値が、すべての月と地区の合計売上が 1200 万より大きい product ディメンションの値に置換されます。

```
LIMIT product TO TOTAL(sales product) GT 12000000
```

LIMIT による多次元ブール式の処理

複数のディメンションを持つブール式を LIMIT コマンドが処理する動作を理解することは、コマンドを正しく使用するために重要です。

単純なブール式の結果は、1 つの値です。ブール式に LIMIT コマンドを使用した場合、式の値の配列を作成および戻すために、ディメンションに対してループが実行されません。かわりに、ディメンションのステータス・リスト内の最初の値が、式内の各ディメンションに対して識別され、それらの値を使用している式が評価され、1 つの値が戻されます。

ブール式の結果に次元性を持たせる必要がある場合、EVERY、ANY または NONE ファンクションを使用します。これによって、ブール式の結果のディメンションを指定できます。

month、district および product に、次に示すディメンションのステータスが存在すると想定します。

```
The current status of MONTH is:
JAN95 TO MAR95
The current status of DISTRICT is:
BOSTON
The current status of PRODUCT is:
ALL
```

1 か月の売上が 1 回でも 90,000 ドルを超えた製品は、product ディメンションのステータス内に存在する必要があると想定します。次のコマンドを発行すると、ディメンションの現行のステータス内の値のうちこの条件を満たしているものを確認できます。

```
REPORT sales GT 90000
```

次に示すとおり、レポートの FOOTWEAR と CANOES の行の両方に YES が表示されます。これらの製品は、1995 年の 1 月から 3 月の間で 1 回以上 90,000 ドルを超える売上を記録しています。

```
DISTRICT: BOSTON
                ---------SALES GT 90000---------
                -------------MONTH--------------
PRODUCT           JAN95      FEB95      MAR95
--------------- ---------- ---------- ----------
TENTS                  NO         NO         NO
CANOES                 NO         NO        YES
RACQUETS               NO         NO         NO
SPORTSWEAR             NO         NO         NO
FOOTWEAR              YES         NO        YES
```

次に示す単純なブール式のみを使用して product ディメンションを制限しても、必要な結果が得られるように考えられます。

```
LIMIT product TO sales GT 90000
```

ただし、ブール式が評価される際、product ディメンションの値の配列を作成および戻すために、sales 変数に対してループが実行されません。かわりに、ディメンションのステータス・リスト内の最初の値が、sales に含まれる、product 以外の各ディメンションに使用されます。この場合、sales 変数の month ディメンションの値には JAN95 が使用され、DISTRICT ディメンションの値には BOSTON が使用されます。

JAN95 および BOSTON は、FOOTWEAR 製品に対してのみブール式で TRUE に評価されます。このため、FOOTWEAR のみが product ディメンションのステータス内に存在することになります。

次に示すとおり、Boston の売上レポートには、1995 年の 1 月から 3 月の間で 1 回以上 90,000 ドルを超える売上を記録した FOOTWEAR 製品の値のみが表示されます。

```
REPORT sales

The current status of PRODUCT is:
FOOTWEAR
DISTRICT: BOSTON
                -------------SALES--------------
                -------------MONTH--------------
PRODUCT           JAN95      FEB95      MAR95
--------------- ---------- ---------- ----------
FOOTWEAR        91,406.82  86,827.32 100,199.46
```

式に一致する値の制限

ブール式に一致するすべてのディメンション値にディメンションを制限するには、ブール式で ANY ファンクションを使用します。

例 6-1 ANY ファンクションを使用した制限

次の LIMIT コマンドでは、ANY ファンクションを使用して、product ディメンションを、sales 変数の値が 90,000 ドルより大きいすべてのディメンション値（CANOES および FOOTWEAR）に制限する方法を示します。

- ANY ファンクションの 1 つ目の引数（sales GT 90000）には、評価するブール式を指定します。
- ANY ファンクションの 2 つ目の引数（product）には、ブール式の結果の次元性を指定します。

この例では、ブール・ファンクションが評価される際に、product ディメンションの TRUE 値にテストが実行され、値の配列が戻されます。

```
LIMIT product TO ANY(sales GT 90000, product)
```

product ディメンションのステータス内には、CANOES と FOOTWEAR の両方が存在します。これらの製品は、1995 年の 1 月から 3 月の間で 1 回以上 90,000 ドルを超える売上を記録しています。

次に示すとおり、Boston の売上レポートには、CANOES と FOOTWEAR 製品の両方が表示されます。

```
REPORT sales

The current status of PRODUCT is:
CANOES, FOOTWEAR
DISTRICT: BOSTON
                -------------SALES--------------
                -------------MONTH--------------
PRODUCT           JAN95       FEB95       MAR95
--------------- ---------- ---------- ----------
CANOES           66,013.92  76,083.84  91,748.16
FOOTWEAR         91,406.82  86,827.32 100,199.46
```

上位または下位の値への制限

現在ステータス内に存在するディメンション値を、式として表した基準に従って、上位または下位の実行者に設定できます。この方法で LIMIT コマンドを使用する単純な構文を次に示します。

```
LIMIT dimension TO [BOTTOM|TOP] n BASEDON expression
```

また、現在ステータス内に存在するディメンション値を、式として表した基準に従って、上位または下位の任意の割合の実行者に設定することもできます。この方法で LIMIT コマンドを使用する単純な構文を次に示します。

```
LIMIT dimension TO [BOTTOM|TOP] percent PERCENTOF expression
```

この構文では、式内の値の占有率の割合によって値がソートされ、識別された値がステータスに挿入されます。

PERCENTOF 基準に基づいた制限では、同じ基準値が関連付けられている多くのディメンション値の 1 つが、ステータス内の最下位に存在する場合があります。この場合、基準値の合計が指定した割合を大幅に超過する場合でも、LIMIT によって、その基準値を持つすべてのディメンション値が結果のステータスに含まれます。

注意： それ自体の値を変更する条件式を使用しないでください。

例 6-2 基準に基づいた上位または下位の値への制限

ステータス・リストを sales の値に従って降順にソートし、上位 2 つの実行者のみをステータス内に残すと想定します。この場合、TOP および BASEDON キーワードを使用して、変数の値を基準としてディメンションのステータスを制限します。

```
LIMIT product TO 'SPORTSWEAR'
LIMIT month TO 'JUL96'
LIMIT district TO TOP 2 BASEDON sales
```

次の REPORT コマンドを発行すると想定します。

```
REPORT DOWN district sales
```

LIMIT コマンドの結果を示す、次のレポートが生成されます。

```
PRODUCT: SPORTSWEAR
                --SALES---
                --MONTH---
DISTRICT          JUL96
--------------- ----------
DALLAS          220,416.81
ATLANTA         211,666.14
```

例 6-3 割合に基づいた上位または下位の値への制限

TOTAL(sales) に対する各製品の占有率によって製品を降順にソートし、上位から順に、product ごとの sales の累積合計が全売上の 30% 以上になるまで値をステータス・リストに追加すると想定します。この方法でディメンションを制限するには、次のコマンドを使用します。

```
LIMIT product TO TOP 30 PERCENTOF TOTAL(sales, product)
```

次のコマンドを実行すると、Boston 地区で全売上の 30% 以上を占める製品の 2002 年の 1 月から 3 月までのレポートが生成されます。

```
LIMIT month TO 'JAN02' 'FEB02' 'MAR02'
LIMIT district TO 'BOSTON'
LIMIT product TO TOP 30 PERCENTOF TOTAL(sales, product)
REPORT sales
```

レポートの出力は次のとおりです。

```
DISTRICT: BOSTON
                -------------SALES--------------
                -------------MONTH--------------
PRODUCT           JAN02      FEB02      MAR02
--------------- ---------- ---------- ----------
FOOTWEAR         91,406.82  86,827.32 100,199.46
CANOES           66,013.92  76,083.84  91,748.16
```

関連するディメンションの値への制限

LIMIT コマンドを使用すると、1 つ以上の関連するディメンションの値にディメンションを制限できます。この方法で LIMIT コマンドを使用する単純な構文を次に示します。

```
LIMIT dimension TO reldim [reldim-val]
```

reldim 引数には、制限するディメンションに関連付けられているリレーションまたはディメンションの名前を指定します。リレーションの名前を使用すると、複数のリレーションが存在する場合に使用するリレーションを指定できます。

reldim-val 引数には、制限するディメンションではなく、関連するディメンションの値のリストを指定します。LIMIT コマンドにこの引数を指定すると、関連する値に関連付けられた、制限するディメンションの値を選択することによってステータスが取得されます。*reldim-val* を指定しない場合、*reldim* の現行のステータスが使用されます。

例 6-4 関連するディメンションを使用した制限

次のコマンドを実行すると、district が、EAST 地域内の BOSTON および ATLANTA に制限されます。

```
LIMIT district TO region 'EAST'
```

次のコマンドを実行すると、product が、DIVISION 値のリストの最後に表示される部門である SPORTSWEAR および FOOTWEAR に制限されます。

```
LIMIT product TO division LAST 1
```

関連するディメンションへの制限によるステータスの決定

ディメンションを関連するディメンションに制限すると、現行のステータス・リストが次の 2 つの手順で作成されます。

1. ディメンションの現行のステータス・リスト内の値が、関連するディメンションの値の順序で配置されます。
2. 関連するディメンションの値のいずれかに対して複数のディメンション値が存在する場合、ディメンションの現行のステータス・リスト内の値が、そのデフォルトのステータス・リストの順序で配置されます。

関連するディメンションへの制限時におけるソートの抑制

ディメンションを関連するディメンションに制限する際にソートを実行するかどうかを指定するには、LIMIT.SORTREL オプションを使用します。ディメンションを関連するディメンションに制限する際にソートを実行しないようにするには、LIMIT.SORTREL を NO に設定します。これによって、大きいディメンションを制限する場合のパフォーマンスが大幅に向上します。

注意： LIMIT.SORTREL が NO である場合、ディメンションの出力は論理的な順序で表示されない場合があります。

ディメンションの値の位置に基づいた制限

LIMIT コマンドを使用すると、次のいずれかのディメンションに含まれる値の位置に基づいてディメンションのステータスを設定できます。

- 制限するディメンション
- 関連のないディメンション

ディメンションの値の位置を使用した制限

ディメンション内の値の位置に基づいてディメンションのステータスを設定するには、LIMIT コマンドとともに FIRST、LAST、NTH および POSLIST キーワードを使用します。

この方法で LIMIT コマンドを使用する単純な構文を次に示します。

```
LIMIT dimension TO {FIRST n|LAST n|NTH n|POSLIST poslist-exp}
```

FIRST、LAST および NTH キーワードは、完全なディメンション値のセット内での値の位置を示します。キーワードの後の *n* 引数には、値の数を指定します。

POSLIST キーワードは、その後に指定された *poslist-exp* 引数が、制限するディメンションの位置（数値）に評価される数値が各行に含まれるテキスト式であることを示します。

関連のないディメンションの値の位置を使用した制限

関連のないディメンションのステータス・リストの値の位置に基づいて値をディメンションのステータス・リストに挿入するには、LIMIT コマンドに NOCONVERT キーワードを指定します。これは、2 つのディメンションが異なるアナリティック・ワークスペース内に存在する場合（たとえば、2 つのアナリティック・ワークスペースの製品ディメンションが 1 対 1 で対応する場合）に特に有効です。

この方法で LIMIT コマンドを使用する単純な構文を次に示します。

```
LIMIT dimension TO NOCONVERT unrelated-dimension
```

unrelated-dimension 引数には、制限するディメンションに関連しないディメンションの名前を指定します。

階層内の関係に基づいた制限

ファミリ・ツリーを使用して、ディメンションの値をステータスに挿入できます。ディメンションを制限するには、次の操作を実行します。

- ディメンションは、指定した値のリスト内の各値、またはステータス内の各値の親、子、祖先または子孫に制限できます。
- また、特定の親関係に基づいて子孫を検索することもできます。これは、ディテール・レベルと、より低いレベルの集計であるレベルの両方が含まれる階層ディメンションに有効です。この方法で LIMIT コマンドを使用するには、アナリティック・ワークスペースに、ディメンションの各値の親を含む関係が存在している必要があります。

階層内の関係に基づいてディメンションを制限する単純な構文を次に示します。

```
LIMIT dimension TO {PARENTS|CHILDREN|ANCESTORS|DESCENDANTS|HIERARCHY} -
 USING parent-rel[valuelist]
```

PARENTS キーワードを指定すると、*valuelist* 内で各値の親が検索されます。*valuelist* が存在しない場合、ステータス内で各値の親が検索されます。親の検索には *parent-rel* が使用されます。

CHILDREN キーワードを指定すると、*valuelist* 内で各値の子が検索されます。*valuelist* が存在しない場合、ステータス内で各値の子が検索されます。子の検索には *parent-rel* が使用されます。

ANCESTORS キーワードを指定すると、*valuelist* 内で各値の祖先（親、祖父母など）が検索されます。*valuelist* が存在しない場合、ステータス内で各値の祖先が検索されます。

DESCENDANTS キーワードを指定すると、*valuelist* 内で各値の子孫（子、孫など）が検索されます。*valuelist* が存在しない場合、ステータス内で各値の子孫が検索されます。

HIERARCHY キーワードは DESCENDANTS キーワードに類似しており、これを指定すると、*parent-rel* 引数の値に基づいて子孫（子、孫など）が検索されます。

parent-rel 引数には、ディメンションとそれ自体のリレーションの名前を指定します。このリレーションには、各ディメンションに対して、その親ディメンションの値（任意の階層内ですぐ上の値）である別のディメンション値が含まれています。親リレーションには、複数のディメンションが含まれる場合があります。

valuelist 引数には、任意の値のリストを指定できます。

参照：

- 階層ディメンションの詳細は、3-21 ページの「階層ディメンションおよび階層ディメンションを使用する変数の定義」を参照してください。
- 連結ディメンションおよび階層の詳細は、3-24 ページの「連結ディメンションおよび連結ディメンションを使用する変数の定義」を参照してください。
- HIERARCHY キーワードの使用方法の詳細は、6-13 ページの「HIERARCHY キーワードと DESCENDANTS キーワードの相違点」を参照してください。

HIERARCHY キーワードと DESCENDANTS キーワードの相違点

LIMIT コマンドの HIERARCHY と DESCENDANTS キーワードでは、両方ともファミリ・ツリーに基づいてディメンションのステータスが設定できますが、戻される結果が異なります。

相違点の 1 つは、値の順序です。

- DESCENDANTS では、値がレベルごと（すべての子、次にすべての孫）にグループ化されます。
- HIERARCHY では、各親の次にその子のグループが配置されます。

また、HIERARCHY キーワードには、次の表に示す追加の引数を使用できます。これらの引数を使用して、現行のステータス・リストの内容に対する追加の操作を行うことができます。

目的	使用するキーワードまたは句
親の前に子をリストする。	INVERTED キーワード
valuelist 内の各値の *n* 世代をスキップする。*valuelist* が存在しない場合、ステータス内の各値の *n* 世代をスキップする。	SKIP *n* 句
valuelist 内の各値の *n* 世代下まで含める。*valuelist* が存在しない場合、ステータス内の各値の *n* 世代下まで含める。	DEPTH *n* 句
子のグループを構成するたびに、テキスト式として表されたコマンドを実行する。	RUN *textexp* 句
現行のステータス・リストから元の値を除外する。	NOORIGIN キーワード

例 6-5 世代のスキップ

アプリケーションで次のコマンドを実行すると想定します。

```
LIMIT market TO HIERARCHY DEPTH 2 SKIP 1 USING market.market 'TOTUS'
```

このコマンドの処理では、親リレーションが検索されて（market.market）、TOTUS の子および孫が検出され（DEPTH 2）、最初の世代が廃棄されます（SKIP 1）。

結果のステータスは次のとおりです。

```
TOTUS
BOSTON
ATLANTA
CHICAGO
DALLAS
DENVER
SEATTLE
```

TOTUS がステータスに含まれていることに注意してください。HIERARCHY を使用すると、元の値がステータスに含められます。

例 6-6 子のグループのソート

LIMIT コマンドで HIERARCHY キーワードを使用すると、RUN キーワードを使用して、子のグループが構成されるたびに、テキスト式として指定されたコマンドが実行されるように設定できます。これによって、ステータス内の値に対する追加の操作を行うことができます。

次のコマンドを実行すると、market.market セルフ・リレーションを使用して market ディメンションの値が子孫に制限されるのみでなく、子のグループが構成されるたびに、単位売上に基づいて昇順に market ディメンションの値がソートされます。

```
LIMIT market TO HIERARCHY RUN 'SORT market A unit.m' USING market.market
```

> **注意：** このコマンドでは、LIMIT コマンドの TO のかわりに KEEP または REMOVE を使用すると、SORT コマンドが無効になります。

例 6-7 リレーションを使用した階層のドリルダウン

market ディメンションの地域レベルから地区レベルにドリルダウンすると想定します。このプロセスは、2 つの手順で構成されます。

このプロセスの最初の手順では、地区、地域および米国全域レベルでの合計が埋め込まれた、market ディメンションを地域レベルのデータに制限します。これは、market と marketlevel 間のリレーションである mlv.market を使用して行います。

次のコマンドを実行すると、次に示すレポートが作成されます。このレポートには、mlv.market の値が表示されています。

```
REPORT mlv.market

MARKET          MLV.MARKET
--------------- ----------
TOTUS           TOTUS
EAST            REGION
BOSTON          DISTRICT
ATLANTA         DISTRICT
CENTRAL         REGION
CHICAGO         DISTRICT
DALLAS          DISTRICT
WEST            REGION
DENVER          DISTRICT
SEATTLE         DISTRICT
```

次のコマンドを実行すると、market の値が希望の値に制限され、market ディメンションのステータス内に現在存在する値が表示されます。

```
LIMIT market TO mlv.market 'REGION'
STATUS market

The current status of MARKET is:
EAST, CENTRAL, WEST
```

このプロセスの次の手順では、地域レベルから地区レベルのデータにドリルダウンします。セルフ・リレーション market.market を使用してドリルダウンを行うことができます。このリレーションには、market ディメンションの各値の親の名前が含まれています。

```
DEFINE MARKET.MARKET RELATION MARKET <MARKET>
LD Self-relation for the Market Dimension
```

market.market のレポートの出力は、次のとおりです。

```
MARKET          MARKET.MARKET
--------------- -------------
TOTUS           NA
EAST            TOTUS
BOSTON          CENTRAL
ATLANTA         EAST
CENTRAL         TOTUS
CHICAGO         CENTRAL
DALLAS          CENTRAL
WEST            TOTUS
DENVER          WEST
SEATTLE         WEST
```

次のコマンドを実行すると、market が EAST、CENTRAL および WEST 地域に制限され、LIMIT コマンドに CHILDREN キーワードを使用して地区レベルのデータにドリルダウンします。

```
LIMIT market TO mlv.market 'REGION'
LIMIT market tO CHILDREN USING market.market
```

market のレポートの出力は、次のとおりです。この出力には、現在ステータス内に存在する値が表示されています。

```
MARKET
-------------
BOSTON
ATLANTA
CHICAGO
DALLAS
DENVER
SEATTLE
```

複合ディメンションおよび結合ディメンションの制限

複合ディメンションの値は明示的に制限できません。複合ディメンションはディメンションではないため、固有のステータスは存在しません。ステータス内の複合ディメンションの値は、複合ディメンションのベース・ディメンションのステータス内の値によって決まります。通常、OLAP DML のファンクションおよびコマンドを使用して複合ディメンションで定義されたオブジェクトを処理する際、デフォルトの動作では、オブジェクトの定義時に SPARSE キーワードまたは名前付き複合ディメンションを使用していない場合と同様にこれらのオブジェクトが処理されます。

LIMIT コマンドを使用して、複合ディメンションで定義されていない変数と同様に、複合ディメンションで定義された変数のディメンションにステータスを設定できます。

参照： 複合ディメンションの詳細は、3-17 ページの「疎密なデータを効率的に処理する変数の定義」を参照してください。

例 6-8 複合ディメンションで使用するディメンションの制限

次の定義に示すとおり、month および（prod_market 複合ディメンションを使用した）product と market によってディメンション化される coupons という名前の変数がアナリティック・ワークスペースに存在すると想定します。

```
DEFINE coupons VARIABLE INTEGER <month prod_market <product market>>
```

次のコマンドを実行すると、coupons 変数のすべてのベース・ディメンションのデフォルトのステータスが表示されます。

```
STATUS coupons

The current status of MONTH is:
ALL
The current status of PRODUCT is:
ALL
The current status of MARKET is:
ALL
```

その後、スポーツウェアに適用される coupon の値のみにアクセスする場合、ベース・ディメンション product を次のように制限します。

```
LIMIT product TO 'SPORTSWEAR'
```

結合ディメンションを制御する方法

結合ディメンションは、次のいずれかの方法で制限できます。

- ベース・ディメンションの制限
- 結合ディメンション自体の制限

> **参照：** 結合ディメンションの詳細は、3-17 ページの「疎密なデータを効率的に処理する変数の定義」を参照してください。

値の組合せを使用した結合ディメンションの制限

結合ディメンションを値のリストに制限するには、次の操作を実行します。

- 実際の値を指定し、それぞれの値の組合せを山カッコで囲みます。

```
LIMIT proddist TO <'TENTS' 'BOSTON'> <'FOOTWEAR' 'DENVER'>
```

- その値の変数名を使用して、その組合せを山カッコで囲みます。

```
prodname = 'CANOES'
distname = 'SEATTLE'
LIMIT proddist TO <prodname distname>
```

- 複数行のリストを作成し、各行で組合せを山カッコで囲み、¥n（改行エスケープ・シーケンス）で区切ります。

```
namelist = mytext = '<¥'TENTS¥' ¥'BOSTON¥'>¥n <¥'FOOTWEAR¥' ¥'DENVER¥'>'
LIMIT proddist TO namelist
```

ベース・ディメンションの値を使用した結合ディメンションの制限

結合ディメンションとそのベース・ディメンション間には暗黙的なリレーションが存在するため、ベース・ディメンションを制限することによって結合ディメンションを制限できます。

たとえば、次のコマンドを実行すると、proddist という名前の結合ディメンションが、ベース・ディメンション product の値の 1 つに CANOES を含むすべての結合値に制限されます。

```
LIMIT proddist TO product 'CANOES'
```

連結ディメンションの制限

連結ディメンションの現行のステータス・リストは、そのベース・ディメンションの現行のステータス・リストと異なります。ただし、連結ディメンションを制限するには、そのベース・ディメンションの値を指定します。

例 6-9 では、連結ディメンション reg.dist.cc のベース・ディメンションは、単純なディメンション region および結合ディメンション proddist です。この例では、連結ディメンションが WEST 地域に制限され、proddist が結合値 TENTS DENVER および RACQUETS DENVER に制限され、その連結ディメンションの値がレポートされます。

例 6-9 連結ディメンションのベース・ディメンションの制限

```
LIMIT reg.dist.ccdim TO region'WEST'
LIMIT reg.proddist.ccdim ADD proddist <'TENTS' 'DENVER'> -
<'RACQUETS', 'DENVER'>

REPORT reg.proddist.ccdim

REG.PRODDIST.CCDIM
------------------------------
<REGION: WEST>
<PRODDIST: <TENTS, DENVER>>
<PRODDIST: <RACQUETS, DENVER>>
```

NULL ステータスの処理

ディメンションの現行のステータス・リストは、NULL のステータスが許可されるように明示的に指定した場合にのみ NULL（空のステータス）に設定できます。これを行うには、次のいずれかの方法を実行します。

- OKNULLSTATUS オプションを yes に設定します。これによって、LIMIT コマンドに IFNONE 引数が指定されている場合以外は、常に NULL のステータスが許可されます。
- LIMIT コマンドで NULL キーワードを使用して、特定のディメンションまたは値セットのステータスを NULL に設定します。これを行うには、TO NULL または KEEP NULL を指定します。これによって、この LIMIT コマンドに対してのみ、NULL のステータスが許可されます。

これらのいずれかの方法を使用して NULL のステータスを許可していない場合、結果が NULL になる LIMIT コマンドを実行しても、ステータスが NULL に変更されません。ステータスは、コマンドの発行前と同じままになります。

1 つの MAINTAIN コマンドで IFNONE と NULL キーワードを同時に使用することはできません。

プログラムでの NULL のステータスの管理

LIMIT コマンドに IFNONE 引数を指定すると、ディメンションのステータスが NULL に設定されている場合は、通常の方法でプログラムが実行されません。そのため、IFNONE を指定すると、OKNULLSTATUS が YES の場合でも、IFNONE ラベルに分岐が実行され、ステータスが NULL に設定されません。IFNONE とともに NULL キーワードを指定した場合、非一貫性によってエラーが発生します。

ヒント： IFNONE 引数を使用すると、ラベルへの分岐が実行されるのみであるため、NULL のステータスの処理に対する柔軟性が制限されます。柔軟性を向上するには、ステータスが NULL の場合に実行を分岐するかどうかを制御する OKNULLSTATUS オプション、または NULL のステータスをテストする WHILE ループの使用を検討する必要があります。

NULL 値へのステータスの制限時のエラー

存在しない値を明示的にリストしないかぎり、値が含まれていないディメンションまたは値セットのステータスの制限を試行しても、エラーは発生しません。たとえば、新しく定義したディメンション WEEK に任意の値を追加した場合、次のコマンドを実行してもエラーは発生しません。

```
LIMIT week TO FIRST 10
```

ただし、次のコマンドを実行すると、PETE は値ではないためエラーが発生します。

```
LIMIT week TO 'PETE'
```

同様に、次のコマンドを実行すると、WEEK の位置 20 に値が存在しないため、エラーが発生します。

```
LIMIT week TO 20
```

値セットの処理

値セットは、特定のディメンションのディメンション値のリストを含むアナリティック・ワークスペース・オブジェクトです。値セットを使用して、後で使用するためにディメンションのステータスを保存します。値セット内の値は、OLAP セッションにまたがって保存できます。アナリティック・ワークスペースのアタッチ時には、各ディメンションにはデフォルトのステータス・リスト内のすべての値が含まれています。その後、任意のディメンションを、そのディメンションの値セットに格納されている値に制限できます。初めて定義された値セットの値は NULL です。値セットの定義後、LIMIT コマンドを使用して、ディメンションから値セットに値を割り当てます。値セットは、ディメンションに対して LIMIT コマンドを使用する多くの場合に使用できます。たとえば、LIMIT コマンドを使用して、値セット内の一連の値を追加、削除および置換できます。

値セットの作成

値セットを作成するには、次の手順を実行します。

1. ディメンション値の値セットを定義します。VALUESET キーワードを指定して DEFINE コマンドを使用します。
2. 対応する値セットを作成するディメンションを、保存する値に制限します。
3. 手順 1 で作成した値セットを、手順 2 で制限したディメンションに制限します。

例 6-10 値セットの作成

この例では、lineset という名前の値セットが定義されます。この値セットは line によってディメンション化されているため、line ディメンションの現在の値によって制限できます。

次のコマンドを実行すると、line ディメンションが最初の 2 つの値に制限され、line の現行のステータスが表示されます。

```
LIMIT line TO FIRST 2
STATUS line

The current status of LINE is:
REVENUE, COGS
```

これらのコマンドでは、lineset という値セットが定義され、それが line ディメンションの現行のステータス・リストに設定されて、その値が表示されます。LD コマンドを使用すると、オブジェクトに説明が添付されます。

```
DEFINE lineset VALUESET line
LD Valueset for LINE dimension values
LIMIT lineset TO line
SHOW VALUES(lineset)

REVENUE
COGS
```

値セットを使用した制限

値セットの定義後、それを使用して、1 つの LIMIT コマンドでディメンションを制限できます。

たとえば、次のコマンドを実行すると、line ディメンションが、lineset 値セットに格納された値に制限され、line の新しいステータスが表示されます。

```
LIMIT line TO lineset
STATUS line

The current status of LINE is:
REVENUE, COGS
```

例 6-11 値セットを使用した制限

次のコマンドを実行すると、district が、1996 年のスポーツウェアの売上が 1,000,000 ドルを超えた地区に制限されます。district ディメンションの現行のステータス・リストは、値セット SPORTS.DISTRICT に保存されます。値セットを一度作成すると、1 つの LIMIT コマンドを使用して、district ディメンションを同じ値に制限できます。

```
DEFINE sports.district VALUESET district
LIMIT product TO 'SPORTSWEAR'
LIMIT month TO year 'YR96'
LIMIT sports.district TO TOTAL(sales district) GT 1000000
LIMIT district TO sports.district
```

STATUS コマンドを実行すると、district の新しいステータスが表示されます。

```
STATUS district

The current status of DISTRICT is:
ATLANTA TO DENVER
```

値セットの値の変更

LIMIT コマンドを使用して、値セット内の値を変更できます。この方法で LIMIT コマンドを使用する単純な構文を次に示します。

```
LIMIT valueset keyword selection
```

valueset 引数には、変更する値セットの名前を指定します。

指定する *keyword* によって、現在値セット内に存在する値に対するコマンドの動作が決まります。次の表に、キーワードの使用方法を示します。

操作	LIMITコマンドとともに使用するキーワード
現在の値セット内に存在する値を新しい値に置き換える。	TO または COMPLEMENT キーワードのいずれか
現在の値セットから値を削除する。	REMOVE または KEEP キーワードのいずれか
値セットを拡張する。	ADD または INSERT キーワードのいずれか
値セット内の値をソートする。	SORT キーワード

selection 引数には、値セットに割り当てる値を決定するための選択基準を指定します。通常、LIMIT コマンドを使用してディメンションを制限する際に使用可能な値セットの値を、LIMIT コマンドに選択するときと同じ引数を使用できます。

値セットの値の識別および取得

値セット内のディメンション値を識別および取得するには、次のコマンドおよびファンクションを使用します。

コマンドまたはファンクション	説明
INSTAT ファンクション	ディメンション値が値セット内に存在するかどうかを確認します。
STATFIRST ファンクション	値セット内の最初の値を取得します。
STATLAST ファンクション	値セット内の最後の値を取得します。
STATUS コマンド	値セット内の 1 つ以上の値のステータスを現行の送信ファイルに送信します。
VALUES ファンクション	値セット内の値を取得します。INTEGER キーワードを指定すると、行ごとに 1 つのディメンション値を含む複数行のテキスト値が戻されます。このキーワードを指定しないと、値セット内ではなく、既存のディメンション内の値の位置番号が整数として戻されます。

値セット内の値の取得

アナリティック・ワークスペースに、値 JAN95、MAY95 および DEC95 を持つ monthset という名前の値セットが存在すると想定します。VALUES ファンクションを使用して、この値セット内の値をリストできます。

次の OLAP DML コマンドを実行すると、次に示す出力が生成されます。

```
SHOW VALUES(monthset)

JAN95
MAY95
DEC95
```

値セット内の値のディメンション位置の取得

monthset 値セット内の実際の値自体ではなく、それらの値の位置を取得する必要があると想定します。値の位置を取得するには、VALUES ファンクションとともに INTEGER キーワードを使用します。このキーワードを使用すると、値セットに含まれる実際のディメンション値ではなく、その位置番号が戻されます。戻される位置番号は、値セット内の位置ではなく、値セットが基づいているディメンション内の位置を表しています。

次の OLAP DML コマンドを実行すると、次に示す出力が生成されます。

```
SHOW VALUES(monthset INTEGER)

61
65
72
```

値 JAN95、MAY95、DEC95 は monthset 内でそれぞれ 1 番目、2 番目、3 番目の値ですが、month ディメンションではそれぞれ 61 番目、65 番目、72 番目と表示されています。

第 II 部

アプリケーション開発

第 II 部では、アプリケーション開発者向けの情報を示します。

第 II 部に含まれる章は、次のとおりです。

- 第 7 章「プログラムの開発」
- 第 8 章「モデルの使用」
- 第 9 章「データのアロケーション」

7

プログラムの開発

この章では、OLAP DML で記述されたプログラムの作成、コンパイル、テストおよびコールについて説明します。この章では、次の項目について説明します。

OLAP DML プログラムの概要

OLAP DML プログラムは、OLAP DML で記述されています。このプログラムは、アナリティック・ワークスペースのデータを処理して、アナリティック・ワークスペースの管理または分析タスクを実行するために有効です。OLAP DML プログラムは、アナリティック・ワークスペースで繰り返し行う必要があるタスクを実行するために作成するか、または開発中のアプリケーションの一部として作成することができます。

OLAP DML プログラムには、値を戻さないプログラムと値を戻すプログラムの 2 つの主要なタイプがあります。値を戻すプログラムは、ユーザー定義ファンクションといいます。

値を戻さない OLAP DML プログラムは、スタンドアロン・プログラムとして使用するか、またはマルチプログラム・アプリケーションのメイン・プログラムまたはサブプログラムとして使用することができます。これらのプログラムは、OLAP DML コマンドと同様に動作します。

ユーザー定義ファンクションは、組込み OLAP DML ファンクションを使用する場合と同様に、コマンドおよび式で使用できます。

プログラムの形式とは異なり、内容はそのプログラムが実行するように作成されたジョブに関連し、プログラムの個々の行がその内容を提供します。特定の目的を果たすプログラム行については、他の章を参照してください。

プログラムの実行

値を戻さないプログラムは、CALL コマンドを使用してコールできます。引数はカッコで囲み、それらの引数は値によって渡されます。

たとえば、2 つの整数を加算する、addit という名前の単純なプログラムを作成すると想定します。アプリケーションのメイン・プログラムで次の CALL コマンドを使用すると、このプログラムをコールできます。

```
CALL addit (3, 4)
```

CALL コマンドを使用してプログラムをコールするための構文は次のとおりです。

```
CALL program-name [(arg1 [arg2 ...])]
```

program-name 引数は、コールするプログラムの名前です。

arg1... 引数はオプションであり、コールするプログラムで使用可能な任意の引数を指定します。引数は、プログラムで定義される順序と一致するように指定します。

ユーザー定義ファンクションの実行

ユーザー定義ファンクションは、値を戻すプログラムです。ユーザー定義ファンクションは、組込みファンクションを使用する場合と同様の方法でコールします。式でプログラムの名前のみを使用し、プログラムの引数が存在する場合はそれをカッコで囲みます。

次に例を示します。

- プログラム名はコマンドで式として使用できます。

  次の REPORT コマンドを実行すると、actual という単一の引数を持つユーザー定義ファンクション isrecent の戻り値が使用されます。

  ```
  REPORT isrecent(actual)
  ```

- = コマンドを使用すると、ファンクションの戻り値を変数に代入することができます。

  次のコマンドを実行すると、tempsales という名前のユーザー定義ファンクションの戻り値が、mytempsales という名前の一時変数に代入されます。

  ```
  mytempsales = tempsales
  ```

重要： CALL コマンドを使用してユーザー定義ファンクションを実行することもできますが、戻り値にはアクセスできません。

プログラムの定義および編集

プログラムは、ディメンションや変数と同様に、アナリティック・ワークスペース・オブジェクトです。プログラムを定義するには、DEFINE コマンドを使用します。次の例では、hello という名前のプログラムが定義されます。

```
DEFINE hello PROGRAM
```

プログラム・オブジェクトを定義した場合は、それにプログラムの本体を追加する必要があります。

OLAP Worksheet で提供されているエディタを使用すると、プログラム定義に内容を追加できます。

参照： OLAP Worksheet の使用方法の詳細は、1-5 ページの「OLAP Worksheet からアナリティック・ワークスペースへのアクセス」を参照してください。

プログラムを編集する場合の書式設定ガイドライン

プログラムに行を追加するときは、次の書式設定ガイドラインに従ってください。

- コードの各行には、最大 4000 バイトを含めることができます。

- 1 つのコマンドを次の行に続けるには、行の末尾にハイフン（-）を入力します。このハイフンは、継続文字といいます。

  継続文字は、テキスト・リテラルの途中には使用できません。

- 1 行に複数のコマンドを入力するには、それらのコマンドをセミコロン（;）で区切ります。
- リテラル・テキストを一重引用符（'）で囲みます。リテラル・テキスト内に一重引用符を含めるには、その前に円記号（¥）を付けます。
- コメントの前に二重引用符（"）を付けます。二重引用符が前に付いたコメントは、行の先頭またはいくつかのコマンドの後の行末に配置できます。

hello という名前の次のプログラムは、「Hello World」という句を表示します。

```
DEFINE hello PROGRAM
PROGRAM
SHOW 'Hello World'
END
```

参照： エスケープ・シーケンスの詳細は、3-6 ページの「エスケープ・シーケンス」を参照してください。

プログラムでの変数の使用

アナリティック・ワークスペースでデータを保持する変数は、永続変数です。この変数は、OLAP セッション間で保持されます。ただし、プログラムでデータの操作中に処理情報を保持するために使用される変数を保存する必要がない場合があります。一時変数およびローカル変数を定義して、アナリティック・ワークスペースに不要な変数を残さないようにできます。

- 一時変数は、現行セッション中のみ値を持ちます。アナリティック・ワークスペースを更新およびコミットすると、変数の定義のみが保存されます。アナリティック・ワークスペースをデタッチすると、データ値は廃棄されます。
- ローカル変数は、その変数を定義するプログラムの実行中のみ存在する単一セル変数です。ローカル変数は、プログラム内で一時変数のかわりに使用できます。

  ローカル変数はディメンションを持たないため、ディメンション化されたデータの格納には使用できません。ローカル変数はその変数を定義するプログラムの継続期間中のみ存在するため、1 つのプログラムのローカル変数に情報を格納して、その変数を別のプログラムで使用することはできません。ディメンション化されたデータを格納する必要がある場合、または複数のプログラムで情報を使用する必要がある場合は、かわりに一時変数を定義します。

グローバル設計方法とモジュール設計方法の比較

ほとんどの OLAP DML プログラムの用途は、データを操作することです。プログラミングの方法およびアプリケーションの要件に応じて、次のどちらかの方法を使用できます。

- すべてのプログラムがアクセス可能な永続変数の使用。この方法を使用すると、必要なプログラミング・オーバーヘッドが減少（定義数の減少など）しますが、モジュール性が低下します。注意しないと、複数のプログラムが永続変数の値を設定するときに相互に影響を及ぼす場合があります。
- プログラム引数、ローカル変数およびユーザー定義ファンクションからの戻り値の使用。この方法を使用する場合、明確な入力責任および出力責任を持つモジュール・プログラムを作成する必要があります。

ほとんどのアプリケーションでは、これらの方法を組み合せて、永続変数およびユーザー定義ファンクションが適宜使用されます。一般に、モジュール・プログラムの方が、読込み、デバッグおよびメンテナンスが簡単であると考えられています。

一時変数の定義

一時変数は、次の例に示すとおり、DEFINE コマンドで TEMP キーワードを使用して定義します。

```
DEFINE total.sales DECIMAL TEMP
```

一時変数を定義してプログラムで使用する場合、アナリティック・ワークスペースに一時データが残ることを防ぐために有効ですが、その場合もアナリティック・ワークスペースにオブジェクトは追加されます。ほとんどの単純なアプリケーションでは、いくつかの一時オブジェクトが追加されても問題はありません。ただし、多くのプログラムを必要とする複雑なアプリケーションでは、多数の一時オブジェクトが使用される場合があり、これがアプリケーションのパフォーマンスに影響を及ぼす場合があります。

一度定義した一時変数は、削除されるか、またはユーザーのセッションが終了するまで存在します。一時変数は、プログラムのクリーンアップの一部として削除するか、またはその一時変数がすでに存在しないことを条件に作成する必要があります。これによって、セッション中にその一時変数を再実行する際にエラーが発生しません。

ローカル変数の定義

ローカル変数は、すべての実行可能なコマンドより前の、プログラムの先頭に指定する必要があります。ローカル変数は、次の構文を持つ VARIABLE コマンドを使用して指定します。

```
VARIABLE name datatype
```

name 引数には、変数の名前を指定します。混同や問題を最小限に抑えるために、アナリティック・ワークスペース変数とローカル変数の両方に同じ名前を使用しないでください。アナリティック・ワークスペース変数とローカル変数の両方が同じ名前を持つ場合、通常、ローカル変数が優先されます。ただし、アナリティック・ワークスペース・オブジェクトを

操作するいくつかのコマンドおよびファンクション（OBJ ファンクションなど）では、定義された変数が優先されます。

datatype 引数には、ローカル変数のデータ型を指定します。データ型の詳細は、3-4 ページの「データ型」を参照してください。

次に示す west.rpt という名前のプログラムには、data および rpt.month という名前の 2 つのローカル変数の定義が含まれています。

```
DEFINE west.rpt PROGRAM
LD Produce report for Western Sales District
PROGRAM
VARIABLE data TEXT
VARIABLE rpt.month TEXT
LIMIT month TO LAST 3
    .
    .
    .
```

引数の指定

OLAP DML では、次の 2 つの方法によってプログラムで引数を指定できます。

- ARGUMENT コマンド。ARGUMENT コマンドを使用して、プログラムで引数を宣言できます。ARGUMENT コマンドを使用すると、単純な引数と複雑な引数（式など）の両方を使用できます。また、ARGUMENT コマンドは、1 つのプログラムから別のプログラムへ引数を渡したり、独自のユーザー定義ファンクションを作成する場合に有効です。
- ARG ファンクション。任意のプログラムで ARG、ARGS および ARGFR ファンクションを使用して、コマンドから引数を取得できます。これらのファンクションは、主に単純なテキスト引数の場合に有効です。

ARGUMENT コマンドの使用

ARGUMENT コマンドを使用すると、任意のデータ型、ディメンションまたは値セットの引数を宣言できます。すべての ARGUMENT コマンドは、プログラム内の最初の実行可能な行の前に配置する必要があります。プログラムを実行すると、宣言したこれらの引数は、そのプログラムの引数として指定した値で初期化されます。これによって、プログラムはローカル変数を使用する場合と同様にこれらの引数を使用できるようになります。

例 7-1 ARGUMENT コマンドの使用

レポートを生成する、product.rpt という名前のプログラムを作成すると想定します。このレポート・プログラムに引数を指定して、レポート内の NA 値用に表示されるテキストを指定する必要があります。product.rpt プログラムでは、宣言した引数 natext を = コマンドで使用して、引数として指定した値に NASPELL オプションを設定できます。

```
ARGUMENT natext TEXT
NASPELL = natext
```

次のコマンドを実行すると、Missing が NA 値用のテキストとして指定されます。

```
CALL product.rpt ('Missing')
```

この例では、一重引用符で囲まれたリテラル・テキストがテキスト引数の値を指定しています。ただし、次の例に示すような他のタイプのテキスト式も同様に適切に機能します。

```
DEFINE natemp VARIABLE TEXT TEMP
natemp = 'Missing'
CALL product.rpt (natemp)
```

複数の引数の使用

プログラムは、必要な数の引数を宣言できます。引数を指定してプログラムを実行すると、それらの引数は、順序に基づいて、プログラムで宣言された引数に対応付けられます。

プログラムを実行する場合、引数をカンマや他の記号ではなく空白で区切る必要があります。記号は、引数の一部として処理されます。

例 7-2 複数の引数の指定

product.rpt プログラムで、レポートのデータ列の列幅を指定する 2 つ目の引数を指定すると想定します。次に示すとおり、product.rpt プログラムに 2 つ目の ARGUMENT コマンドを追加して、COLWIDTH オプションの値の設定に使用する整数引数を宣言します。

```
ARGUMENT natext TEXT
ARGUMENT widthamt INTEGER
NASPELL = natext
COLWIDTH = widthamt
```

8 文字の列を指定するには、次のコマンドを使用して product.rpt プログラムを実行します。

```
CALL product.rpt ('Missing' 8)
```

product.rpt プログラムに 3 つ目の引数として製品名も指定する必要がある場合は、次に示すとおり、product.rpt プログラムに製品引数を処理する 3 つ目の ARGUMENT コマンドを追加し、この引数を使用して product ディメンションのステータスを設定します。

```
ARGUMENT natext TEXT
ARGUMENT widthamt INTEGER
ARGUMENT rptprod PRODUCT
NASPELL = natext
COLWIDTH = widthamt
LIMIT product TO rptprod
```

次のコマンドを使用して、product.rpt プログラムを実行できます。

```
CALL product.rpt ('Missing' 8 'TENTS')
```

この例では、アナリティック・ワークスペース内のすべてのディメンション値は大文字表記であると想定して、3 つ目の引数は大文字で指定されています。

アンパサンド置換によるテキストとしての引数の指定

多くの場合、プログラムには単純なテキスト引数が渡されます。ただし、複数のディメンション値や 1 つの式のテキストで構成された引数などの、より複雑なテキスト引数を渡す必要がある場合もあります。この場合、引数名が使用される状況にかかわらず、渡すテキストを指定したとおりに置換する必要があります。

テキスト引数をこの方法で処理することを指定するには、引数名をプログラムのコマンドラインで使用する場合に引数名の前にアンパサンドを付けます。引数をこのように指定することを、**アンパサンド置換**といいます。

アンパサンド置換を使用してアナリティック・ワークスペース・オブジェクトの（値ではなく）名前をプログラムに渡すと、その名前はプログラムによって認識されるため、プログラムはオブジェクト自体にアクセスできます。これは、プログラムが複数の操作でオブジェクトを操作する必要がある場合に有効です。

重要： アンパサンドを含むすべてのプログラム行は、コンパイルおよび保存できません。かわりに、行が実行時に評価され、プログラムの速度が低下する場合があります。したがって、別の方法を使用してパフォーマンスを最適化できる場合は、アンパサンド置換を使用しないでください。

例 7-3 複数のディメンション値の指定

前述の product.rpt プログラムに対して 2 つの製品を指定する場合は、それらの製品を処理する 2 つのディメンション値引数を宣言できます。ただし、LIMIT キーワードを使用して任意の数の製品を指定できるようにするには、単一の引数をアンパサンド置換とともに使用します。

プログラムで次のコマンドを使用すると想定します。

```
ARGUMENT natext TEXT
ARGUMENT widthamt INTEGER
ARGUMENT rptprod TEXT
    .
    .
    .
LIMIT product TO &rptprod
```

次のようにプログラムを実行し、最初の 3 つの製品をレポートに表示することを指定できます。

```
CALL product.rpt ('Missing' 8 'first 3')
```

「first 3」を空白で区切られた 2 つの個別の引数ではなく単一の引数として指定するために、一重引用符で囲む必要があります。アンパサンドによって、LIMIT が 'first 3' をディメンション値ではなくキーワード式として解釈します。

例 7-4 式のテキストの指定

REPORT コマンドを含む custom.rpt という名前のプログラムが存在します。次に示すとおり、このプログラムを使用して、単一の変数のみでなく sales - expense などの式の値も表す必要があると想定します。

```
custom.rpt 'sales - expense'
```

> **注意：** 式は一重引用符で囲む必要があります。式には記号（マイナス記号）が含まれているため、式全体が単一の引数であることを示すために一重引用符が必要です。

custom.rpt プログラムで次のコマンドを実行すると、この式のレポートを生成できます。

```
ARGUMENT rptexp TEXT
REPORT &rptexp
```

オブジェクト名およびキーワードの指定

次のタイプの引数では、常にアンパサンドを使用して適切な置換を行う必要があります。

- アナリティック・ワークスペース・オブジェクトの名前（units や product など）
- コマンドのキーワード（REPORT コマンドの COMMA や NOCOMMA、SORT コマンドの A や D など）

例 7-5 アナリティック・ワークスペース・オブジェクト名およびキーワードの指定

引数として指定された変数に関するレポートを生成し、別の引数で指定された順に product ディメンションをソートする sales.rpt というプログラムを設計すると想定します。次のようなコマンドを実行して、sales.rpt プログラムを実行します。

```
sales.rpt units d
```

sales.rpt プログラムで、次のコマンドを使用できます。

```
ARGUMENT varname TEXT
ARGUMENT sortkey TEXT
SORT product &sortkey &varname
REPORT &varname
```

引数の置換後、次のコマンドが sales.rpt プログラムで実行されます。

```
SORT product D units
REPORT units
```

参照： アンパサンド置換の詳細は、4-27 ページの「置換式」を参照してください。

ユーザー定義ファンクションの作成

値を戻す OLAP DML プログラムは、ユーザー定義ファンクションといいます。ユーザー定義ファンクションは、コマンドおよび式で使用できます。

次に示すとおり、ユーザー定義ファンクションには、RETURN コマンドおよび後続の式が含まれます。

```
RETURN expression
```

RETURN コマンドは、プログラムの終了時に単一の値を戻します。

ユーザー定義ファンクションのデータ型

ユーザー定義ファンクションを作成する場合、次の構文の DEFINE コマンドを使用して、データ型またはディメンション名によってプログラムを定義します。

```
DEFINE programname PROGRAM [datatype|dimension]
```

datatype 引数には、プログラムがファンクションとしてコールされた場合に戻す値のデータ型を指定します。

dimension 引数には、プログラムがファンクションとしてコールされた場合に値を戻すディメンションの名前を指定します。この戻り値は、位置（整数）ではなくそのディメンションの単一の値です。そのディメンションは、プログラムと同じアナリティック・ワークスペースで定義されている必要があります。プログラムによって戻される値は、定義で指定したデータ型を持ちます。ディメンション名を指定すると、プログラムはそのディメンションの値を戻します。

プログラムの戻り式は、その定義で指定したデータ型と一致する必要があります。戻り値のデータ型がその定義で指定したデータ型と一致しない場合、その戻り値は定義のデータ型に変換されます。

プログラムのデータ型を指定しない場合、戻り値はコール元が要求するデータ型に変換されます。

ユーザー定義ファンクションの引数

ユーザー定義ファンクションは、引数を受け入れることができます。ユーザー定義ファンクションは、単一の値のみを戻します。ただし、ディメンションに対してループを実行するコンテキスト（REPORT コマンドなど）でユーザー定義ファンクションの引数を指定すると、ファンクションはその引数と同じディメンションでの結果を戻します。

引数は、プログラム内で ARGUMENT コマンドを使用して宣言し、プログラム名の後にカッコで囲んで指定する必要があります。

参照： プログラムでの引数の使用の詳細は、7-6 ページの「引数の指定」を参照してください。

例 7-6 ユーザー定義ファンクション

アナリティック・ワークスペースに、product、district および month ディメンションによってディメンション化された units.plan という名前の変数が含まれていると想定します。この変数は、製品の販売予想数を示す整数データを保持します。

また、units_goals_met という名前のプログラムも定義すると想定します。このプログラムはユーザー定義ファンクションです。これは、units.plan 変数の特定のセルを指定する 3 つのディメンション値引数を受け入れ、そのセルの販売実績数を指定する 4 つ目の引数を受け入れます。このプログラムは、コール元プログラムにブール値を戻します。このプログラムは、実績値が計画値から 10% の範囲内である場合は YES を戻し、実績値が計画値から 10% の範囲内でない場合は NO を戻します。

units_goals_met プログラムの定義を次に示します。

```
DEFINE units_goal_met PROGRAM BOOLEAN
LD Tests whether actual units met the planned estimate
"Program Initialization
ARGUMENT userprod  TEXT
ARGUMENT userdist  TEXT
ARGUMENT usermonth TEXT
ARGUMENT userunits integer
VARIABLE answer boolean
TRAP ON errorlabel
PUSH product district month
"Program Body
LIMIT product TO userprod
LIMIT district TO userdist
LIMIT month TO usermonth
IF (units.plan - userunits) / units.plan GT .10
   THEN answer = NO
   ELSE answer = YES
"Normal Exit
POP product district month
```

```
RETURN answer
"Abnormal Exit
errorlabel:
POP product district month
SIGNAL errorname errortext
END
```

units_goal_met プログラムを実行し、戻り値を success という変数に格納するには、次の代入文を使用します。

```
success = units_goal_met('TENTS' 'BOSTON' 'JUN96' 2000)
```

実行フローの制御

通常、プログラムの行は順次（線形的に）実行されます。ただし、適切に設計されたプログラムでは、実行のパスを適宜リダイレクトするコマンドを使用して実行フローを制御できます。

次の制御構造を使用すると、コマンドの実行順序を変更できます。

コマンドまたはキーワード	操作	操作をトリガーするイベント
IF コマンド	代替のコマンドまたはコマンド・グループを実行します。	指定したブール条件を満たしているかどうか
WHILE コマンド	コマンド・グループを繰り返し実行します。	指定したブール条件を満たしている場合
FOR コマンド	コマンドまたはコマンド・グループを実行します。	ディメンションの値ごとに 1 回
GOTO コマンド	ラベルが付いた特定の場所に分岐します。	コマンドの発行
SWITCH コマンド	複数のパスを持つブランチで特定のブランチに分岐します。	特定の基準を満たしている場合
TRAP コマンド	ラベルが付いた特定の場所に分岐します。	プログラムの実行中にエラーが発生した場合
LIMIT、REPORT、ROW または HEADING コマンドの IFNONE キーワード	ラベルが付いた特定の場所に分岐します。	ステータスの設定を試行すると、値が設定されないか、または NULL ステータスが設定される場合
RETURN コマンド	プログラムの最終コマンドの前にプログラムの分岐から戻るか、またはコール元プログラムに戻ります。	コマンドの発行

ラベルを構成する場合のガイドライン

制御構造を使用して特定の場所に分岐する場合、その場所を明確に識別するために、その場所用のラベルを指定する必要があります。ラベルを作成する場合は、次のガイドラインに従ってください。

- ラベルの最初の文字は、文字、ピリオド（.）またはアンダースコア（_）である必要があります。
- ラベルの残りの文字には、文字、数字、ピリオドまたはアンダースコアを任意に組み合せて使用できます。
- ラベルの直後には、コロン（:）を付ける必要があります。
- ラベルの最初の 8 バイトが一意であることを確認します。（キャラクタ・セットによって 1 バイトが 1 文字に相当する場合と相当しない場合があることに注意してください。）ラベルには、最大 3999 バイト（テキスト行の最大長からラベルを識別するコロン用の 1 バイトを引いたもの）を含めることができます。ただし、ラベル名の最初の 8 バイトのみが使用されるため、最初の 8 バイトが一意でない場合、8 バイトを超えるラベル名の使用時に問題が発生することがあります。

GOTO コマンドの代替方法

GOTO はプログラム内での分岐を容易にしますが、このコマンドを頻繁に使用すると、プログラムのロジックが不明瞭になり、そのフローに従うことが困難になる場合があります。これは、特に、複数のラベルおよび多くのコードをスキップする複数の GOTO コマンドを含む複雑なプログラムの場合に当てはまります。

プログラムのロジックを明瞭にしておくために、GOTO の使用は最小限に抑えてください。

GOTO コマンドが最適なプログラミング方法である場合もありますが、多くの場合、より適切な代替方法が存在します。次に例を示します。

- IF コマンドで GOTO コマンドを使用するかわりに、多くの場合、IF コマンド自体内の DO コマンドと DOEND コマンドの間に代替のコマンド・セットを配置できます。
- 各コマンド・セットが長い場合、またはコマンド・セットをプログラム内の複数の箇所で使用する場合は、コマンド・セットをサブプログラム内に配置することを検討できます。この場合、IF コマンドを使用して 2 つの異なるプログラムのどちらかを選択するか、または SWITCH コマンドを使用して多くの異なるプログラムのいずれかを選択することができます。

例 7-7 FOR コマンドを使用したディメンション値のループ処理

FOR コマンドは、ディメンションの現行のステータス内の値ごとにコマンドをループ実行します。FOR コマンドを実行する前に、ディメンションを必要な値に制限する必要があります。たとえば、次に示すとおり、各製品の価格を示す一連の出力行を生成できます。

```
LIMIT month TO FIRST 1
LIMIT product TO ALL
FOR product
SHOW JOINCHARS('Price for ' product ': $' price)
```

各出力行の形式は次のとおりです。

```
Price for TENTS: $165.50
```

データが多次元である場合は、FOR コマンドで複数のディメンションを指定して、処理順序を制御できます。たとえば、次のコマンドを実行すると、units データのディメンション値が処理される順序を制御できます。

```
FOR month district product
   units = ...
```

この代入文を実行すると、month ディメンションが最も遅く変化し、district ディメンションが次に遅く変化し、product ディメンションが最も速く変化します。したがって、次の地区を処理する前に最初の地区のすべての製品に対してループが実行され、次の月を処理する前に最初の月のすべての地区に対してループが実行されます。

FOR ループ内では、コマンドをループ実行する間、指定された各ディメンションは一時的に 1 つの値に制限されます。したがって、ループ内でディメンション値の特定の組合せを処理できます。

例 7-8 FOR ループでの DO/DOEND の使用

単位売上の実績数が units という変数に保存され、単位売上の予想数が units.plan という名前の変数に保存されている場合、ループ内の次のコードはディメンション値の同じ組合せに対するこれらの数値を比較できます。

```
LIMIT month TO FIRST 1
LIMIT product TO ALL
LIMIT district TO ALL
FOR district product
   DO
     IF (units.plan - units)/units.plan GT .1
     THEN SHOW JOINCHARS(-
       'Unit sales for ' product ' in ' -
       district ' are not within 10% of plan.')
   DOEND
```

これらのコードは、次の手順で処理されます。

1. データは、特定の月に制限されます。
2. すべての地区および製品はステータス内に配置され、FOR ループに入ります。
3. FOR ループ内で、実績値が計画値に対してテストされます。BOSTON での TENTS の単位売上数が計画値のマイナス 10% 未満である場合、次のメッセージが現行の送信ファイルに送信されます。

   ```
   Unit sales for TENTS in BOSTON are not within 10% of plan.
   ```

4. すべての製品を処理した後、最初の地区に対する FOR ループが完了します。
5. 2 つ目の地区、3 つ目の地区というようにループが実行されます。

注意： FOR ループの実行中、FOR コマンドで指定された各ディメンションは一時的に 1 つの値に制限されます。FOR ループで district は指定し、product は指定しなかった場合、FOR ループの実行中、product のすべての値がステータス内に配置されます。その後、IF コマンドが product ディメンションの最初の値についてのみデータをテストします。

例 7-9 NULL ステータスの設定を回避するための分岐

プログラムが、単位売上数が 500 個を超える製品のみを含むように product ディメンションのステータスの設定または改良を試行する場合があります。単位売上数が 500 個を超える製品が存在しない場合は、次に示すとおり、IFNONE キーワードを使用して、実行が novals ラベルに分岐するように指定できます。

```
LIMIT product KEEP units GT 500 IFNONE novals
```

novals ラベルの後続のコマンドでは、販売数が 500 個を超える製品が存在しないという特殊な状況を処理できます。

例 7-10 NULL ステータスの設定後の分岐

IFNONE ラベルに分岐するかわりに、次に示すとおり、OKNULLSTATUS オプションを使用してディメンションの NULL ステータスを処理することもできます。OKNULLSTATUS を YES に設定すると、ディメンションのステータスを NULL に設定できるようになります。その後、ステータスが NULL ステータスであるかどうかを確認し、IF コマンドを使用して適切なコマンドを実行するか、または SWITCH コマンドでの 1 つのケースとして NULL ステータスを処理できます。

```
OKNULLSTATUS = YES
LIMIT month TO sales GT salesnum
IF STATLEN(month) LT 1
   THEN GOTO showerr
```

出力の送信

出力をファイルに送信するには、OUTFILE コマンドの後にディレクトリ別名およびファイル名を指定して、その 2 つをスラッシュ（/）で区切ります。指定した名前を使用してファイルが作成されます。OUTFILE コマンドを実行する前に、CDA コマンドを使用してカレント・ディレクトリ別名を指定できます。この場合、Oracle OLAP はファイルをカレント・ディレクトリ別名で作成する必要があると想定するため、OUTFILE コマンドでディレクトリ別名を指定する必要はありません。

ディレクトリ別名は、データベースで定義され、ディレクトリへのアクセスを制御します。ご使用のデータベース・ユーザー名が読込み / 書込み権限を持つディレクトリ別名の名前については、Oracle DBA に連絡してください。ファイル名は、ご使用のオペレーティング・システムの標準ファイル名形式に従って指定する必要があります。

OUTFILE コマンドは、後続のすべての出力の転送を変更します。したがって、プログラムがレポートをファイルに転送する場合は、そのプログラムを終了する前に出力をデフォルトの送信ファイルに再転送する必要があります。後続の出力をデフォルトの送信ファイルに送信する場合は、OUTFILE EOF コマンドをレポート・コマンドの直後に配置します。また、エラーの発生によってプログラムが異常終了した場合に OUTFILE EOF コマンドが実行されるようにするために、異常終了セクションにもこのコマンドを配置します。

OLAP Worksheet で作業している場合、デフォルトの送信ファイルは「Response」ウィンドウです。現行の宛先は、現行の送信ファイルといいます。

例 7-11 ファイルへの出力の送信

year.end.sales というプログラムが存在し、このプログラムが作成したレポートをファイルに保存すると想定します。次のコマンドを入力して、レポートのファイルを作成します。この例では、userfiles はディレクトリ別名で、yearend.txt はファイルの名前です。

```
OUTFILE 'userfiles/yearend.txt'
year.end.sales
OUTFILE EOF
```

この時点で、ファイルには year.end.sales のレポートが含まれています。OUTFILE の APPEND キーワードを使用すると、同じファイルにさらにレポートを追加できます。year.end.expenses という名前の別のプログラムが存在すると想定します。次のコマンドを使用して、そのレポートをファイルに追加します。APPEND を使用しない場合、OUTFILE コマンドは経費レポートを上書きすることに注意してください。

```
OUTFILE APPEND 'userfiles/yearend.txt'
year.end.expenses
OUTFILE EOF
```

エラー・メッセージの取得

次に示すとおり、ECHOPROMPT オプションを YES に設定すると、エラー・メッセージを取得できます。

```
ECHOPROMPT = YES
```

ECHOPROMPT を YES に設定すると、出力行とともに入力行およびエラー・メッセージが現行の送信ファイルにエコー表示されます。OUTFILE コマンドまたは DBGOUTFILE コマンドを使用すると、エラー・メッセージをファイルに取得できます。DBGOUTFILE の詳細は、7-28 ページの「デバッグ・ファイルへの出力の送信」を参照してください。

設定を変更するたびに、PUSH コマンドおよび POP コマンドを使用してその元の値を保存およびリストアする必要があります。

セッション環境の保持

モジュール設計方法のメリットの 1 つは、各プログラムが明確に定義された責任領域を持ち、他のプログラムの動作に影響を及ぼさないことです。これを可能にするには、各プログラムがグローバル設定を変更する前にそれらの設定を保存し、実行を終了する前にグローバル設定をリストアする必要があります。

環境設定には次の 2 つのタイプが存在します。

- セッション環境。セッション環境は、プログラムが実行される前に有効になっているディメンションのステータス、オプション値および出力先で構成されます。
- プログラム環境。プログラム環境は、プログラムで使用するディメンションのステータス、オプション値および出力先で構成されます。

プログラム環境の変更

プログラム内でタスクを実行するには、多くの場合、出力先、またはいくつかのディメンション値とオプション値を変更する必要があります。たとえば、常に最新の 6 か月の売上データを表示する月間売上レポートを実行する場合があります。この場合、データを整数で表示し、売上高が 0（ゼロ）である場合に「No Sales」というテキストを追加し、レポートをファイルに送信すると想定します。このプログラム環境を設定するには、プログラムで次のコマンドを使用します。

```
LIMIT month TO LAST 6
DECIMALS = 0
ZSPELL = 'No Sales'
OUTFILE monsales.txt
```

セッション環境の混乱を回避するには、プログラムの初期化セクションで、プログラムで設定するディメンションおよびオプションの値を保存する必要があります。他のプログラムが変更された値があるかどうかを確認する必要がないように、プログラムの正常終了セクショ

ンおよび異常終了セクションで、保存された環境をリストアできます。また、出力をファイルに送信した場合、終了セクションで、出力先をデフォルトの送信ファイルに戻す必要があります。

環境を保存およびリストアする方法

次の推奨項目に従うと、プログラムまたはセッションの環境を保存できます。

- ディメンションの現行のステータスや値セット、オプションまたは単一セル変数の値を現行のプログラムで使用するために保存する場合は、PUSHLEVEL コマンドおよび PUSH コマンドを使用します。現行のステータスの値は、POPLEVEL コマンドおよび POP コマンドを使用してリストアできます。
- ディメンションの現行のステータスや値セット、オプション、単一セル変数または単一セル・リレーションの値を現行セッションで使用するために保存、アクセスまたは更新する場合は、名前付きコンテキストを使用します。コンテキストを定義するには、CONTEXT コマンドを使用します。

コンテキストは、セッション中に使用するためにオブジェクト値を保存するための、最も高度な方法です。コンテキストを使用すると、保存したオブジェクト値に対してアクセス、更新およびコミットを行うことができます。一方、PUSH および POP を使用すると、値を保存およびリストアできます。通常、PUSH コマンドおよび POP コマンドは、プログラム実行中にのみ適用される変更を行うために、プログラム内で使用します。

ディメンションのステータスまたはオプション値の保存

PUSH コマンドを実行すると、ディメンションの現行のステータス、オプションの値または単一セル変数の値が保存されます。たとえば、プログラムの継続期間中、異なる値に設定できるように DECIMALS オプションの現行値を保存するには、初期化セクションで次のコマンドを使用します。

```
PUSH DECIMALS
```

オプションの元の値を保存または後でリストアする場合、その値を知っている必要はありません。保存された値は、次の POP コマンドを使用してリストアできます。

```
POP DECIMALS
```

プログラムが正常に終了した場合のみでなく、エラーの発生によってプログラムが異常終了した場合にも POP コマンドが実行されるようにする必要があります。そのため、POP コマンドをプログラムの正常終了セクションと異常終了セクションに配置する必要があります。

複数の値の同時保存

次に示すとおり、1 つ以上のディメンションのステータスおよび任意の数のオプションと変数の値を 1 つの PUSH コマンドで保存し、1 つの POP コマンドを使用して値をリストアすることができます。

```
PUSH month DECIMALS ZSPELL
        .
        .
        .
POP month DECIMALS ZSPELL
```

レベル・マーカーの使用

複数のディメンションおよびオプションの値を保存する場合、PUSHLEVEL コマンドおよび POPLEVEL コマンドを使用すると、セッション環境を保存およびリストアする際に有効です。

最初に、PUSHLEVEL コマンドを使用してレベル・マーカーを設定します。レベル・マーカーを設定した後で、PUSH コマンドを使用して、ディメンションのステータス、およびオプションまたは単一セル変数の値を保存します。

PUSHLEVEL コマンドと POPLEVEL コマンドの間に複数の PUSH コマンドを配置すると、それらの PUSH コマンドで指定されたすべてのオブジェクトが 1 つの POPLEVEL コマンドによってリストアされます。

PUSHLEVEL および POPLEVEL を使用すると、オブジェクトのリストを 1 回入力する必要があるのみであるため、プログラム作成時の入力操作が軽減されます。また、リストからオブジェクトが欠落したり、オブジェクト名のスペルを間違える危険も減ります。

例 7-12 レベル・マーカーの作成

たとえば、次の PUSHLEVEL コマンドを使用して firstlevel というレベル・マーカーを設定した後、PUSH を使用して現行値を保存することができます。

```
PUSHLEVEL 'firstlevel'
PUSH month DECIMALS ZSPELL
```

レベル・マーカーには、一重引用符で囲んだ任意のテキストを指定できます。また、単一セル ID または TEXT 変数の名前を指定することもでき、それらの値がレベル・マーカーの名前になります。その後、プログラムの終了セクションで次のような POPLEVEL コマンドを使用して、firstlevel マーカーの設定以降に保存したすべての値をリストアできます。

```
POPLEVEL 'firstlevel'
```

例 7-13 PUSHLEVEL コマンドと POPLEVEL コマンドのネスト

PUSHLEVEL コマンドと POPLEVEL コマンドをネストして、特定の値グループをプログラム内の 1 箇所に保存し、他の値グループをプログラム内の別の箇所に保存することができま

す。次の例では、ネストした PUSHLEVEL コマンドと POPLEVEL コマンドの 2 つのセットを示します。

```
PUSHLEVEL 'firstlevel'
PUSH PAGESIZE DECIMALS "Saves values in FIRSTLEVEL
       .
       .
       .
PUSHLEVEL 'secondlevel'
PUSH month product     "Saves values in SECONDLEVEL
       .
       .
       .
POPLEVEL 'secondlevel' "Restores values in SECONDLEVEL
       .
       .
       .
POPLEVEL 'firstlevel'  "Restores values in FIRSTLEVEL
```

通常、1 つのプログラムで PUSHLEVEL コマンドと POPLEVEL コマンドの複数のセットを使用しません。ただし、1 つのプログラムが別のプログラムをコールし、各プログラムに PUSHLEVEL コマンドと POPLEVEL コマンドの 1 つのセットが含まれる場合は、ネスト機能が自動的に有効になります。

CONTEXT を使用した複数の値の同時保存

PUSHLEVEL および POPLEVEL を使用するかわりに、CONTEXT コマンドを使用できます。コンテキストを作成した後、そのコンテキストにディメンションの現行のステータスやオプション、単一セル変数、値セットおよび単一セル・リレーションの値を保存できます。その後、そのコンテキストからオブジェクト値の一部または全部をリストアできます。CONTEXT ファンクションは、コンテキスト内のオブジェクトに関する情報を戻します。

エラーの処理

適切に設計されたプログラムは、エラーを効率的に処理し、各エラーを有益な方法でレポートします。OLAP DML は、プログラム内のエラーを検出およびレポートするために有効な TRAP などのコマンドを提供します。

エラーを通知する方法

プログラム内でエラーが発生すると、そのエラーは通知されます。エラーを通知するために、次の操作が実行されます。

1. エラーの名前が ERRORNAME オプションに保存され、エラー・メッセージのテキストが ERRORTEXT オプションに保存されます。

2. ECHOPROMPT が YES の場合は、エラー・メッセージが現行の送信ファイルまたはデバッグ・ファイル（存在する場合）に送信されます。
3. エラー・トラップがオフの場合は、プログラムの実行が停止されます。エラー・トラップがオンの場合は、エラーがトラップされます。

エラーをトラップする方法

プログラムを正常に動作させるために、エラーの発生を予想し、エラーを処理するためのシステムを設定する必要があります。TRAP コマンドを使用すると、プログラムでエラー・トラップ・メカニズムをオンにできます。エラーが通知されたときにエラー・トラップがオンである場合、プログラムの実行は停止されません。かわりに、エラー・トラップ機能によって次の操作が実行されます。

1. エラー処理プロセス中に別のエラーが発生した場合に無限ループを回避するために、エラー・トラップ・メカニズムをオフにします。
2. TRAP コマンドで指定されたラベルに分岐します。
3. ラベルの後続のコマンドを実行します。

セッション環境の保存中のエラー処理

セッション環境の保存中に発生するエラーを適切に処理するには、次に示すとおり、TRAP コマンドの前に PUSHLEVEL コマンドを配置し、TRAP コマンドの後に PUSH コマンドを配置します。

```
PUSHLEVEL 'firstlevel'
TRAP ON error
PUSH . . .
```

プログラムの異常終了セクションに、error ラベル（後続のコロン付き）、およびセッション環境をリストアしてエラーを処理するコマンドを配置します。異常終了セクションは、次のようになります。

```
error:
POPLEVEL 'firstlevel'
OUTFILE EOF
```

これらのコマンドによって、保存されたディメンションのステータスおよびオプション値がリストアされ、出力がデフォルトの送信ファイルに再転送されます。

エラー・メッセージの抑制

特定のエラーの発生時に通常提供されるエラー・メッセージを生成しない場合は、TRAP コマンドで NOPRINT キーワードを使用します。

```
TRAP ON error NOPRINT
```

TRAP で NOPRINT キーワードを使用すると、制御が error ラベルに分岐し、エラーの発生時にエラー・メッセージが発行されません。その後、error ラベルの後続のコマンドが実行されます。

エラー・メッセージを抑制する場合、独自のメッセージを異常終了セクションに生成することがあります。次に示すとおり、SHOW コマンドを実行すると、指定したテキストは生成されますが、エラーは通知されません。

```
TRAP ON error NOPRINT
        .
        .
        .
error:
        .
        .
        .
SHOW 'The report will not be produced.'
```

プログラムは、メッセージを生成した後に次のコマンドを実行します。

発生したエラーの特定

すべてのエラーは名前を持ちます。エラーが通知されるたびに、そのエラー名が ERRORNAME オプションに保存されます。あるタイプのエラーが発生したときにある一連のアクティビティを実行し、別のタイプのエラーが発生したときに異なる一連のアクティビティを実行する場合は、ERRORNAME オプションの値をテストします。ERRORTEXT オプションには、エラーの説明が含まれます。

独自のエラー・メッセージの作成

コマンドまたはコマンド・シーケンスがその要件を満たしていない場合に発生するすべてのエラーは、自動的に通知されます。プログラムで、アプリケーションに対する追加の要件を設定できます。要件が満たされていない場合、SIGNAL コマンドを実行してエラーを通知できます。

エラーには、任意の名前を付けることができます。SIGNAL コマンドが実行されると、通常のエラー名と同様に、指定したエラー名が ERRORNAME オプションに保存されます。
SIGNAL コマンドで独自のエラー・メッセージを指定すると、通常のエラー・メッセージと同様に、独自のメッセージが生成されます。TRAP コマンドを使用してエラーをトラップする場合は、エラー・メッセージが生成された後に SIGNAL コマンドが TRAP ラベルに分岐します。

例 7-14 エラーの通知

プログラムが 1 ～ 9 か月分のデータを表示可能なレポートを生成すると想定します。プログラムが 9 より大きい引数値を使用してコールされた場合にエラーを通知することができます。次の例では、nummonths は 9 以下である必要がある引数の名前です。

```
select:
TRAP ON error
PUSH month
LIMIT month TO nummonths
IF STATLEN(month) GT 9
   THEN SIGNAL toomany -
     'You can specify no more than 9 months.'
REPORT DOWN district W 6 units
finish:
POP month
RETURN
error:
POP month
IF ERRORNAME EQ 'TOOMANY'
   THEN SHOW 'No report produced'
```

error ラベルに分岐せずに警告メッセージを生成する場合は、次の SHOW コマンドを使用します。

```
select:
LIMIT month TO nummonths
IF STATLEN(month) GT 9
   THEN DO
     SHOW 'You can select no more than 9 months.'
     GOTO finish
     DOEND
REPORT DOWN district W 6 units
finish:
POP month
RETURN
```

ネストしたプログラムでのエラーの処理

別のプログラムを実行するプログラムを作成する場合、2 つ目のプログラムは最初のプログラム内にネストされます。2 つ目のプログラムは、別のネストしたプログラムを実行する場合があります。

各プログラム内のエラー処理セクションでは、環境をリストアする必要があります。このセクションは、そのプログラム固有のすべての特殊なエラー状態も処理できます。たとえば、プログラムが独自のエラーを通知する場合、そのエラーをテストするコマンドを追加することができます。

ネストしたプログラムで発生した他のすべてのエラーは、プログラムの連鎖を経由して渡され、各プログラムで処理される必要があります。ネストしたプログラムの連鎖を経由してエラーを渡すには、エラー・メッセージを生成するタイミングに応じて、次の 2 つの方法のいずれかを使用します。

- エラー・メッセージをすぐに生成し、その後にプログラムの連鎖を経由してエラー状態を渡します。
- 最初にプログラムの連鎖を経由してエラーを渡し、連鎖の終わりにエラー・メッセージを生成します。

どちらの方法でも SIGNAL コマンドが使用されます。

例 7-15 エラー・メッセージの即時生成

エラー・メッセージをすぐに生成するには、ネストした各プログラムで TRAP コマンドを使用します。ただし、NOPRINT キーワードは使用しません。エラーが発生すると、すぐにエラー・メッセージが生成され、実行が trap ラベルに分岐します。

trap ラベルでは、必要な任意のエラー処理コマンドを実行し、環境をリストアします。その後、次に示すとおり、PRGERR キーワードを指定して SIGNAL コマンドを実行します。

```
SIGNAL PRGERR
```

SIGNAL コマンドで PRGERR キーワードを使用すると、エラー・メッセージが生成されず、名前 PRGERR は ERRORNAME に保存されません。SIGNAL コマンドはエラー状態を通知し、そのエラー状態が現行プログラムの実行元のプログラムに渡されます。コール元のプログラムに trap ラベルが含まれている場合は、実行がそのラベルに分岐します。

ネストしたプログラムの連鎖内の各プログラムで TRAP コマンドおよび SIGNAL コマンドをこのように使用すると、連鎖全体を経由してエラー状態を渡すことができます。各プログラムには、次のようなコマンドが含まれます。

```
TRAP ON error
   .
   .       "Body of program and normal exit commands
   .
RETURN
error:
   .
   .       "Error-handling and exit commands
   .
SIGNAL PRGERR
```

例 7-16 連鎖の終わりでのエラー・メッセージの生成

ネストしたプログラムの連鎖の終わりにエラー・メッセージを生成するには、NOPRINT キーワードを付けて TRAP コマンドを使用します。ネストしたプログラムでエラーが発生すると、実行が trap ラベルに分岐しますが、エラー・メッセージは抑制されます。

trap ラベルでは、必要な任意のエラー処理コマンドを実行し、環境をリストアします。その後、次の SIGNAL コマンドを実行します。

```
SIGNAL ERRORNAME ERRORTEXT
```

ERRORNAME オプションには元のエラーの名前が含まれ、ERRORTEXT オプションには元のエラーのエラー・メッセージが含まれます。前述の SIGNAL コマンドは、元のエラー名およびエラー・テキストをコール元のプログラムに渡します。コール元のプログラムに trap ラベルが含まれている場合は、実行がそのラベルに分岐します。

ネストしたプログラムの連鎖内の各プログラムで TRAP コマンドおよび SIGNAL コマンドをこのように使用すると、連鎖の終わりに元のエラー・メッセージが生成されます。各プログラムには、次のようなコマンドが含まれます。

```
TRAP ON error NOPRINT
   .
   .       "Body of program and normal exit commands
   .
RETURN
error:
   .
   .       "Error-handling and exit commands
   .
SIGNAL ERRORNAME ERRORTEXT
```

プログラムのコンパイル

COMPILE コマンドを使用すると、プログラムを明示的にコンパイルできます。プログラムを明示的にコンパイルしない場合、プログラムは最初に実行されたときにコンパイルされます。

プログラムをコンパイルすると、そのプログラムのコマンドが効率的に処理されるコードに変換され、そのプログラムの元のテキストよりはるかに高速に実行されます。プログラムでエラーが発生すると、コンパイルは完了せず、プログラムはコンパイルされていないとみなされます。

プログラムをコンパイルすると、そのプログラムを現行セッションで実行するたびにコンパイル済コードが使用されます。プログラムをコンパイルした後にアナリティック・ワークスペースを更新およびコミットすると、コンパイル済コードがアナリティック・ワークスペースに保存され、以降のセッションでそのプログラムを実行するために使用されます。したがって、プログラムをコンパイルした後には更新およびコミットを行う必要があります。これは、プログラムが多くのユーザーによって実行されるアプリケーションの一部である場合は特に重要です。コンパイル済のプログラムがアナリティック・ワークスペースに保存されないかぎり、そのプログラムは各ユーザー・セッションで個別に再コンパイルされます。

例 7-17 COMPILE コマンドの使用

myprog プログラムをコンパイルする COMPILE コマンドの例を次に示します。

```
COMPILE myprog
```

myprog プログラム内の LIMIT コマンドで、次に示すとおり、month ディメンションのスペルを間違って指定したと想定します。

```
LIMIT motnh TO LAST 6
```

COMPILE コマンドがこのコマンドを検出すると、次のメッセージが生成されます。

```
ERROR: (MXMSERR00) Analytic workspace object MOTNH does not exist.
In DEMO!MYPROG PROGRAM:
limit motnh to last 6
```

プログラムを編集してエラーを修正し、プログラムを再度コンパイルできます。

プログラムがコンパイル済かどうかの確認

次に示すとおり、OBJ ファンクションの ISCOMPILED オプションを使用すると、アナリティック・ワークスペースの特定のプログラムが最後に変更されてからコンパイルされているかどうかを判断できます。このファンクションは、ブール値を戻します。

```
SHOW OBJ(ISCOMPILED 'myprogram')
```

コンパイルを防止するプログラミング方法

アンパサンド置換を含むプログラム行はコンパイルされません。すべての構文エラーは、そのプログラムが実行されるまで検出されません。他の行が正常にコンパイル済であるプログラムは、コンパイル済プログラムとみなされます。

プログラムがオブジェクトを定義し、そのオブジェクトをプログラムで使用する場合、そのプログラムはコンパイルされません。そのオブジェクトはアナリティック・ワークスペースにまだ存在しないため、COMPILE では、そのオブジェクトへの参照はスペル間違いとして処理されます。

参照： アンパサンド置換については、7-8 ページの「アンパサンド置換によるテキストとしての引数の指定」を参照してください。

プログラムのテストおよびデバッグ

プログラムがコンパイルされた場合でも、プログラムを実行してそれをテストする必要があります。プログラムを実行すると、アンパサンド置換を使用したコマンドのエラー、ロジックのエラー、およびネストしたすべてのプログラムのエラーを検出する場合に有効です。

プログラムを実行してそれをテストするには、そのプログラムが処理するすべての典型的なデータ・セットであるテスト・データを使用します。エラー処理メカニズムを含む、プログラムのすべての機能をテストしたことを確認するために、異なるデータおよび応答を使用して、プログラムを数回実行します。次のテスト・データを使用します。

- 予想される範囲内のデータ
- 予想される範囲外のデータ
- プログラムの各セクションを実行するデータ

診断メッセージの生成

プログラムを実行するたびに、プログラムがそのコマンドを正しい順序で実行し、出力が適切であることを確認します。SHOW コマンドをプログラムに追加して診断メッセージまたは状態メッセージを生成すると、プログラムの実行を分析する場合に有効です。テストの完了後、SHOW コマンドを削除します。

プログラムまたはネストしたプログラムでエラーが検出されたか、またはエラーが存在すると思われる場合は、この項の後半で説明するデバッグ方法を使用してエラーを特定できます。

不適切なコード行の特定

BADLINE オプションを YES に設定すると、不適切なコード行が検出された場合に、エラー・メッセージとともに詳細情報が生成されます。エラーが発生すると、エラー・メッセージ、プログラム名、およびそのエラーを発生させたプログラム行が現行の送信ファイルに送信されます。

特定されたプログラムを編集してエラーを修正し、元のプログラムを実行できます。

例 7-18 BADLINE オプションの使用

test という名前の次の単純なプログラムでは、変数 myint1 が 0（ゼロ）によって除算されます。

```
DEFINE test PROGRAM
PROGRAM
VARIABLE myint1 INTEGER
VARIABLE myint2 INTEGER
myint1 = 0
myint2 = 250/myint1
END
```

DIVIDEBYZERO オプションが NO に設定されている場合にこのプログラムを実行すると、0（ゼロ）による除算は許可されないため、エラーが発生します。BADLINE が YES に設定されている場合は、次のメッセージが現行の送信ファイルに記録されます。

```
ERROR: (MXXEQ01) A division by zero was attempted. Set DIVIDEBYZERO to
YES if you want NA to be returned as the result of division by zero.
In DEMO!TEST PROGRAM:
myint2 = 250/myint1
```

デバッグ・ファイルへの出力の送信

プログラムにロジックのエラーが存在する場合、プログラムはエラー・メッセージを生成せずに実行されることがありますが、そのプログラムは不適切な一連のコマンドを実行するか、または不適切な結果を生成します。たとえば、IF コマンドでブール式を不適切に（EQ のかわりに NE を使用するなど）作成したと想定します。そのプログラムは指定したコマンドを実行しますが、コマンドは不適切な条件で実行されます。

プログラム・ロジックのエラーを検出するには、多くの場合、コマンドが実行される順序を確認する必要があります。これを実行できる 1 つの方法は、デバッグ・ファイルを作成し、そのファイルを調べてプログラムに問題がないかを診断することです。

デバッグ・ファイルの作成

デバッグ・ファイルを作成するには、DBGOUTFILE コマンドを使用します。DBGOUTFILE コマンドの構文は次のとおりです。

```
DBGOUTFILE {EOF|[APPEND] file-id [NOCACHE]}
```

このコマンドは、次の引数を持ちます。

- EOF キーワードを指定すると、現行のデバッグ・ファイルが閉じられ、デバッグ出力がそれ以上ファイルに送信されません。
- APPEND キーワードを指定すると、出力が既存のディスク・ファイルの終わりに追加されます。この引数を省略した場合、指定した名前のファイルが存在すると、新しい出力でファイルの現在の内容が置換されます。
- *file-id* 引数には、デバッグ出力を受信するファイルの名前を指定します。
- NOCACHE キーワードを指定すると、OLAP DML がコード行を実行するたびにデバッグ・ファイルに書込みを行います。このキーワードを使用しない場合は、テキストを保存して、それを定期的にファイルに書き込むことによって、ファイルの I/O アクティビティが削減されます。

DBGOUTFILE コマンドの詳細は、Oracle9*i* OLAP DML Reference ヘルプのこのコマンドの項を参照してください。

デバッグ・ファイルの内容の指定

DBGOUTFILE コマンドは、デバッグ用のファイルを作成するのみです。各プログラム行の実行時にそれをデバッグ・ファイルに送信することを指定するには、PRGTRACE オプションを YES に設定します。

デバッグ・ファイルにプログラムの行とともにプログラムの入力とエラー・メッセージの両方を記録するには、ECHOPROMPT オプションを YES に設定します。

ECHOPROMPT オプションおよび PRGTRACE オプションの構文の詳細は、Oracle9*i* OLAP DML Reference ヘルプの各オプションの項を参照してください。

例 7-19 デバッグ・ファイルの使用

次のコマンドを実行すると、カレント・ディレクトリ別名に debug.txt という有効なデバッグ・ファイルが作成されます。

```
prgtrace = yes
echoprompt = yes
dbgoutfile 'debug.txt'
```

これらのコマンドを実行した後、通常どおりプログラムを実行できます。デバッグ・ファイルを閉じるには、次のコマンドを実行します。

```
dbgoutfile eof
```

次のサンプル・プログラムでは、最初の LIMIT コマンドに構文エラーが存在します。

```
DEFINE ERROR_TRAP PROGRAM
PROGRAM
trap on traplabel
limit month to first badarg
limit product to first 3
limit district to first 3
report sales
traplabel:
signal errorname errortext
END
```

PRGTRACE と ECHOPROMPT の両方を YES に設定し、デバッグ出力を debug.txt というファイルに送信するように DBGOUTFILE を設定すると、error_trap プログラムの実行時に次のテキストが debug.txt ファイルに送信されます。

```
(PRG= ERROR_TRAP)
(PRG= ERROR_TRAP) trap on traplabel
(PRG= ERROR_TRAP)
(PRG: ERROR_TRAP) limit month to first badarg
ERROR: BADARG does not exist in any attached database.
(PRG= ERROR_TRAP) traplabel:
(PRG= ERROR_TRAP) signal errorname errortext
ERROR: BADARG does not exist in any attached database.
```

8

モデルの使用

この章では、モデルを使用してデータを計算する方法について説明します。この章では、次の項目について説明します。

- モデルを使用したデータの計算
- ネストしたモデル階層の作成
- 基本的なモデリング・コマンド
- モデルのコンパイル
- モデルの実行
- モデルのデバッグ
- 複数のシナリオのモデリング

モデルを使用したデータの計算

モデルは、結果を変数またはディメンション値に割り当てることができる、相互に関連した一連の方程式です。たとえば、財務モデルでは、次に示すとおり、値を gross.margin や net.income などの特定の明細項目に割り当てることができます。

```
gross.margin = revenue - cogs
```

= コマンドがデータをディメンション値に代入するか、またはその計算でディメンション値を参照する場合、ディメンションベースの方程式と呼ばれます。ディメンションベースの方程式は、ディメンション自体ではなく、ディメンションの値のみを参照します。したがって、モデルにディメンションベースの方程式が含まれている場合は、DIMENSION コマンドでモデルの先頭に各ディメンションの名前を指定する必要があります。

モデルにディメンションベースの方程式が含まれている場合は、そのモデルの実行時に**ソリューション変数**の名前を指定する必要があります。ソリューション変数は、データのソースであるとともに、モデルの方程式の割当てターゲットでもあります。ソリューション変数はディメンションベースの方程式で使用される入力データを保持し、計算結果が指定したソリューション変数値に格納されます。たとえば、line ディメンションに基づいた財務モデルを実行する場合、actual をソリューション変数として指定できます。

ディメンションベースの方程式は、財務モデリングにおける柔軟性を提供します。モデルを解決するまでモデリング変数を指定する必要がないため、actual 変数や budget 変数などの、line によってディメンション化された任意の変数で同じモデルを実行できます。

例 8-1 モデルの作成

損益計算書の明細項目を計算する income.calc というモデルを次のとおり定義すると想定します。

```
define income.calc model
ld Calculate line items in income statement
```

モデルを定義した後、MODEL コマンドまたは OLAP Worksheet エディタを使用してモデルの内容を指定できます。モデルは、DIMENSION コマンド、= コマンドおよびコメントを含むことができます。すべての DIMENSION コマンドは、最初の方程式の前に配置する必要があります。この例では、次のモデルに示す行を指定します。

```
DEFINE INCOME.CALC MODEL
LD Calculate line items in income statement
MODEL
DIMENSION line
net.income = opr.income - taxes
opr.income = gross.margin - (marketing + selling + r.d)
gross.margin = revenue - cogs
END
```

モデルの方程式を作成するときは、方程式を任意の順に配置できます。COMPILE コマンドを使用するか、またはモデルを実行してモデルをコンパイルするときに、モデルの方程式を解決する順序が決定されます。ある方程式の計算結果が別の方程式の入力として使用される場合、それらの方程式は必要な順に解決されます。

income.calc モデルを実行し、actual をソリューション変数として使用するには、次のコマンドを実行します。

```
income.calc actual
```

ソリューション変数にモデルの方程式の基礎となるディメンション以外のディメンションが含まれている場合は、これらの余分な各ディメンションの現行のステータス・リストに対して自動的にループが実行されます。たとえば、actual は、line のみでなく month および division によってもディメンション化されています。division が ALL に制限され、month が OCT96 ～ DEC96 に制限される場合、income.calc モデルは部門ごとにステータス内の前述の 3 か月に対して解決されます。

モデルでディメンション値を処理する方法

データをディメンション値に代入する = コマンドがモデルに含まれている場合、ディメンションは一時的にその値に制限されて計算が実行され、その後、そのディメンションの初期ステータスがリストアされます。

たとえば、モデルには次のコマンドが含まれる場合があります。

```
DIMENSION line
gross.margin = revenue - cogs
```

このモデルの実行時に actual をソリューション変数として指定すると、次のコードが構成および実行されます。

```
PUSH line
LIMIT line TO gross.margin
actual = actual(line revenue) - actual(line cogs)
POP line
```

この暗黙的な構成によって、単純なモデルの方程式で複雑な計算を実行できます。たとえば、明細項目データを line によってディメンション化された actual 変数に格納します。ただし、明細項目のディテール・データは detail.line という名前のディメンションを含む、detail.data という名前の変数に格納します。

アナリティック・ワークスペースに各詳細項目が関連する明細項目を指定する、line と detail.line の間のリレーションが含まれている場合、次のようなモデルの方程式を作成できます。

```
revenue = total(detail.data line)
expenses = total(detail.data line)
```

detail.line と line の間のリレーションは、ディテール・データを適切な明細項目に集計するために自動的に使用されます。モデルの実行時に構成されるコードによって、適切な合計が line ディメンションの各値に割り当てられることが保証されます。たとえば、revenue 項目の方程式の計算中、line は一時的に revenue に制限され、TOTAL ファンクションは line の revenue 値に対する明細項目の合計を戻します。

ネストしたモデル階層の作成

INCLUDE コマンドを使用すると、1 つのモデル内に別のモデルを含めることができます。モデルは、INCLUDE コマンドを 1 つのみ含むことができます。INCLUDE コマンドは、モデル内のすべての方程式の前に配置する必要があり、含めるモデル名を 1 つのみ指定できます。INCLUDE コマンドを含むモデルは、**親モデル**といいます。含まれているモデルは、**ベース・モデル**といいます。

ベース・モデル内で INCLUDE コマンドを使用すると、複数のモデルをネストできます。たとえば、モデル m1 はモデル m2 を含むことができ、モデル m2 はモデル m3 を含むことができます。ネストしたモデルは階層を構成します。この例では、m1 が階層の最上位に位置し、m3 がルートに位置します。

INCLUDE コマンドの使用

モデルに INCLUDE コマンドが含まれている場合、モデルは DIMENSION コマンドを含むことができません。親モデルは、含まれている階層のルート・モデル内の DIMENSION コマンドから自身のディメンション（存在する場合）を継承します。前述の例では、モデル m1 および m2 はどちらもモデル m3 内の DIMENSION コマンドから自身のディメンションを継承します。

INCLUDE コマンドを使用すると、モジュール・モデルを作成できます。複数のモデルに共通の特定の方程式が存在する場合、これらの方程式を別のモデルに配置し、そのモデルを必要に応じて他のモデルに含めることができます。

INCLUDE コマンドによって、what-if 分析も容易になります。実験モデルでは、ベース・モデルから方程式を取り出し、それらを選択的に新しい方程式で置換することができます。what-if 分析をサポートするには、モデルの方程式を使用して以前の方程式をマスクします。以前の方程式は、同じモデルのものである場合または含まれているモデルのものである場合があります。マスクした方程式は実行されません。

モデルを実行するか、または COMPILE コマンドを使用してモデルをコンパイルした後、MODEL.COMPRPT という OLAP DML プログラムを実行してコンパイル済モデルの構造に関するレポートを生成できます。マスクされた方程式を含むモデルをコンパイルした後に MODEL.COMPRPT を実行すると、マスクされた方程式はレポートに表示されません。

基本的なモデリング・コマンド

次の表に、モデルを定義および実行する場合に使用する一般的な OLAP DML コマンドを示します。

コマンド	説明
DEFINE	新しいモデルをアナリティック・ワークスペースに追加します。
MODEL	新しいモデルまたは既存のモデルの新しい内容を指定します。
DIMENSION	モデル内のディメンションベースの方程式で参照される 1 つ以上のディメンションを指定します。
INCLUDE	親モデルに含めるベース・モデルを指定します。
=	計算を行い、その結果をターゲットに代入します。ターゲットは、変数であるか、またはディメンション値で表すことができます。
COMPILE	モデルを実行せずにそのモデルをコンパイルし、コンパイル済コードをアナリティック・ワークスペースに保存します。新しいモデルまたは変更されたモデルをコンパイルせずに実行すると、そのモデルはその時点で自動的にコンパイルされます。

モデルの方程式の作成

モデルの方程式を作成する場合は、次の点に注意してください。

- 単一のディメンションベースの方程式内では、すべてのディメンション値が同じディメンションに含まれる必要があります。
- モデルの方程式ではアンパサンド置換を使用できません。

DIMENSION および INCLUDE コマンドの作成

DIMENSION コマンドおよび INCLUDE コマンドを作成する場合は、次の点に注意してください。

- すべての DIMENSION コマンドまたは INCLUDE コマンドは、モデル内の最初の方程式の前に配置する必要があります。
- DIMENSION コマンドでは、モデルの方程式の基礎となるすべてのディメンションの名前を指定する必要があります。次の例では、gross.margin、revenue および cogs は line ディメンションの値であるため、DIMENSION コマンドで line が指定されています。

```
DIMENSION line
gross.margin = revenue - cogs
```

- DIMENSION コマンドでは、ディメンション値を参照するファンクションの引数であるディメンションも指定する必要があります。次の例では、DIMENSION コマンドで month を指定する必要があります。

  ```
  DIMENSION line, month
  revenue = LAG(revenue, 1, month) * 1.05
  ```

- モデルに INCLUDE コマンドが含まれている場合、モデルは DIMENSION コマンドを含むことができません。含まれているモデル（階層内のルート・モデル）には、親モデルに必要な DIMENSION コマンドが含まれている必要があります。
- モデルの方程式が結果をディメンション値に割り当てる場合、DIMENSION コマンドで指定された他のすべての非ターゲット・ディメンションの値に対してループを実行するコードが構成されます。DIMENSION コマンドで最初に指定された非ターゲット・ディメンションは、最も遅く変化するディメンションとして処理されます。
- DIMENSION コマンドのディメンションおよびソリューション変数のディメンションを調整するための次のガイドラインに従うと、モデルを最も効率的に実行できます。
  - モデルのターゲット・ディメンションを DIMENSION コマンドでは最初のディメンションとして指定し、ソリューション変数の定義では最後のディメンションとして指定します。
  - DIMENSION コマンドでは、非ターゲット・ディメンションをソリューション変数の定義で指定される順序と逆の順序で指定します。これは、最も速く変化するディメンションおよび最も遅く変化するディメンションという点で、非ターゲット・ディメンションが、モデル内とソリューション変数内で同じ順序になることを意味します。
- ソリューション変数にモデルの方程式で使用または参照されていないディメンションが含まれている場合は、DIMENSION コマンドにそれらのディメンションを含めないでください。
- アナリティック・ワークスペースにディメンション値と同じ名前の変数が含まれている場合、または異なるディメンションに同じディメンション値が存在する場合は、モデルの方程式が不明瞭になる可能性があります。変数およびディメンション値はモデルの方程式で同様に使用できるため、名前は、変数の名前である場合またはアナリティック・ワークスペース内のいずれかのディメンションの値である場合があります。
- DIMENSION コマンドは、代入文（= コマンド）の各名前参照が変数またはディメンション値のどちらであるかを決定するために使用します。名前参照の解決方法の詳細は、8-6 ページの「モデルのコンパイル」を参照してください。

モデルのコンパイル

モデルのコマンドの作成後、COMPILE コマンドを使用してそのモデルをコンパイルできます。コンパイル中、COMPILE は書式エラーが存在するかどうかを確認します。そのため、COMPILE を使用すると、モデルを実行する前にコードをデバッグできます。モデルを実行

する前に COMPILE コマンドを使用しない場合、モデルは解決される前に自動的にコンパイルされます。

COMPILE コマンドを使用するか、またはモデルを実行してそのモデルをコンパイルすると、モデル・コンパイラは各方程式を調べて、割当てターゲットおよび各データ・ソースが変数またはディメンション値のどちらであるかを判断します。

各名前参照を解決するために、次の手順が実行されます。

1. DIMENSION コマンド内のディメンションが指定されている順に検索され、指定されたディメンションのディメンション値がその名前と一致するかどうかが確認されます。一致するディメンション値が検出されるとすぐに検索が完了します。
2. 指定されたディメンションの値がその名前と一致しない場合は、アタッチされたアナリティック・ワークスペース内の変数が検索され、一致するものが検出されます。

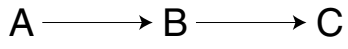
各名前参照を解決した後で、モデル・コンパイラはモデルの方程式間の依存関係を分析します。ある方程式の等号の右側にある式が別の方程式の割当てターゲットを参照している場合は、依存関係が存在します。方程式間の依存関係によって = コマンドが自身に間接的に依存している場合は、方程式間に循環依存が存在します。

モデル・コンパイラは、方程式をブロックに構成し、各ブロック内の方程式およびブロック自体を順序付けて依存関係を示します。コンパイラは、単純ブロック、ステップ・ブロックおよび連立ブロックという 3 つのタイプのソリューション・ブロックを生成できます。

単純ブロック

単純ブロックには、相互に非依存の方程式および非循環型の相互依存関係を持つ方程式が含まれます。

ブロックに値 A、B および C に対して解決される方程式が含まれている場合、非循環依存は次のとおり表すことができます。ここで、矢印は A が B に依存し、B が C に依存することを示しています。

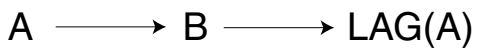


ステップ・ブロック

ステップ・ブロックには、一方向のディメンション依存である循環依存を持つ方程式が含まれます。**ディメンション依存**は、現行のディメンション値のデータが以前または将来のディメンション値のデータに依存する場合に発生します。一方向のディメンション依存とは、データが以前の値または将来の値のどちらかにのみ依存し、両方には依存しない場合です。

通常、ディメンション依存は時間ディメンションで発生します。たとえば、一般に、明細項目の値は以前の時間間隔の同じ明細項目または異なる明細項目の値に依存します。ブロックに値 A および B に対して解決される方程式が含まれている場合、一方向のディメンション

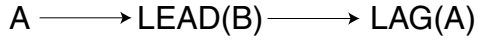
依存は次のとおり表すことができます。ここで、矢印は A が B に依存し、B が以前の時間間隔の A の値に依存することを示しています。



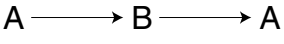
連立ブロック

連立ブロックには、一方向のディメンション依存以外の循環依存を持つ方程式が含まれます。循環依存は、双方向ディメンション依存である場合またはディメンション修飾子を含んでいない場合があります。

双方向ディメンション依存である循環依存の例は、次のとおり表すことができます。ここで、矢印は A が将来の期間の B の値に依存し、B が以前の期間の A の値に依存することを示しています。



ディメンション修飾子に依存しない循環依存の例は、次のとおり表すことができます。ここで、矢印は A が B に依存し、B が A に依存することを示しています。



モデルの実行

モデルを実行する場合は、次の点に注意してください。

- モデルを実行する前に、入力データがソリューション変数で使用可能である必要があります。たとえば、`actual` をソリューション変数として使用して `income.calc` モデル（この章の前述の項を参照）を実行する前に、`actual` の明細項目 `revenue`、`cogs`、`marketing`、`selling`、`r.d` および `taxes` に現行データが存在する必要があります。
- 連立方程式のブロックを含むモデルを実行する前に、連立ブロックの解決を制御するいくつかの OLAP DML オプションの値を確認または変更する場合があります。連立方程式の詳細は、8-9 ページの「連立方程式の解決」を参照してください。
- モデルにディメンションベースの方程式が含まれている場合、データのソースおよび方程式の結果の割当てターゲットの両方として機能する数値ソリューション変数を指定する必要があります。通常、ソリューション変数はモデルの方程式の基礎となるすべてのディメンションによってディメンション化され、余分なディメンションを含んでいる場合もあります。

- モデルを実行すると、ソリューション変数の余分な各ディメンションの現行のステータス・リストに含まれる値に対して自動的にループが実行されます。
- モデルの方程式の計算が以前の時間間隔のデータに基づいて行われる場合（LAG ファンクションを使用する場合など）、ソリューション変数にはその以前の期間のデータが含まれている必要があります。データが含まれていない場合、または時間ディメンションの最初の値がステータス内にある場合は、計算の結果が NA になります。

過去および将来の時間間隔のデータの使用

いくつかの OLAP DML ファンクションを使用すると、過去または将来の時間間隔のデータを簡単に使用できるようになります。たとえば、LAG ファンクションは指定した以前の時間間隔のデータを戻し、LEAD ファンクションは指定した将来の期間のデータを戻します。財務データの分析に有効な組込みファンクションの詳細は、Oracle9*i* OLAP DML リファレンス・ヘルプを参照してください。

計算で過去または将来のデータを使用するモデルを実行する場合は、ソリューション変数に必要な過去または将来のデータが含まれていることを確認する必要があります。たとえば、モデルには、先月の revenue 明細項目に基づいて今月の revenue 明細項目を概算する代入文（= コマンド）が含まれている場合があります。

```
DIMENSION line month
.
.
.
revenue = LAG(revenue, 1, month) * 1.05
```

モデルの実行時に month ディメンションが apr96 ～ jun96 に制限されている場合は、ソリューション変数に mar96 の revenue データが含まれていることを確認する必要があります。

モデルに LEAD ファンクションが含まれている場合は、ソリューション変数に必要な将来のデータが含まれている必要があります。たとえば、1996 年 4 ～ 6 月のデータを計算し、モデルが 1 か月先からデータを取得する場合は、モデルの実行時にソリューション変数に 1996 年 7 月のデータが含まれている必要があります。

連立方程式の解決

連立ブロックの方程式の解決には、反復方法が使用されます。各反復では、各方程式の値が計算され、新しい値が以前の反復の値と比較されます。比較が指定した許容範囲内である場合、方程式は解決に収束したとみなされます。比較が指定した限度を超える場合、方程式は発散したとみなされます。

ブロックのすべての方程式が収束する場合、ブロックは解決したとみなされます。いずれかの方程式が発散するか、または指定した回数の反復内で収束しなかった場合、ブロック（およびモデル）の解決は失敗し、エラーが発生します。

OLAP DML オプションを使用すると、連立方程式の解決を制御できます。たとえば、使用する解決方法、収束および発散のテストに使用する要素、実行する反復の最大回数、および = コマンドが発散するか、または収束しなかった場合に行う処置を指定できます。

モデルのデバッグ

OLAP DML では、モデルをデバッグする場合に有効な様々なツールが提供されています。次の表に、これらのツールを示します。

ツール	用途
MODTRACE	モデルの実行中にモデルの各行を現行の送信ファイルに送信するかどうかを制御するオプション。MODTRACE を YES に設定すると、モデルの行が表示され、方程式の解決順序を確認できます。
DBGOUTFILE	MODTRACE 出力の送信先のデバッグ・ファイルを作成するコマンド。このコマンドの詳細は、7-28 ページの「デバッグ・ファイルへの出力の送信」を参照してください。
MODEL.COMPRPT	コンパイル済モデルの構造に関するレポートを生成するプログラム。レポートには、モデルの方程式がどのようにブロックにグループ化されたかどうかが示されます。
MODEL.DEPRPT	モデルの方程式の依存関係に関するレポートを生成するプログラム。レポートでは、各方程式の割当てターゲットおよびデータ・ソースが示され、その方程式の依存関係のディメンションが特定されます。
MODEL.XEQRPT	モデルの解決ステータスに関するレポートを生成するプログラム。モデルに連立方程式が含まれる場合、レポートでは連立方程式の解決を制御するオプションの値が特定されます。
INFO	コンパイルまたは実行したモデルに関する特定の情報を取得できるファンクション。

複数のシナリオのモデリング

1 つの月および部門について単一の数値セットを計算するのではなく、複数の数値セットをそれぞれ異なる想定に基づいて計算する場合があります。

様々な入力値セットに基づいて予測または予算額を計算および格納する**シナリオ・モデル**を定義できます。たとえば、楽観値、悲観値および最良予測値に基づいて収益を計算できます。

シナリオ・モデルの作成

シナリオ・モデルを作成するには、次の手順を実行します。

1. シナリオ・ディメンションを定義します。

2. シナリオ・ディメンションによってディメンション化されたソリューション変数を定義します。
3. ソリューション変数に入力データを入力します。
4. 入力データに基づいて結果を計算するモデルを作成します。

たとえば、部門ごとに楽観収入額、悲観収入額および最良予測収入額に基づいて収益高を計算すると想定します。このシナリオ・モデルを作成する手順については、次の例を参照してください。

例 8-2 シナリオ・モデルの作成

シナリオ・ディメンション scenario をコールし、それに計算するシナリオを表す値を与えることができます。

次のコマンドを実行すると、scenario に値 optimistic、pessimistic および bestguess が指定されます。

```
DEFINE scenario DIMENSION TEXT
LD Names of scenarios
MAINTAIN scenario ADD optimistic pessimistic bestguess
```

次のコマンドを実行すると、scenario ディメンションに加えて他の 3 つのディメンション（month、line および division）によってディメンション化された plan という名前の変数が作成されます。

```
DEFINE plan DECIMAL <month line division scenario>
LD Scenarios for financials
```

この例では、収入や売上原価などの入力データを plan 変数に入力する必要があります。

最良予測データの場合、budget 変数のデータを使用する場合があります。次に示すとおり、line ディメンションを入力明細項目に制限し、budget データを plan 変数にコピーします。

```
LIMIT scenario TO 'BESTGUESS'
LIMIT line TO 'REVENUE' 'COGS' 'MARKETING' 'SELLING' 'R.D'
plan = budget
```

最良予測データに基づいて楽観データおよび悲観データを設定する場合があります。たとえば、楽観データを最良予測データより 15% 高く設定し、悲観データを最良予測データより 12% 低く設定する場合があります。line を入力明細項目に制限したままで、次のコマンドを実行します。

```
plan(scenario 'OPTIMISTIC') = 1.15 * plan(scenario 'BESTGUESS')
plan(scenario 'PESSIMISTIC') = .88 * plan(scenario 'BESTGUESS')
```

最後に、入力データに基づいて結果を計算するモデルを作成します。モデルには、budget.calc モデル（この章の前述の項を参照）の計算と非常に類似した計算が含まれる場合があります。

各シナリオには、同じ方程式を使用することも、異なる方程式を使用することもできます。たとえば、売上原価を計算し、各シナリオの計算で異なる定数要素を使用する場合があります。シナリオごとに異なる定数要素を使用するには、scenario によってディメンション化された変数を定義し、その変数に適切な値を指定します。変数の名前が cogsval である場合、モデルには cogs 明細項目を計算するための次の方程式が含まれていることがあります。

```
cogs = cogsval * revenue
```

scenario によってディメンション化された変数を使用すると、シナリオ・モデルの柔軟性を大幅に向上できます。

同様に、部門ごとに異なる定数要素を使用する場合があります。division によってディメンション化された変数を定義して、各部門の値を保持できます。たとえば、部門によって人件費が異なる場合、cogsval を division および scenario によってディメンション化できます。

モデルの実行時に、plan をソリューション変数として指定します。たとえば、scenario.calc というモデルの場合、次のコマンドでモデルを解決します。

```
scenario.calc plan
```

plan の各ディメンションの現行のステータス・リストに対して自動的にループが実行されます。したがって、scenario.calc モデルの実行時に scenario ディメンションが ALL に制限される場合、optimistic、pessimistic および bestguess という 3 つのすべてのシナリオに対してモデルが解決されます。

9

データのアロケーション

この章では、ALLOCATE コマンドを使用してソースからターゲット変数にデータを割り当てる方法について説明します。この章では、次の項目について説明します。

- アロケーションの概要
- アロケーションの準備
- アロケーション用の集計マップの作成
- アロケーション演算子および引数の使用

アロケーションの概要

Oracle OLAP の ALLOCATE コマンドは、データをソース・オブジェクトからターゲットのセルに分散します。ターゲットは、1 つ以上の階層ディメンションによってディメンション化されている変数です。ソース・データは、ターゲット・セルを指定するディメンション値より上位のレベルの階層ディメンションに含まれるディメンション値によって指定されます。

ALLOCATE では、集計マップを使用して、アロケーションで使用する階層のディメンションと値、ディメンションに対して使用する操作方法、およびアロケーションの他の動作を指定します。

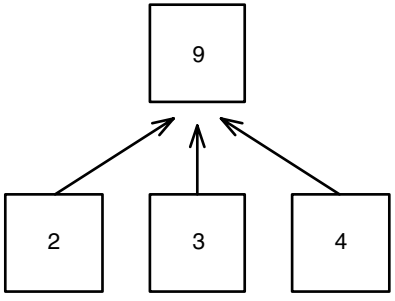
一部のアロケーション操作は、既存のデータに基づきます。そのようなデータを含むオブジェクトを、アロケーションの基礎オブジェクトと呼びます。これらの操作では、ALLOCATE を実行すると、基礎オブジェクトの値に基づいて、ソースからデータが分散されます。

予測および予算システムでは、通常、従業員の現在の給与および業績評価に基づいた額をボーナス・プールから自動的に分配する操作などで、アロケーションを使用します。

アロケーションは、AGGREGATE コマンドを使用して実行する集計の逆の操作です。集計では、階層の下位レベルのデータを上位レベルのデータに結合します。アロケーションでは、階層の上位レベルのデータを下位レベルに分散します。

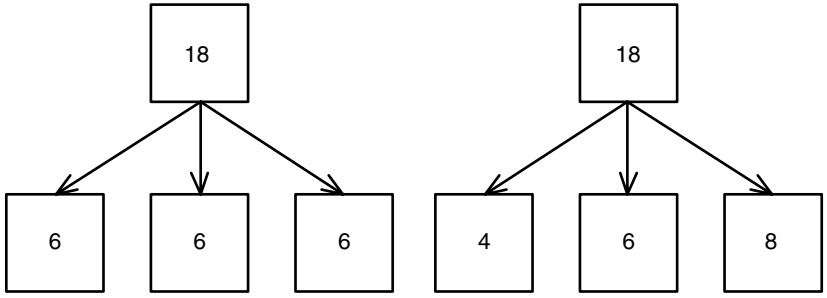
ALLOCATE コマンドは、AGGREGATE コマンドと逆の操作を行います。図 9-1 に、単純な階層での集計を示します。集計の SUM 操作では、ディテール・レベルの値 2、3 および 4 を足して集計レベルの値 9 を導出します。

図 9-1 単純な階層での集計



アロケーションの例は、取得した集計値 9 を 2 倍して 18 とした後、結果をディテール・レベルに割り当てる場合です。アロケーションの基礎として、ディテール・レベルのセル内の以前の値を使用します。図 9-2 の左の階層に、ソース値を均等に分散する EVEN アロケーション操作の結果を示します。ディテール・レベルの各セルは値 6 を受け取ります。右の階層に、ソース値を比例して分散する PROPORTIONAL アロケーション操作の結果を示します。ディテール・レベルに割り当てられる値は 4、6 および 8 です。

図 9-2 単純な階層でのアロケーション



アロケーション操作方法では、単純なアロケーション（ターゲット変数のセルへのソース・データのコピーなど）から非常に複雑なアロケーション（フォーミュラであるソースからのデータの比例分散など）まで、様々な操作を実行できます。また、他のフォーミュラに基づいた量の分散、複数のターゲット変数の使用、およびディメンションごとに異なるアロケーション方法を指定する集計マップの使用も可能です。

Oracle OLAP のアロケーション・システムは非常に柔軟であり、次のような多くの機能を使用できます。

- ソース、基礎およびターゲット・オブジェクトには、同じ変数または異なるオブジェクトのどちらも使用できます。
- フォーミュラをソース・オブジェクトおよび基礎オブジェクトとすることができるため、既存のデータに対して計算を実行し、その結果をアロケーションのソースまたは基礎として使用できます。
- ディメンションに対してアロケーションの操作方法を指定することができます。単純なものから非常に複雑なものまで、様々な操作を実行できます。アロケーション・システムの操作は、集計操作の逆の操作です。
- 割り当てられた値は、ターゲット・セルの既存の値に対して加算または置換のどちらを行うかを指定できます。
- ターゲット・セルに結果を割り当てる前に、割り当てられた値に加算または乗算する量を指定できます。
- ディメンション値のターゲット・セルのデータがアロケーションによって変更されないように、ディメンション階層の個々の値をロックできます。ディメンション値をロックすると、アロケーション・システムがソース・データを正規化し、アロケーションの前にソースからロックされたデータが除外されます。ソース・データを正規化しないことも選択できます。
- 特定の操作では、最小値、最大値、下限値または上限値を指定できます。
- 割り当てられたデータを 2 つ目の変数にコピーして、複数アロケーションのターゲットであるセルへの個々のアロケーションを記録できます。

- 基礎に NULL 値が含まれる場合のアロケーションの処理方法を指定できます。
- 異なる ALLOCATE コマンドで同じ集計マップを使用して、異なるソース、基礎およびターゲット・オブジェクトで同じディメンション階層値、操作および引数のセットを使用できます。

OLAP DML では、ALLOCATE コマンドとともにアロケーションをサポートする他のコマンドを使用できます。表 9-1 に、それらのコマンドを示します。

表 9-1

コマンド	説明
AGGMAPINFO コマンド	集計マップ・オブジェクトの内容に関する情報（集計またはアロケーション用のコマンドが含まれているかどうかを示す集計マップのタイプなど）を戻します。
ALLOCATE コマンド	ソース・オブジェクトからターゲット変数にデータを割り当てます。
ALLOCERRLOGFORMAT コマンド	ALLOCATE コマンドの ERRORLOG 引数で指定するエラー・ログの内容および書式を決定します。
ALLOCERRLOGHEADER コマンド	エラー・ログの列ヘッダーを決定します。
ALLOCMAP コマンド	アロケーションを行うディメンション階層のパス、操作の方法およびアロケーションの他の動作を指定する内容を aggmap オブジェクトに追加します。オブジェクトを ALLOCMAP タイプの集計マップとしてマークします。
POUTFILEUNIT オプション	ALLOCATE コマンドの処理過程に関する情報を受け取る位置を指定します。

この章の後半では、データのアロケーションで使用するオブジェクト、アロケーション操作のタイプ、および様々なアロケーションの例について説明します。

アロケーションの準備

データのアロケーションを準備するには、割り当てるデータ、アロケーションの基礎となるデータおよび割り当てられたデータを割り当てる変数セルを決定します。データを含むオブジェクトまたはデータを生成するオブジェクトを指定または作成します。

ターゲットは、1 つ以上の階層ディメンションによってディメンション化されている変数である必要があります。ソースは変数またはフォーミュラで、基礎はフォーミュラ、リレーションまたは変数です。ソース、基礎およびターゲットのすべてを同じ変数にすることができます。

アロケーションのコピーを保持するためにターゲット・ログ・オブジェクトを使用するかどうか、およびエラーを記録するファイルを指定するかどうかを決定します。ターゲット・ログには変数を使用し、エラー・ログにはファイル・ユニットを指定します。

アロケーションに使用する集計マップを作成します。集計マップの内容によって、アロケーションで使用するディメンションとディメンション階層の値、操作の方法およびアロケーションの他の動作を指定します。

リレーション・オブジェクトを使用したアロケーションで使用するディメンション階層の値を指定します。アロケーションで使用する階層の親子リレーションを指定するリレーションが存在しない場合、セルフ・リレーション（同じディメンションによってディメンション化されているディメンション上のリレーション）を作成します。リレーションに割り当てる親子関係によって、アロケーションのパスを指定します。

アロケーション用の集計マップの作成

アロケーション用の集計マップには、ディメンション階層内のアロケーションのパスを定義するリレーション、アロケーション操作の方法、およびアロケーションの他の動作を指定するコマンドが含まれます。集計マップを作成するには、DEFINE コマンドを使用して aggmap オブジェクトを定義するか、または CONSIDER コマンドを使用して既存の集計マップを指定します。次に、ALLOCMAP コマンドを使用して集計マップにコマンドを追加し、それを ALLOCMAP タイプの集計マップとしてマークします。

ALLOCMAP 集計マップには、次のコマンドを追加できます。

- 1 つ以上の RELATION コマンド
- CHILDLOCK コマンド
- DEADLOCK コマンド
- DIMENSION コマンド
- ERRORLOG コマンド
- ERRORMASK コマンド
- SOURCEVAL コマンド

RELATION コマンドでは、アロケーションで使用するディメンション階層の親子関係を示すセルフ・リレーションを指定します。アロケーションで使用するそれぞれのディメンションに対して、別々の RELATION コマンドを使用します。集計マップ内の RELATION コマンドの順序によって、アロケーションの順序が決まります。

RELATION コマンドでは、階層におけるアロケーション方法を決定する演算子を指定します。また、RELATION コマンドの引数を使用してアロケーションの他の動作を指定できます。たとえば、ARGS MIN *minval* 引数を使用すると、アロケーションで割り当てられた値が最小値より小さい場合にターゲット・セルに割り当てる値を指定できます。ARGS ADD 引数を使用すると、現在のデータを割り当てられたデータに置換するかわりに、アロケーション

で割り当てられたデータをそのセルの現在のデータに加算した後でターゲット・セルに結果を割り当てるように指定できます。

CHILDLOCK コマンドでは、集計マップの RELATION コマンドによってディメンション階層の親と子要素の両方にロックが指定されたかどうかを、ALLOCATE コマンドで判断するかどうかを指定します。

DEADLOCK コマンドでは、ALLOCATE コマンドでデッドロックが発生した場合にアロケーションを続行するかどうかを指定します。デッドロックは、ターゲット・セルがロックされているか、または一部の操作でターゲット・セルの基礎値が NA であるために、アロケーションで値を分散できない場合に発生します。

DIMENSION コマンドでは、ターゲット変数、およびソース・オブジェクトまたは基礎オブジェクトによって共有されていないディメンションのステータスとして設定する単一の値を指定します。

ERRORLOG コマンドでは、ALLOCATE コマンドで指定されたエラー・ログで許可するエラーの数、および最大数のエラーが発生した場合にアロケーションを続行するかどうかを指定します。

ERRORMASK コマンドでは、エラー・ログから除外するエラー条件を指定します。

SOURCEVAL コマンドでは、ALLOCATE で、アロケーション後にソース・データ値を変更するかどうかを指定します。ソースが変数の場合のみ、SOURCEVAL を使用します。SOURCEVAL では、アロケーション後のソースの値を 0（ゼロ）、NA または現行の値（アロケーション前のそのセルの値）に指定できます。再帰的アロケーションでは、ALLOCATE を実行すると、SOURCEVAL で指定された値がアロケーションでソースとして使用された任意のセルに適用されます。

アロケーション演算子および引数の使用

アロケーション用の集計マップで RELATION コマンドに OPERATOR 引数を使用して、アロケーションの操作方法を指定する必要があります。操作方法のカテゴリは次のとおりです。

- コピー演算子
- 均等分散演算子
- 比例分散演算子

コピー演算子は、COPY、HCOPY、MIN、MAX、FIRST、LAST、HFIRST および HLAST です。均等分散演算子は、EVEN および HEVEN です。比例分散演算子は、PROPORTIONAL です。

先頭に H が付いた演算子は、リレーション・オブジェクトで指定された階層関係に基づいてデータを割り当てる階層演算子です。COPY や EVEN などの非階層演算子は、ターゲット・セルの基礎値が NA の場合、そのセルに値を割り当てません。

階層演算子は基礎値を使用しません。かわりに、ターゲット・セルの既存の値が NA の場合でも、リレーションによって指定されたディメンション階層のすべての値にデータを割り当

てます。基礎値が NA であるセルに値を割り当てることによってディテール・レベルのデータが大幅に増加する場合があるため、階層演算子は注意して使用する必要があります。

RELATION コマンドを ARGS キーワードとともに使用して、アロケーションに影響を及ぼす引数を指定できます。引数を使用すると、次の内容を指定できます。

- アロケーションでターゲット・セルに割り当てる最小値または最大値。
- 下限値または上限値。割り当てられた値が下限値より小さいか、または上限値より大きい場合、ALLOCATE ではターゲット・セルに NA が割り当てられます。
- 割り当てられた値を既存の値に加算するか、または既存の値を割り当てられた値と置換するか。
- 値セット・オブジェクトのディメンション値によって指定された、ターゲット変数のセルのロック。PROTECT 引数を指定すると、セルの既存の値が保護され、セルがアロケーションのターゲットになることが回避されます。また、ロックされたセルがアロケーションのソースになることができるかどうかを指定できます。たとえば、値セットがディメンション階層の中間レベルに位置するディメンション値を指定するときに、WRITE キーワードを使用する場合、ALLOCATE は階層に割り当てるソースとしてロックされた値を使用します。READWRITE キーワードを使用する場合、ALLOCATE は階層のそのブランチより下位にアロケーションを続行しません。また、ソース値を正規化するかどうかも指定できます。ソース値を正規化すると、アロケーションの前にソースからロックされた値が除外されます。
- 加重オブジェクト。加重オブジェクトは、ターゲット・セルに結果の値を割り当てる前に、ALLOCATE が割り当てられた値に加算または乗算する値を指定します。また、加重要素を適用する前に、NA 値に値を入力するかどうかも指定できます。

HEVEN 演算子、MAX 演算子および ADD 引数の使用

HEVEN 演算子は、基礎値に関係なく、ソース・データをターゲット・セルに均等に割り当てます。MAX 演算子は、最大の基礎値に対応するターゲット・セルにソース値を割り当てます。例 9-1 に、多次元アロケーションでこれらの演算子および ADD 引数を使用する方法を示します。アロケーション・パスは、ディメンション階層で大きい値から小さい値へ直接連結され、中間階層の値へのアロケーションは行われません。

fcstunits 変数は、階層ディメンション time、geog および product によってディメンション化されています。ディメンションは、1 つの製品、一部の都市および地域、2002 年および 2002 年の 4 か月に制限されています。

下位の階層ディメンション値である都市および月によってディメンション化されている fcstunits のセルには、値が割り当てられます。値は、その月にその都市に出荷される製品数の予測です。より上位の階層ディメンション値である YEAR02 および地域の値によってディメンション化されているセルの値は、都市および月に割り当てる追加製品数です。

fcstunits 変数のレポートを生成すると、次の値が表示されます。

```
PRODUCT: SHORTS - BOYS
                ----------------FCSTUNITS----------------
                -----------------TIME-------------------
GEOG              YEAR02     JUN02      JUL02      AUG02
--------------- ---------- ---------- ---------- ----------
EAST                   755         NA         NA         NA
WEST                   515         NA         NA         NA
CENTRAL                625         NA         NA         NA
BOSTON                  NA      5,760      5,690      4,750
ATLANTA                 NA      7,600      8,520      7,300
CHICAGO                 NA      4,660      4,840      5,120
DALLAS                  NA      8,380      9,380      8,150
DENVER                  NA      5,400      6,080      5,170
SEATTLE                 NA      7,210      7,490      7,310
```

geogcityreg リレーションは、都市の値を地域に関連付けます。timemonthyear リレーションは、月の値を年に関連付けます。リレーションのレポートを生成すると、次の値が表示されます。

```
GEOG      GEOGCITYREG
--------- -----------
EAST      NA
WEST      NA
CENTRAL   NA
BOSTON    EAST
ATLANTA   EAST
CHICAGO   CENTRAL
DALLAS    CENTRAL
DENVER    WEST
SEATTLE   WEST

TIME    TIMEMONTHYEAR
------  -------------
YEAR02  NA
JUN02   YEAR02
JUL02   YEAR02
AUG02   YEAR02
```

xunitsalloc 集計マップの最初の RELATION コマンドでは、geogcityreg リレーションによって指定された geog ディメンション階層に最初のアロケーションを実行することが指定されます。アロケーションによって、YEAR02 および地域の値によってディメンション化されているセルの値が均等に分割され、結果が地域の子に割り当てられます。除算の余りが存在する場合、REMOPERATOR LAST キーワードによって、最後のセルに割り当てられます。

xunitsalloc の SOURCEVAL コマンドでは、ALLOCATE でアロケーションのソース値を含むセルに値 0（ゼロ）を割り当てることが指定されるため、最初のアロケーションで地域に割り当てられた値は、ALLOCATE コマンドの完了後、変数のレポートに表示されません。YEAR02 の都市に割り当てられる地域データは、BOSTON 377、ATLANTA 378、CHICAGO 312、DALLAS 313、DENVER 257 および SEATTLE 258 です。

xunitsalloc の 2 番目の RELATION コマンドでは、timemonthyear リレーションによって指定された time ディメンション階層に 2 番目のアロケーションを実行することが指定されます。アロケーションのソース値は、YEAR02 および都市の値によってディメンション化されているセルの値です。アロケーションによって、その都市で最大値が含まれる月に各ソース値が割り当てられます。

例の ALLOCATE コマンドでは、fcstunits 変数のみが指定されています。そのため、その変数はアロケーションのソース、基礎およびターゲットです。また、このコマンドでは、アロケーションで xunitsalloc 集計マップを使用することも指定されます。

例 9-1 HEVEN および MAX 演算子を使用した多次元アロケーション

```
LIMIT product TO 'SHORTS - BOYS'
LIMIT geog TO 'EAST' 'WEST' 'CENTRAL' -
  'BOSTON' 'ATLANTA' 'CHICAGO' 'DALLAS' 'DENVER' 'SEATTLE'
LIMIT time TO 'YEAR02' 'JUN02' TO 'AUG02'
DEFINE xunitsalloc AGGMAP
ALLOCMAP JOINLINES( -
  'RELATION geogstcity OPERATOR HEVEN REMOPERATOR LAST' -
  'RELATION timemonthyear OPERATOR MAX ARGS ADD' -
  'SOURCEVAL ZERO')
ALLOCATE fcstunits USING xunitsalloc
REPORT fcstunits
```

アロケーションの後、fcstunits 変数の REPORT を実行すると、次のレポートが生成されます。

```
PRODUCT: SHORTS - BOYS
                ----------------FCSTUNITS-----------------
                -----------------TIME-------------------
GEOG              YEAR02     JUN02      JUL02      AUG02
--------------- ---------- ---------- ---------- ----------
EAST                     0         NA         NA         NA
WEST                     0         NA         NA         NA
CENTRAL                  0         NA         NA         NA
BOSTON                   0      6,137      5,690      4,750
ATLANTA                  0      7,600      8,898      7,300
CHICAGO                  0      4,660      4,840      5,432
DALLAS                   0      8,380      9,693      8,150
DENVER                   0      5,400      6,337      5,170
SEATTLE                  0      7,210      7,748      7,310
```

COPY 演算子および PROTECT 引数の使用

例 9-2 に、ディメンション階層の親によって指定されたソース・データの再帰的コピーを示します。このデータはその親の子に割り当てられ、割り当てられたデータはそれらの子の子へのアロケーションのソースとなります。また、この例に、異なるソース・データを 1 つの子およびその子にのみコピーする 2 番目のアロケーションも示します。

unitcost 変数は、time および prodid によってディメンション化されています。prodid ディメンションは、値として製品の識別番号を持つ NUMBER ディメンションです。最初の LIMIT コマンドを実行すると、prodid ディメンションのステータスが 1 つの値に設定されます。次の LIMIT コマンドを実行すると、time ディメンションのステータスが 2002 年、2002 年の最初の 2 つの四半期および 2002 年の最初の 6 か月に設定されます。

製品の unitcost の YEAR02 セルにソース値を割り当てます。unitcost のレポートを生成すると、次の値が表示されます。

```
               -UNITCOST-
               --PRODID--
TIME             45285
-------------- ----------
YEAR02              34.25
Q1.02                 NA
Q2.02                 NA
JAN02                 NA
FEB02                 NA
MAR02                 NA
APR02                 NA
MAY02                 NA
JUN02                 NA
```

例 9-2 では、costalloc 集計マップが定義され、ALLOCMAP コマンドを使用して内容がそのマップに追加されます。RELATION コマンドを実行すると、アロケーションのパスとして timeparent リレーション、メソッドとして HCOPY 演算子が指定されます。timeparent リレーションは、time ディメンション階層の子をその親へ関連付けます。

ALLOCATE コマンドを実行すると、アロケーションのソースおよびターゲットとして unitcost 変数が使用されます。メソッドが HCOPY であるため、アロケーションで基礎オブジェクトは使用されません。

最初のアロケーションの後、unitcost のレポートを生成すると、次の値が表示されます。

```
               -UNITCOST-
               --PRODID--
TIME             45285
-------------- ----------
YEAR02              34.25
Q1.02               34.25
Q2.02               34.25
JAN02               34.25
```

```
FEB02               34.25
MAR02               34.25
APR02               34.25
MAY02               34.25
JUN02               34.25
```

例では、次に YEAR02 のソース値が変更されます。値セットが定義され、その値が Q1.02 に制限されます。

2 番目の ALLOCMAP コマンドを実行すると、集計マップの内容が変更されます。RELATION コマンドに同じリレーションおよび COPY 演算子が指定され、また、PROTECT 引数も指定されます。SOURCEVAL コマンドでは、データのアロケーション後、ソース・データを含むセルに値 0（ゼロ）を割り当てることが指定されます。

2 番目のアロケーションによってセル YEAR02 から値がコピーされますが、子 Q1.02 およびその子がロックされるため、子 Q2.02 およびその子のみが割り当てられた値を受け取ります。

2 番目のアロケーションの後、unitcost のレポートを生成すると、次の値が表示されます。

```
              -UNITCOST-
              --PRODID--
TIME            45285
--------------- ----------
YEAR02               0.00
Q1.02               34.25
Q2.02               35.00
JAN02               34.25
FEB02               34.25
MAR02               34.25
APR02               35.00
MAY02               35.00
JUN02               35.00
```

例 9-2 COPY 演算子および PROTECT 引数の使用

```
LIMIT prodid TO 45285
LIMIT time TO 'YEAR02' 'Q1.02' 'Q2.02' 'JAN02' TO 'JUN02'

unitcost(time 'YEAR02' prodid 45285) = 34.25

DEFINE costalloc AGGMAP
ALLOCMAP 'RELATION timeparent OPERATOR HCOPY'
ALLOCATE unitcost USING costalloc

unitcost(time 'YEAR02' prodid 45285) = 35.00

DEFINE lvset VALUESET time
```

```
LIMIT lvset TO 'Q1.02'

CONSIDER costalloc
ALLOCMAP JOINLINES( -
  'RELATION timeparent OPERATOR COPY ARGS PROTECT NONORMALIZE lvset' -
  'SOURCEVAL ZERO')
ALLOCATE unitcost USING costalloc
```

HFIRST および HLAST 演算子の使用

HFIRST 演算子および HLAST 演算子は、基礎値に関係なく親の最初または最後の子にデータを割り当てる場合に使用します。例 9-3 では、現金繰越残高および現金繰越データが actual 変数から budget 変数に割り当てられ、次に親の time 値によって指定された budget セルから子の time 値によって指定された budget セルにデータが割り当てられます。

actual および budget 変数は、time、line および product ディメンションによってディメンション化されています。timeparent リレーションは、time ディメンションの子の値をその親へ関連付けます。

最初の一連の LIMIT コマンドを実行すると、time および line ディメンションのステータスが設定され、product ディメンションが 1 つの値に制限されます。そのディメンション・ステータスでの actual 変数のレポートを生成すると、次の値が表示されます。

```
PRODUCT: DRESSES - WOMEN
          ----------------------ACTUAL-------------------------
          ------------------------TIME-------------------------
LINE        Q4.01      JAN02      FEB02      MAR02      Q1.02
--------- ---------- ---------- ---------- ---------- ----------
CASH B/F    1,000.00         NA         NA         NA         NA
CASH MVT      500.00         NA         NA         NA         NA
CASH C/F    1,500.00         NA         NA         NA         NA
```

アロケーションのソース・データは、actual 変数の現金繰越行から budget 変数の現金繰越残高行に割り当てられます。次の LIMIT コマンドを実行すると、line ディメンションが CASH B/F に制限され、アロケーションがその値に制限されます。例 9-3 では、次に集計マップが定義され、ALLOCMAP コマンドを使用して内容がそのマップに追加されます。その内容は、HFIRST 演算子を指定する 1 つの RELATION コマンドです。ALLOCATE コマンドを実行すると、親 Q1.02 からその最初の子である JAN02 にデータが割り当てられます。

四半期の終わりまでに現金繰越高が 50% 増加することを予測し、この例では、actual 変数の Q4.01 現金繰越行の値が 1.5 倍され、結果が budget 変数の Q1.02 現金繰越行に割り当てられます。CONSIDER および ALLOCMAP コマンドを実行すると集計マップの内容が変更され、RELATION コマンドに HLAST 演算子が指定されます。

line ディメンションを現金繰越に制限します。次に ALLOCATE コマンドを実行すると、親 Q1.02 からその最後の子である MAR02 にデータが割り当てられます。最後に、LIMIT コマ

ンドを実行して line ディメンションのステータスをリセットします。アロケーションの後、budget 変数のレポートを生成すると、次の値が表示されます。

```
PRODUCT: DRESSES - WOMEN
          ------------------------BUDGET------------------------
          ------------------------TIME--------------------------
LINE        Q4.01      JAN02      FEB02      MAR02      Q1.02
--------- ---------- ---------- ---------- ---------- ----------
CASH B/F          NA   1,500.00         NA         NA   1,500.00
CASH MVT          NA         NA         NA         NA         NA
CASH C/F          NA         NA         NA   2,250.00   2,250.00
```

例 9-3 親の最初または最後の子へのデータのアロケーション

```
LIMIT time TO 'Q4.01' 'JAN02' TO 'MAR02' 'Q1.02'
LIMIT line TO 'CASH B/F' 'CASH MVT' 'CASH C/F'
LIMIT product TO 'DRESSES - WOMEN'

" Assign the value of actual Q4.01 CASH C/F to budget Q1.02 CASH B/F
budget(time 'Q1.02' line 'CASH B/F') = actual(time 'Q4.01' line 'CASH C/F')

LIMIT line TO 'CASH B/F'

DEFINE qtomalloc AGGMAP
ALLOCMAP 'RELATION timeparent OPERATOR HFIRST'

" Allocate the Q1.02 value to the first month of the quarter
ALLOCATE budget USING qtomalloc

" Forecast a 50% increase in cash forward by the end of the quarter
budget(time 'Q1.02' line 'CASH C/F') = actual(time 'Q4.01' line 'CASH C/F') * 1.5

CONSIDER qtomalloc
ALLOCMAP 'RELATION timeparent OPERATOR HLAST'

LIMIT line TO 'CASH C/F'

" Allocate the Q1.02 value to the last month of the quarter
ALLOCATE budget USING qtomalloc

LIMIT line TO 'CASH B/F' 'CASH MVT' 'CASH C/F'
```

PROPORTIONAL 演算子の使用

PROPORTIONAL 演算子は、基礎オブジェクトの値に比例してターゲット・セルにソース・データを割り当てます。例 9-4 に、time ディメンション階層に再帰的にデータを割り当てる 2 つの比例アロケーションを示します。

actual および budget 変数は、time、line および product ディメンションによってディメンション化されています。timeparent リレーションは、time ディメンションの子の値をその親へ関連付けます。

最初のアロケーションでは、YEAR02 からその年の四半期、月の順に、予測収入値を割り当てます。アロケーションは、前年の同じ時期の収入に基づきます。次に、2002 年の第 1 四半期の実際の値を budget 変数のセルに割り当てます。2 番目のアロケーションでは、第 1 四半期の budget セルおよびその子をロックし、ロックされた四半期の値をソースから除外してソース値を正規化します。その後、残りの値を他の四半期およびその子に割り当てます。

最初の一連の LIMIT コマンドを実行すると、line および product ディメンションのステータスがそれぞれ 1 つの値に制限され、time ディメンションが 2002 年の年、四半期および月の値に制限されます。

2002 年の budget 変数の値は、2001 年の actual 変数からコピーします。例には、その操作を示しません。その製品の 2002 年の予測総収入が budget 変数に割り当てられます。その値は、2001 年の実際の値より 10% 増加するように計算されます。

budget 変数の最初の REPORT を実行すると、次のレポートが生成されます。

```
PRODUCT: OUTERWEAR - MEN
                --BUDGET--
                ---LINE---
TIME             REVENUE
--------------- ----------
YEAR02           1,100,000
Q1.02              275,000
Q2.02              225,000
Q3.02              200,000
Q4.02              300,000
JAN02              100,000
FEB02               90,000
MAR02               85,000
APR02               82,000
MAY02               70,000
JUN02               73,000
JUL02               64,000
AUG02               69,000
SEP02               67,000
OCT02               85,000
NOV02              105,000
DEC02              110,000
```

例 9-4 では、次に集計マップが定義され、ALLOCMAP コマンドを使用して内容がそのマップに追加されます。その内容は、PROPORTIONAL 演算子を指定する 1 つの RELATION コマンドです。ALLOCATE コマンドを実行すると、親 YEAR02 から timeparent リレーションによって指定された階層へデータが割り当てられます。

アロケーションの後、budget 変数の REPORT を実行すると、次のレポートが生成されます。

```
PRODUCT: OUTERWEAR - MEN
                --BUDGET--
                ---LINE---
TIME             REVENUE
--------------- ----------
YEAR02           1,100,000
Q1.02              302,500
Q2.02              247,500
Q3.02              220,000
Q4.02              330,000
JAN02              110,000
FEB02               99,000
MAR02               93,500
APR02               90,200
MAY02               77,000
JUN02               80,300
JUL02               70,400
AUG02               75,900
SEP02               73,700
OCT02               93,500
NOV02              115,500
DEC02              121,000
```

2002 年の第 1 四半期の実際のデータが actual 変数に割り当てられ、budget 変数にコピーされます。timelockvs 値セットが定義され、1 つの値 Q1.02 に制限されます。

ファイル・ユニット値の変数が定義され、FILEOPEN ファンクションによって戻された値が割り当てられます。CONSIDER および ALLOCMAP コマンドを実行すると集計マップの内容が変更され、RELATION コマンドに PROTECT 引数が含まれます。

2 番目の ALLOCATE コマンドを実行すると、親 YEAR02 から timeparent リレーションによって指定された階層へデータが割り当てられます。ただし、このアロケーションでは、まず、ソース値から Q1.02 のロックされた値が除外されて、残りの値が分散されます。また、このコマンドを実行すると、errlogfunit ファイル・ユニットによって指定された allocerrlog ファイルにエラー・メッセージまたは情報メッセージが送信されます。

allocerrlog ファイルの内容は次のとおりです。

```
Dim      Source   Basis
TIME     BUDGET   BUDGET   Description
-------- -------- -------- -----------
  YEAR02   850000  1100000 Renormalizing data (6)
```

正規化後のアロケーション用のソース値は、元の値である 1,100,000 ではなく、850,000 です。2 番目のアロケーションの後、budget 変数の REPORT を実行すると、次のレポートが生成されます。Q1.02 の値は保護されています。

```
PRODUCT: OUTERWEAR - MEN
                --BUDGET--
                ---LINE---
TIME             REVENUE
--------------- ----------
YEAR02           1,100,000
Q1.02              250,000
Q2.02              263,793
Q3.02              234,483
Q4.02              351,724
JAN02               90,000
FEB02               82,000
MAR02               78,000
APR02               96,138
MAY02               82,069
JUN02               85,586
JUL02               75,034
AUG02               80,897
SEP02               78,552
OCT02               99,655
NOV02              123,103
DEC02              128,966
```

例 9-4 PROPORTIONAL 演算子および PROTECT 引数の使用

```
LIMIT line TO 'REVENUE'
LIMIT product TO 'OUTERWEAR - MEN'
LIMIT time TO 'YEAR02' TO 'DEC02'

" Specify no decimal places
DECIMALS = 0

budget(time 'YEAR02') = actual(time 'YEAR01') * 1.1

REPORT DOWN time budget

DEFINE budgalloc AGGMAP
```

```
ALLOCMAP 'RELATION timeparent OPERATOR PROPORTIONAL'
ALLOCATE budget USING budgalloc

REPORT DOWN time budget

" Assign actual values for first quarter of 2002.
actual(time 'Q1.02' line 'REVENUE' product 'OUTERWEAR - MEN') = 250000
actual(time 'JAN02' line 'REVENUE' product 'OUTERWEAR - MEN') = 90000
actual(time 'FEB02' line 'REVENUE' product 'OUTERWEAR - MEN') = 82000
actual(time 'MAR02' line 'REVENUE' product 'OUTERWEAR - MEN') = 78000

LIMIT time TO 'Q1.02' 'JAN02' 'FEB02' 'MAR02'

" Copy the actual values to the budget variable
budget = actual

LIMIT time TO 'Q4.01' 'JAN02' 'FEB02' 'MAR02' 'Q1.02'

DEFINE timelockvs valueset time
LIMIT timelockvs TO 'Q1.02'

DEFINE errlogfunit VARIABLE INTEGER
errlogfunit = FILEOPEN('allocerrlog' WRITE)

CONSIDER budgalloc
ALLOCMAP 'RELATION timeparent OPERATOR PROPORTIONAL ARGS PROTECT timelockvs'
ALLOCATE budget USING budgalloc ERRORLOG errlogfunit

REPORT DOWN time budget
```

第 III 部

アナリティック・ワークスペース管理

第 III 部では、アナリティック・ワークスペース用のデータを取得および生成する方法について説明します。

第 III 部に含まれる章は、次のとおりです。

10

リレーショナル表の処理

この章では、OLAP DML の SQL コマンドを使用して OLAP DML プログラムを作成する方法について説明します。これらのプログラムを使用すると、リレーショナル表を更新して、リレーショナル表とアナリティック・ワークスペース・オブジェクトの間でデータをコピーできます。

この章では、次の項目について説明します。

OLAP DML を介した SQL 文の発行

SQL は、リレーショナル表に格納されたデータを取得、削除、挿入、変更および操作する文で構成されています。OLAP DML プログラムに SQL 文を埋め込むには、次の OLAP DML の SQL コマンドを使用します。

```
SQL sql_statement
```

OLAP DML の SQL コマンドの引数として SQL 文を記述する場合、SQL 文で通常は二重引用符（"）を使用する場所に、一重引用符（'）を使用します。OLAP DML では、二重引用符（"）はコメントの始まりを示します。

サポートされている SQL 文

OLAP DML の SQL コマンドでは、Oracle でサポートされているほぼすべての SQL 文を使用できます。INSERT コマンドを使用すると、アナリティック・ワークスペース・オブジェクトからリレーショナル表へデータをコピーできます。FETCH コマンドを使用すると、リレーショナル表からアナリティック・ワークスペース・オブジェクトへデータをコピーできます。

次の Oracle SQL の拡張機能もサポートされています。

- カーソルの宣言での SELECT 文の FOR UPDATE 句。これを使用すると、カーソルと関連付けられたデータを更新または削除できます。
- UPDATE および DELETE 文の WHERE CURRENT OF *cursor* 句。これを使用すると、表の対話型変更が可能になります。

ストアド・プロシージャおよびトリガーもサポートされています。ストアド・プロシージャの使用方法の詳細は、10-28 ページの「ストアド・プロシージャおよびトリガーの使用」を参照してください。

サポートされていない SQL 文

通常、OLAP DML プログラムでは SQL コマンドを使用しますが、OLAP Worksheet ではいくつかの対話型 SQL コマンドを実行することもできます。対話型 SQL を使用するのは、通常、SELECT コマンドを実行してデータのリレーショナル表を生成する場合です。ただし、OLAP DML 内で SQL を使用する場合は、SELECT 文を含むカーソルを定義する必要があります（10-5 ページの「カーソルの宣言」を参照）。

また、COMMIT または ROLLBACK を OLAP DML の SQL コマンドの引数としてコード化する場合、これらのコマンドは無視されます。OLAP DML を使用してロールバックすることはできません。変更をコミットするには、OLAP DML の COMMIT コマンドを発行します。

リレーショナル表からのアナリティック・ワークスペースの作成

CWM1 メタデータを使用してリレーショナル表が OLAP カタログに定義されている場合、Oracle OLAP で提供されるツールを使用して、その表用のアナリティック・ワークスペースを設計および移入できます。この方法でリレーショナル表からアナリティック・ワークスペースを作成する方法の詳細は、『Oracle9*i* OLAP ユーザーズ・ガイド』を参照してください。

場合によっては、次の手順を実行して、アナリティック・ワークスペースを設計および移入できます。

1. アナリティック・ワークスペースを設計します（10-3 ページの「リレーショナル・データを保持するためのアナリティック・ワークスペースを設計および定義する手順」を参照）。
   a. OLAP `AW CREATE` コマンドを使用して、アナリティック・ワークスペースを定義します。
   b. OLAP `DEFINE` コマンドを使用して、アナリティック・ワークスペース・オブジェクトを定義します。
2. OLAP DML プログラムを定義、作成および実行して、アナリティック・ワークスペース・オブジェクトにリレーショナル・データを移入します（10-4 ページの「アナリティック・ワークスペースにリレーショナル・データを移入するプログラムを作成する手順」を参照）。
3. すべての階層のファクト・データを集計します（第 12 章「データの集計」を参照）。

リレーショナル・データを保持するためのアナリティック・ワークスペースを設計および定義する手順

リレーショナル・データベースをアナリティック・ワークスペースにマップするには、次の手順を実行します。

1. 分析するファクト・データを含む表の列を識別します。リレーショナル・データベースがデータ・ウェアハウスである場合、これらの列はメジャー表の列になります。
2. 手順 1 で識別した表の主キーを識別して、これらのいずれかのキーが任意の階層に関連付けられているかどうかを判断します。リレーショナル・データベースが完全に正規化されている場合、これを行うには、表の外部キーを調べます。リレーショナル・データベースがサマリー・データを含む場合、これを行うには、まず、主キー列が他の列の「子」であるかどうかを判断してから、完全な階層を判断するまで「親」列を調べます。
3. 階層が存在する場合は、アプリケーションに階層の各レベルの集計（サマリー・）ファクト・データが必要かどうかを決定します。
4. アプリケーションにいずれのレベルの集計データも必要ない場合は、主キー列の値の保持に使用する非階層ディメンションを定義します（3-7 ページの「ディメンションの定義」を参照）。

5. アプリケーションに一部またはすべてのレベルの集計ファクト・データが必要な場合は、次のアナリティック・ワークスペース・オブジェクトを定義して、階層を表します。

   a. 集計データが必要なレベルの値を保持するアナリティック・ワークスペース・ディメンション。階層ディメンション（3-21 ページの「階層ディメンションおよび階層ディメンションを使用する変数の定義」を参照）または連結ディメンション（3-24 ページの「連結ディメンションおよび連結ディメンションを使用する変数の定義」を参照）を定義できます。

   b. 階層のセルフ・リレーション。このリレーションは、手順 5a で説明したディメンションによってディメンション化されています。セルフ・リレーションの値は、階層の各値の親です。セルフ・リレーションの例については、3-15 ページの「例 : セルフ・リレーション」を参照してください。

6. 手順 1 で識別したファクトの変数、および分析で使用するディメンション属性の変数を定義します。通常、これらの変数は、手順 4 および 5 で識別したディメンションによってディメンション化されています。ただし、いずれかの変数が疎密に移入されている場合は、ディメンションの複合ディメンションを定義して、その複合ディメンションによって変数をディメンション化できます。

このプロセスに従って設計されたアナリティック・ワークスペースの例については、10-13 ページの「売上履歴データ用のアナリティック・ワークスペースの設計および定義」を参照してください。

アナリティック・ワークスペースにリレーショナル・データを移入するプログラムを作成する手順

アナリティック・ワークスペース構造にリレーショナル表のデータを移入するには、次の操作を実行する 1 つ以上の OLAP SQL プログラムを作成および実行します。

1. SQL カーソルを定義して、`SELECT` 文またはプロシージャと関連付けます（10-5 ページの「カーソルの宣言」を参照）。

2. 手順 1 で定義した SQL カーソルをオープンします（10-7 ページの「カーソルのオープン」を参照）。

3. OLAP DML コマンドの `SQL IMPORT` または `SQL FETCH` を使用して、手順 2 でオープンしたカーソルで指定されたデータを取得および処理します（10-8 ページの「リレーショナル表からアナリティック・ワークスペース・オブジェクトへのデータのインポートおよびフェッチ」を参照）。

   注意 : 1 つの OLAP DML プログラム内でカーソルを宣言およびオープンする必要があります。同じプログラムまたは異なるプログラムでも、データのフェッチおよびカーソルのクローズを行うことができます。

4. 手順 2 でオープンした SQL カーソルをクローズします（10-12 ページの「カーソルのクローズ」を参照）。

5. すべての SQL カーソルの定義を取り消し、SQL カーソルのメモリー・リソースを解放します（10-12 ページの「SQL カーソルのクリーンアップ」を参照）。

アナリティック・ワークスペース・オブジェクトが移入された後は、OLAP DML の UPDATE および COMMIT コマンドを使用して、これらの変更を永続的なものにできます。

この項の後半では、これらの手順の詳細を示します。アナリティック・ワークスペースにリレーショナル表のデータを移入するプログラムの例については、10-17 ページの「アナリティック・ワークスペース・オブジェクトへの売上履歴データの移入」を参照してください。

カーソルの宣言

OLAP DML プログラムでは、SELECT 文を対話的に発行することはできません。かわりに、SELECT 文を含むカーソルを定義する必要があります。問合せのコンテキストでは、カーソルは、問合せ結果のデータのリレーショナル表に含まれる行マーカーとして考えることができます。プログラムは、問合せのすべての結果を一度に受け取るのではなく、カーソルを使用して 1 行ずつ結果を受け取ります。

DECLARE CURSOR 文を使用すると、カーソルがデータ問合せの結果と名前で関連付けられます。OLAP DML の SQL コマンドの引数として指定する DECLARE CURSOR 文の構文は次のとおりです。

```
SQL DECLARE cursor-name CURSOR FOR select-statement
```

ヒント： フェッチするデータを取得する SELECT 文を記述する必要があります。可能であれば、SQL*Plus、SQL Worksheet、OLAP Worksheet などの対話型インタフェースを使用して、これらの SQL 文をテストし、目的の結果が生成されることを確認します。その後で、これらの SELECT 文を、OLAP DML プログラムで使用するために変更できます。

例：カーソルの宣言

例 10-1「カーソルの宣言」では、売上履歴サンプル・スキーマ（sh）で、カーソルの宣言によって costs という名前のリレーショナル表から行を選択しています。costs 表には、製品の識別コード（prod_id）に対する列や unit_price に対する列などの、いくつかの列が含まれています。unit_price 列は WHERE 句で使用され、これによって、戻される行は単価が 20 ドルを超える製品のみに制限されます。

例 10-1 カーソルの宣言

```
SQL DECLARE highprice CURSOR FOR -
   SELECT prod_id FROM costs -
      WHERE unit_price > 20
```

SELECT 文の WHERE 句での変数の使用

OLAP DML の SQL IMPORT コマンドで使用するカーソルを宣言すると、SELECT 文の WHERE 句でリテラル値のみを使用できます。ただし、OLAP DML の SQL FETCH コマンドで使用するカーソルを宣言した場合は、SELECT 文の WHERE 句で、リテラル値ではなく入力ホスト変数の値を使用できます。

入力ホスト変数は、SQL コマンドのパラメータとして Oracle OLAP によって提供された値です。この変数では、選択するデータまたは変更しているデータの値が指定されます。ディメンションまたはディメンション化された変数を指定する場合は、ステータス内の最初の値が使用されます。暗黙的なループは発生しませんが、FOR コマンドを使用してすべての値をループ処理できます。入力ホスト変数には、適切なデータ型を持つ任意の式を指定できます。入力ホスト変数を WHERE 句で使用して、リレーショナル表のデータの一致を検索すると、可能な場合に必要なデータ型の変換が実行されます。入力ホスト変数の値は、カーソルが宣言されるときではなく、オープンされるときに使用されます。

入力ホスト変数には、コロンで始まる任意の式（:myvar など）を指定できます。ただし、多次元の式（変数やディメンションなど）を指定する場合は、ステータス内の最初の値が使用されます。表 10-1 に、入力ホスト変数として使用可能な式の例を示します。例 10-2「入力ホスト変数の使用」では、前述の SQL コマンドを変更するプログラムの一部を示しています。WHERE 句では、明示的な値ではなく、set_price という名前のローカル変数の値が使用されます。

表 10-1 入力ホスト変数として使用可能な式の例

式のタイプ	例
変数（データベースまたはローカル）	:set_price
ディメンション	:prod
修飾データ参照	:units(prod 'P8', geog 'G12', time 'T36')
プログラム引数	:newval
テキスト式	:joinchars('first_name' 'last_name')
算術式	:intpart(6.3049) + 1
ユーザー定義ファンクション	:getgeog

例 10-2 入力ホスト変数の使用

```
VARIABLE set_price SHORT
set_price = 20
SQL DECLARE highprice CURSOR FOR -
   SELECT prod_id FROM costs -
      WHERE unit_price > :set_price
```

WHERE 句での論理演算子の使用

OLAP DML および SQL は、いずれも言語構文の一部として AND および OR を含んでいるため、これらのいずれかの論理演算子を入力ホスト変数とともに使用する場合は、カッコを使用する必要があります。カッコを使用しない場合、コマンドがあいまいになり、予期しない結果が生成される場合があります。AND および OR の前の入力ホスト変数をカッコで囲みます。

ホスト変数式がカッコで始まる場合は、それに対応する右カッコがホスト変数式の終わりとして解釈されます。右カッコの後に式のテキストを追加する場合は、式全体を別のカッコで囲む必要があります。

例 10-3 に示すプログラムの一部では、2 つの引数の値を使用して、products という名前のリレーショナル表の prod_id 列に対して選択される値の範囲を制限しています。

例 10-3 WHERE 句での論理演算子の使用

```
prod1 = 415
prod1 = 49990
      ...
SQL DECLARE twoprods CURSOR FOR -
   SELECT prod_id FROM products -
      WHERE prod_id EQ :(prod1) -
      AND :prod2
```

カーソルのオープン

SQL DECLARE CURSOR コマンドによってカーソルをデータの選択に関連付けると、SQL OPEN 文を使用してデータを取得できるようになります。特定のカーソルに対するこれらのコマンドは、同じ OLAP DML プログラム内に記述する必要があります。これらのコマンドにアンパサンド置換を含めることはできません。

引数として OPEN 文を使用する SQL コマンドの構文は次のとおりです。

```
SQL OPEN cursor-name
```

SQL OPEN コマンドを実行すると、次の操作が実行されます。

- 指定したカーソルの定義で使用される入力ホスト変数（存在する場合）の評価。
- カーソルのアクティブ・セット（SELECT 文を満たす行）の判別。
- SQL FETCH または SQL IMPORT で使用するための、カーソルのオープン状態の保持。カーソルは、結果セットの最初の行の前に配置されます。

カーソルのアクティブ・セットは、カーソルのオープン時に判別され、その後は更新されません。このため、カーソルのオープン後に入力ホスト変数の値を変更しても、カーソルのアクティブ・セットは影響を受けません。

リレーショナル表からアナリティック・ワークスペース・オブジェクトへのデータのインポートおよびフェッチ

カーソルをオープンすると、SQL IMPORT または SQL FETCH コマンド文を使用して、リレーショナル表からアナリティック・ワークスペース・オブジェクトへデータをコピーできます。これらの SQL コマンドを使用する前に、使用する表に対するアクセス権を持っていることを確認します。

SQL IMPORT または SQL FETCH のどちらを使用しても、リレーショナル表からアナリティック・ワークスペース・オブジェクトへデータがコピーされます。SQL FETCH では多くの機能が提供されますが、SQL IMPORT を使用すると、大量のデータをリレーショナル表からアナリティック・ワークスペース・オブジェクトにコピーする場合のパフォーマンスが向上します。

- **SQL FETCH** を使用すると、SQL カーソルによって指定したデータが取得および処理され、取得データが OLAP オブジェクトに割り当てられます。FETCH 文を使用してリレーショナル表からデータを取得する場合は、この文をループに含めるか、または LOOP 引数を使用してカーソルのアクティブ・セットのすべての行を取得する必要があります。また、SQL FETCH に THEN 句を含めると、取得データに対する処理を実行できます。引数として FETCH 文を使用する SQL コマンドの構文は次のとおりです。

  ```
  SQL FETCH cursor [LOOP [loopcount]] INTO :targets... -
  [THEN action-statements...]
  ```

- **SQL IMPORT** を使用すると、カーソルの位置がカーソルのアクティブ・セット内の 1 つ後ろの行に進み、選択したフィールドがアナリティック・ワークスペース・オブジェクトにインポートされます。引数として IMPORT 文を使用する OLAP DML の SQL コマンドの構文は次のとおりです。

  ```
  SQL IMPORT cursor INTO :targets...
  ```

SQL IMPORT および SQL FETCH の構文で、*targets* は出力ホスト変数を表します。出力ホスト変数は、リレーショナル表から取得されたデータの格納に使用されるアナリティック・ワークスペース・オブジェクトです。DECLARE CURSOR 文では、出力ホスト変数の順序を列の順序と一致させる必要があります。また、各出力ホスト変数の名前の前にはコロンを付ける必要があります。データを受け取る変数またはディメンションは、定義済である必要があります。また、互換性のあるデータ型を持つ必要もあります。

IMPORT および FETCH では、いずれも次のうち 1 つ以上の出力ホスト変数を指定できます。

```
[MATCH] dimension|surrogate
APPEND  dimension
ASSIGN  surrogate
variable|qualified data reference|relation|composite
```

出力ホスト変数がディメンションである場合、取得される値はホスト変数名の前に指定したキーワードに基づいて処理されます。MATCH キーワード（デフォルト）または APPEND キーワードのいずれかを指定できます。

- MATCH キーワードを使用すると、ディメンションの既存の値と同じ値のみがフェッチされ、新しい値に対してはエラーが通知されます。このキーワードは、すでにディメンションが保持されている変数にデータをフェッチする場合に使用します。ディメンションはデータの位置合せのためにのみフェッチに含まれます。
- APPEND キーワードを使用すると、一致しないすべての値が、ディメンション値のリストの最後に追加されます。FETCH の場合も、ターゲットに次の構文を使用して、位置に基づいて出力ホスト変数に値を追加できます。

  APPEND [*position*] *dimension*

表 10-2 に、出力ホスト変数として使用可能な式の例を示します。

表 10-2 出力ホスト変数として使用可能な式の例

式のタイプ	例
変数（データベースまたはローカル）	:sales_quantity_sold
ディメンションまたはサロゲート	:prodid
修飾データ参照	:sales_quantity_sold(prod_id 415 cust_id 18670 time_id '1998-01-04' channel_id 'S' promo_id 9999)

ディメンション化されたアナリティック・ワークスペースの変数にデータをフェッチする場合は、常に、ディメンション値をフェッチに含める必要があります。これと同時に新しいディメンション値を追加できますが、新しい値がアナリティック・ワークスペースにすでに存在している場合は追加する必要はありません。かわりに、データの位置合せのためにディメンション値をフェッチで使用します。いずれの場合も、変数の値をフェッチする前にディメンション値をフェッチする必要があります。そうでない場合、ディメンション値でフェッチがループされません。

> **重要：** データがディメンションに書き込まれる際、一時的にディメンションのステータスが一致する値または追加される値に制限されます。これは、IMPORT 文または FETCH 文にも指定したディメンションによってディメンション化された出力ホスト変数が含まれる場合、値がこれらの変数に割り当てられると、ステータスが一時的に制限されることを意味します。

リレーショナル表の NULL 値は NA に相当します。OLAP DML の変数では、NULL 値で問題が発生せず、NA として表示されます。ただし、NA というディメンション値を持つことは

できません。このため、null 値を持つすべての行は、ディメンションにフェッチしている列では廃棄されます。

例 : リレーショナル表からアナリティック・ワークスペース・オブジェクトへのデータのコピー

短時間で分析を行うために、リレーショナル表からアナリティック・ワークスペースへデータをコピーする必要がある場合があります。たとえば、売上履歴サンプル・データベースには sales 表が含まれます（10-10 ページの例 10-4 を参照）。この表には、prod_id、cust_id、time_id、channel_id および promo_id というキーと、2 つのファクト（quantity_sold および amount_sold）が存在します。

OLAP DML で使用可能な予測コマンドを使用して、製品 415 の 2002 年の販売数を予測すると想定します。OLAP DML を使用してこの分析を実行するには、アナリティック・ワークスペースにデータが存在する必要があります。アナリティック・ワークスペースにデータをコピーするには、データを保持するためのアナリティック・ワークスペース・オブジェクトを定義して、リレーショナル表からアナリティック・ワークスペース・オブジェクトへデータをコピーする OLAP DML プログラムを作成した後、このプログラムを実行します。

各キー列（aw_prod_id、aw_cust_id、aw_time_id、aw_channel_id および aw_promo_id）に対して 1 つのアナリティック・ワークスペース・ディメンションを定義して、他の列（aw_quantity_sold および aw_amount_sold）のデータを保持するためのアナリティック・ワークスペースの（これらのディメンションでディメンション化されている）変数を定義すると、sales 表をアナリティック・ワークスペース・オブジェクトに簡単にマップできます。ただし、この場合、変数は時間ディメンションで疎密になります。疎密な状態を回避するには、すべてのキー・ディメンションを表す複合ディメンションを定義し、この複合ディメンションを使用してアナリティック・ワークスペースの変数を定義します（10-11 ページの例 10-5「売上データ用のアナリティック・ワークスペースの定義」を参照）。

10-11 ページの例 10-6「import_sales_for_prod415 プログラム」では、SQL IMPORT を使用して、リレーショナル表からアナリティック・ワークスペース・オブジェクトへデータをコピーする方法を示しています。fetch_sales_for_prod415 プログラム（10-11 ページの例 10-7「fetch_sales_for_prod415 プログラム」を参照）では、SQL FETCH を使用して、リレーショナル表からアナリティック・ワークスペース・オブジェクトへデータをコピーする方法を示しています。これらのプログラムでは、aw_prod_id、aw_cust_id、aw_time_id、aw_channel_id および aw_promo_id の値が事前にアナリティック・ワークスペースにコピーされていないと想定します。複合ディメンションを定義している場合、Oracle OLAP は、他のアナリティック・ワークスペース・オブジェクトと同様に、その複合ディメンションを自動的に移入します。

例 10-4 sales 表の説明

```
PROD_ID                                   NOT NULL NUMBER(6)
CUST_ID                                   NOT NULL NUMBER
TIME_ID                          NOT NULL DATE
CHANNEL_ID                                NOT NULL CHAR(1)
PROMO_ID                                  NOT NULL NUMBER(6)
```

```
QUANTITY_SOLD                              NOT NULL NUMBER(3)
AMOUNT_SOLD                                NOT NULL NUMBER(10,2)
```

例 10-5 売上データ用のアナリティック・ワークスペースの定義

```
DEFINE aw_prod_id DIMENSION NUMBER (6)
DEFINE aw_cust_id DIMENSION NUMBER (6)
DEFINE aw_date DIMENSION TEXT
DEFINE aw_channel_id DIMENSION TEXT
DEFINE aw_promo_id DIMENSION NUMBER (6)
DEFINE aw_sales_dims COMPOSITE <aw_prod_id aw_cust_id aw_date -
     aw_channel_id aw_promo_id>
DEFINE aw_sales_quantity_sold VARIABLE NUMBER (3) <aw_sales_dims <aw_prod_id -
     aw_cust_id aw_date aw_channel_id paw_romo_id>>
DEFINE aw_sales_amount_sold VARIABLE NUMBER (10,2) <aw_sales_dims <aw_prod_id -
     aw_cust_id aw_date aw_channel_id aw_promo_id>>
```

例 10-6 import_sales_for_prod415 プログラム

```
ALLSTAT
NLS_DATE_FORMAT = '<YYYY><MM><DD>'
DATEFORMAT = '<YYYY>-<MM>-<DD>'
" Declare a cursor named GRABDATA
SQL DECLARE grabdata CURSOR FOR SELECT prod_id, cust_id, time_id, -
    channel_id, promo_id, quantity_sold, amount_sold FROM sh.sales -
    WHERE prod_id = 415
" Import new values into the analytic workspace objects
SQL IMPORT grabdata INTO :APPEND aw_prod_id -
                         :APPEND aw_cust_id -
                         :APPEND aw_date -
                         :APPEND aw_channel_id -
                         :APPEND aw_promo_id -
                         :aw_sales_quantity_sold -
                         :aw_sales_amount_sold
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
```

例 10-7 fetch_sales_for_prod415 プログラム

```
ALLSTAT
NLS_DATE_FORMAT = '<YYYY><MM><DD>'
DATEFORMAT = '<YYYY>-<MM>-<DD>'
" Declare a cursor named GRABDATA
SQL DECLARE grabdata CURSOR FOR SELECT prod_id, cust_id, time_id, -
    channel_id, promo_id, quantity_sold, amount_sold FROM sh.sales -
    WHERE prod_id = 415
```

```
" Open the cursor
SQL OPEN grabdata
" Fetch new values into the analytic workspace objects
SQL FETCH grabdata LOOP INTO :APPEND aw_prod_id -
                             :APPEND aw_cust_id -
                             :APPEND aw_date -
                             :APPEND aw_channel_id -
                             :APPEND aw_promo_id -
                             :aw_sales_quantity_sold -
                             :aw_sales_amount_sold
" Close the cursor
SQL CLOSE grabdata
" Cleanup from SQL query
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
```

カーソルのクローズ

カーソルを使用してアクティブ・セットのすべてのデータを取得した後、カーソルをクローズします。カーソルを使用してアクティブ・セットの最初の行からデータを再度取得する場合は、カーソルを再度宣言せずに、OPEN 文を使用できます。CLOSE 文では、カーソルの宣言は取り消されず、アクティブ・セットが未定義になるのみです。

OLAP DML の SQL コマンドで引数として CLOSE 文を使用する場合の構文は次のとおりです。

```
SQL CLOSE cursor-name
```

SQL カーソルのクリーンアップ

OLAP DML の SQL コールを完全に終了したら、すべての SQL カーソルの宣言を取り消し、それらのメモリー・リソースを解放する必要があります。これらの処理を実行するには、次のように、OLAP DML の SQL コマンドに引数として CLEANUP を使用します。

```
SQL CLEANUP
```

この方法ですべての SQL カーソルを取り消した後は、新しい SQL DECLARE CURSOR および SQL OPEN コマンドを発行しないかぎり、SQL カーソルを再利用できません。

例 : 売上履歴表からのアナリティック・ワークスペースの作成

売上履歴サンプル・データベースには、次の 6 つのディメンション表および 2 つのファクト表が存在します（詳細は『Oracle9*i* サンプル・スキーマ』を参照）。

- countries: 主キー country_id を持つディメンション表
- customers: 主キー customer_id および外部キー country_id を持つディメンション表
- promotions: 主キー promo_id を持つディメンション表
- products: 主キー product_id を持つディメンション表
- channels: 主キー channel_id を持つディメンション表
- times: 主キー time_id を持つディメンション表
- sales: customer_id、promo_id、product_id、channel_id および time_id のキーを持つファクト表
- costs: product_id および time_id のキーを持つファクト表

売上履歴サンプル・データベース内のすべてのファクト・データを分析すると想定します。これを行うには、アナリティック・ワークスペースを設計および定義する必要があります（10-13 ページの「売上履歴データ用のアナリティック・ワークスペースの設計および定義」を参照）。次に、OLAP DML プログラムを作成して、必要なリレーショナル・データをアナリティック・ワークスペースにコピーする必要があります（10-13 ページの「アナリティック・ワークスペース・オブジェクトへの売上履歴データの移入」を参照）。

売上履歴データ用のアナリティック・ワークスペースの設計および定義

売上履歴用のアナリティック・ワークスペースを設計および定義するには、10-3 ページの「リレーショナル・データを保持するためのアナリティック・ワークスペースを設計および定義する手順」に示す手順を実行します。実際の手順は次のとおりです。

1. 次の OLAP DML コマンドを使用して、awsh という名前のアナリティック・ワークスペースを作成します。

   ```
   AW CREATE awsh
   ```

2. ファクト・データを識別します。sales 表には、quantity_sold および amount_sold 列に分析するファクトが含まれています。costs 表には、unit_cost および unit_price 列に関連するファクト・データが含まれています。
3. sales および costs 表の主キーを識別します。sales の主キーは、prod_id、cust_id、time_id、channel_id および promo_id です。costs の主キーは、prod_id および time_id です。
4. 主キーを参照して、次の売上履歴データベースの階層を識別します。

- 製品 : products 表の列にマップされる 4 つのレベル（prod_id、prod_subcategory、prod_category および products_all）を持つ階層。この階層の最下位レベルは prod_id、最上位レベルは products_all です。
- チャネル : channels 表の列にマップされる 3 つのレベル（channel_id、channel_class および channels_all）を持つ階層。この階層の最下位レベルは channel_id、最上位レベルは channels_all です。
- プロモーション : promotions 表の列にマップされる 4 つのレベル（promo_id、promo_subcategory、promo_category および promos_all）を持つ階層。この階層の最下位レベルは promo_id、最上位レベルは promos_all です。
- 顧客 : 2 つの異なるリレーショナル表の列にマップされる 7 つのレベルを持つ階層。7 つのレベルのうち 4 つのレベル（country_id、region、subreagion および world）は countries 表の列にマップされ、3 つのレベル（cust_id、state_province および city）は customers 表の列にマップされます。この階層の最下位レベルは cust_id、最上位レベルは world です。
- 時間階層 : 暦年および会計年度の 2 つの時間階層。

  暦年 : times 表の列にマップされる 5 つのレベル（time_id、cal_week_num、cal_month_num、cal_quarter_num および cal_year）を持つ階層。

  会計年度 : times 表の列にマップされる 5 つのレベル（time_id、fis_week_num、fis_month_num、fis_quarter_num および fis_year）を持つ階層。

また、prod_id と supplier_id の間には 1 対多の関係があります。

5. アプリケーションで、製品階層、顧客階層、チャネル階層およびプロモーション階層の各レベルのファクト・データを集計します。時間階層では、階層の最下位レベル（time_id）および年レベル（cal_year および fis_year）の 2 つのレベルを持つ階層のみがアプリケーションで必要です。
6. 次のアナリティック・ワークスペース・オブジェクトを定義して、階層を表します。
   - 製品階層、顧客階層、チャネル階層およびプロモーション階層では、階層の各レベルのディメンション、各階層の連結ディメンション、および各連結ディメンションの親子セルフ・リレーションを定義します。これらの定義の例については、10-15 ページの例 10-8 ～ 10-16 ページの例 10-11 を参照してください。
   - 時間階層では、2 つの階層を定義します。2 つの階層の名前を含むディメンションを作成します。time_id、fis_year および cal_year のベース・ディメンションを定義し、これらのすべてのディメンションをベース・ディメンションとして指定する連結ディメンションを定義します。2 つの時間階層が存在するため、時間階層に対して作成される親子セルフ・リレーションは、連結ディメンションおよび階層によって（名前ごとに）ディメンション化されます。これらの定義の例については、10-16 ページの例 10-12「時間階層用のアナリティック・ワークスペースの定義」を参照してください。

- ファクト（quantity_sold、amount_sold、unit_cost および unit_price）では、アナリティック・ワークスペースの変数を定義します。これらのすべての変数が連結ディメンションによってディメンション化されている場合は、疎密に移入されるため、各変数に対して 1 つの複合ディメンションを定義します。変数はこれらの複合ディメンションによってディメンション化されます。これらの複合ディメンションの定義を含むファクト・データの変数の定義の例については、10-16 ページの例 10-13「ファクトの変数用のアナリティック・ワークスペースの定義」を参照してください。

アプリケーションには、他のデータは必要ありません。ただし、10-17 ページの例 10-14「プロモーション・ディメンション属性の変数の定義」に、プロモーション属性をマップ可能なアナリティック・ワークスペースの変数の定義を示しています。awsh アナリティック・ワークスペースのリレーショナル・ビューを定義する方法の例については、『Oracle9*i* OLAP ユーザーズ・ガイド』の OLAP_TABLE ファンクションの使用例を参照してください。

例 10-8 製品階層用のアナリティック・ワークスペースの定義

```
DEFINE aw_prod_id DIMENSION NUMBER (6)
DEFINE aw_prod_subcategory DIMENSION TEXT
DEFINE aw_prod_category DIMENSION TEXT
DEFINE aw_products_all DIMENSION TEXT
DEFINE aw_products DIMENSION CONCAT (aw_products_all -
                                     aw_prod_category -
                                     aw_prod_subcategory -
                                     aw_prod_id)
DEFINE aw_products.parents RELATION aw_products <aw_products>
DEFINE aw_supplier_id DIMENSION TEXT
DEFINE aw_prod_id.aw_supplier_id RELATION aw_supplier_id <aw_prod_id>
```

例 10-9 チャネル階層用のアナリティック・ワークスペースの定義

```
DEFINE aw_channel_id DIMENSION TEXT
DEFINE aw_channel_class DIMENSION TEXT
DEFINE aw_channels_all DIMENSION TEXT
DEFINE aw_channels DIMENSION CONCAT(aw_channels_all -
                                    aw_channel_class -
                                    aw_channel_id)
DEFINE aw_channels.parents RELATION aw_channels <aw_channels>
```

例 10-10 プロモーション階層用のアナリティック・ワークスペースの定義

```
DEFINE aw_promo_id DIMENSION NUMBER(6)
DEFINE aw_promo_subcategory DIMENSION TEXT
DEFINE aw_promo_category DIMENSION TEXT
DEFINE aw_promos_all DIMENSION TEXT
DEFINE aw_promos DIMENSION CONCAT(aw_promos_all -
                                  aw_promo_category -
```

```
                                aw_promo_subcategory -
                                aw_promo_id)
DEFINE aw_promos.parents RELATION aw_promos <aw_promos>
```

例 10-11 顧客階層用のアナリティック・ワークスペースの定義

```
DEFINE aw_cust_id DIMENSION NUMBER (8)
DEFINE aw_city DIMENSION TEXT
DEFINE aw_state_province DIMENSION TEXT
DEFINE aw_country_id DIMENSION TEXT
DEFINE aw_subregion DIMENSION TEXT
DEFINE aw_region DIMENSION TEXT
DEFINE aw_world DIMENSION TEXT
DEFINE aw_customers DIMENSION CONCAT(aw_world -
                                aw_region -
                                aw_subregion -
                                aw_country_id -
                                aw_state_province -
                                aw_city -
                                aw_cust_id)
DEFINE aw_customers.parents RELATION aw_customers <aw_customers>
```

例 10-12 時間階層用のアナリティック・ワークスペースの定義

```
DEFINE aw_time_id DIMENSION TEXT
DEFINE aw_cal_year DIMENSION NUMBER(4)
DEFINE aw_fis_year DIMENSION NUMBER(4)
DEFINE aw_times DIMENSION CONCAT (aw_cal_year -
                                aw_fis_year -
                                aw_time_id)
DEFINE aw_times_hiernames DIMENSION TEXT
DEFINE aw_times.parents RELATION aw_times <aw_times aw_times_hiernames>
```

例 10-13 ファクトの変数用のアナリティック・ワークスペースの定義

```
DEFINE aw_costsdims COMPOSITE <aw_products aw_times>
DEFINE aw_unit_cost VARIABLE  NUMBER (10,2) <aw_costsdims -
        <aw_products aw_times>>
DEFINE aw_unit_price VARIABLE NUMBER (10,2) <aw_costsdims -
        <aw_products aw_times>>

DEFINE aw_salesdims COMPOSITE <aw_products aw_customers aw_times -
              aw_channels aw_promos>
DEFINE aw_quantity_sold VARIABLE NUMBER(3) <aw_salesdims -
             <aw_products aw_customers aw_times aw_channels aw_promos>>
DEFINE aw_amount_sold VARIABLE NUMBER(10,2) <aw_salesdims -
             <aw_products aw_customers aw_times aw_channels aw_promos>>
```

例 10-14 プロモーション・ディメンション属性の変数の定義

```
DEFINE aw_promo_name VARIABLE TEXT <aw_promo_id>
DEFINE aw_promo_cost VARIABLE NUMBER(10,2) <aw_promo_id>
DEFINE aw_promo_begin_date VARIABLE DATE <aw_promo_id>
DEFINE aw_promo_end_date VARIABLE DATE <aw_promo_id>
```

アナリティック・ワークスペース・オブジェクトへの売上履歴データの移入

この項では、売上履歴リレーショナル・データベースから awsh という名前のアナリティック・ワークスペースのオブジェクトにデータをコピーする、多くの OLAP DML プログラムの例を示します。

- リレーショナル表からアナリティック・ワークスペースのディメンションおよび変数にデータをコピーするプログラムの例は次のとおりです。
  - 10-18 ページの例 10-15「get_products_hier プログラム」では、APPEND キーワードとともに SQL FETCH コマンドを使用して、ディメンション表から aw_products 連結ディメンションのベース・ディメンションにデータがコピーされます。aw_products のベース・ディメンションが移入されるときに、aw_products 自体は Oracle OLAP によって自動的に移入されます。SQL FETCH コマンドの THEN 句を実行すると、Oracle OLAP によって aw_products の親子セルフ・リレーションにデータがフェッチされます。このプログラムでは、aw_supplier_id ディメンションおよびそのリレーションも移入されます。
  - 10-19 ページの例 10-16「get_channels_hier プログラム」、10-20 ページの例 10-17「get_promos_hier プログラム」、10-20 ページの例 10-18「get_customers_hier プログラム」および 10-21 ページの例 10-19「get_times_hiers プログラム」では、ディメンション表から、階層ディメンションを表すために使用されるアナリティック・ワークスペースのディメンションおよびリレーションにデータがコピーされます。これらのプログラムを実行する前はディメンションが空であるため、SQL FETCH コマンドでは APPEND キーワードが使用されます。ベース・ディメンションが移入されるときに、階層を表す連結ディメンションは Oracle OLAP によって自動的に移入されます。SQL FETCH コマンドの THEN 句を実行すると、Oracle OLAP によって、階層を表す連結ディメンションの親子セルフ・リレーションにデータがフェッチされます。
- リレーショナル表からアナリティック・ワークスペースの変数にファクトをコピーするプログラムの例は次のとおりです。これらの例では、変数のベース・ディメンションがすでに移入されていると想定します。このため、これらのプログラムの SQL FETCH コマンドでは、MATCH キーワードが使用されます。また、変数がディメンション化される複合ディメンションは連結ディメンションで構成されているため、SQL FETCH コマンドでは、QDR を使用して変数のディメンション値が指定されます。
  - 10-23 ページの例 10-20「get_costs プログラム」では、costs 表からアナリティック・ワークスペースの変数にファクトがコピーされます。

  - 10-23 ページの例 10-21「get_sales プログラム」では、sales 表からアナリティック・ワークスペースの変数にファクトがコピーされます。

- 10-24 ページの例 10-23「get_promos_attr プログラム」では、プロモーション・ディメンション表からアナリティック・ワークスペースの変数に属性データがコピーされます。このプログラムでは、ベース・ディメンションがすでに移入されていると想定して、MATCH キーワードとともに SQL IMPORT コマンドが使用されます。

例 10-15 get_products_hier プログラム

```
ALLSTAT
" Fetch values into the products hierarchy
SQL DECLARE grabprods CURSOR FOR SELECT prod_total, -
                                        prod_category, -
                                        prod_subcategory, -
                                        prod_id -
                                   FROM sh.products
SQL OPEN grabprods
SQL FETCH grabprods LOOP INTO :APPEND aw_products_all -
                              :APPEND aw_prod_category -
                              :APPEND aw_prod_subcategory -
                              :APPEND aw_prod_id

SQL CLOSE grabprods
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
" Fetch values into supplier_id
SQL DECLARE grabsupid CURSOR FOR SELECT supplier_id -
                                 FROM sh.products
SQL OPEN grabsupid
SQL FETCH grabsupid LOOP INTO :APPEND aw_supplier_id
SQL CLOSE grabsupid
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT

" Populate self-relation for concat dimension
" and relation between aw_prod_id and aw_supplier_id
SQL DECLARE makerels CURSOR FOR SELECT prod_total, -
                                       prod_category, -
                                       prod_subcategory, -
                                       prod_id, -
                                       supplier_id -
                                  FROM sh.products
```

```
SQL OPEN makerels
SQL FETCH makerels LOOP INTO :MATCH aw_products_all -
                             :MATCH aw_prod_category -
                             :MATCH aw_prod_subcategory -
                             :MATCH aw_prod_id -
                             :MATCH aw_supplier_id -
           THEN aw_products.parents(aw_products aw_prod_id) -
               = aw_products(aw_prod_subcategory aw_prod_subcategory) -
           aw_products.parents(aw_products aw_prod_subcategory) -
              = aw_products(aw_prod_category aw_prod_category) -
           aw_products.parents(aw_products aw_prod_category) -
              = aw_products(aw_products_all aw_products_all) -
           aw_prod_id.aw_supplier_id = aw_supplier_id
SQL CLOSE makerels
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
```

例 10-16 get_channels_hier プログラム

```
ALLSTAT
" Fetch values for the Channels hierarchy
" and populate self-relation for the hierarchy
SQL DECLARE grabchanneldata CURSOR FOR SELECT channel_total, -
                                          channel_class, -
                                          channel_id -
                                 FROM sh.channels
SQL OPEN grabchanneldata
" Fetch values into analytic workspace objects for the the channels hierararchy
SQL FETCH grabchanneldata LOOP INTO :APPEND aw_channels_all -
                             :APPEND aw_channel_class -
                             :APPEND aw_channel_id -
         THEN aw_channels.parents(aw_channels aw_channel_id) -
              = aw_channels(aw_channel_class aw_channel_class) -
         aw_channels.parents(aw_channels aw_channel_class) -
              = aw_channels(aw_channels_all aw_channels_all)
SQL CLOSE grabchanneldata
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
```

例 10-17 get_promos_hier プログラム

```
ALLSTAT
" Fetch values for the Promos hierarchy
" and populate self-relation for the hierarchy
SQL DECLARE grabpromodata CURSOR FOR SELECT promo_total, -
                                           promo_category, -
                                           promo_subcategory, -
                                           promo_id -
                                    FROM sh.promotions
SQL OPEN grabpromodata
SQL FETCH grabpromodata LOOP INTO :APPEND aw_promos_all -
                                  :APPEND aw_promo_category -
                                  :APPEND aw_promo_subcategory -
                                  :APPEND aw_promo_id -
        THEN aw_promos.parents(aw_promos aw_promo_id) -
                    = aw_promos(aw_promo_subcategory aw_promo_subcategory) -
                  aw_promos.parents(aw_promos aw_promo_subcategory) -
                   = aw_promos(aw_promo_category aw_promo_category) -
             aw_promos.parents(aw_promos aw_promo_category) -
              = aw_promos(aw_promos_all aw_promos_all)
SQL CLOSE grabpromodata
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
```

例 10-18 get_customers_hier プログラム

```
ALLSTAT
" Fetch values for the Customers hierarchy from the countries table
" and populate the self-relation for the hierarchy with these values
SQL DECLARE grabcountrydata CURSOR FOR SELECT country_total, -
                                country_region, -
                                country_subregion, -
                                country_id -
                             FROM sh.countries
SQL OPEN grabcountrydata
SQL FETCH grabcountrydata LOOP INTO :APPEND aw_world -
                                    :APPEND aw_region -
                                    :APPEND aw_subregion -
                                    :APPEND aw_country_id -
      THEN aw_customers.parents(aw_customers aw_country_id) = -
             aw_customers(aw_subregion aw_subregion) -
           aw_customers.parents(aw_customers aw_subregion) = -
             aw_customers(aw_region aw_region) -
           aw_customers.parents(aw_customers aw_region) = -
             aw_customers(aw_world aw_world)
```

```
SQL CLOSE grabcountrydata
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
" Fetch values for the Customers hierarchy from the customers table
" and populate the self-relation for the hierarchy with these values
SQL DECLARE grabcustdata CURSOR FOR SELECT country_id, -
                                         cust_state_province, -
                                         cust_city, -
                                         cust_id -
                                  FROM sh.customers
SQL OPEN grabcustdata
SQL FETCH grabcustdata LOOP INTO :MATCH aw_country_id -
                                 :APPEND aw_state_province -
                                 :APPEND aw_city -
                                 :APPEND aw_cust_id -
                THEN aw_customers.parents(aw_customers aw_cust_id) = -
                   aw_customers(aw_city aw_city) -
               aw_customers.parents(aw_customers aw_city) = -
                   aw_customers(aw_state_province aw_state_province) -
               aw_customers.parents(aw_customers aw_state_province) = -
                   aw_customers(aw_country_id aw_country_id)
SQL CLOSE grabcustdata
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
```

例 10-19 get_times_hiers プログラム

```
NLS_DATE_FORMAT = '<YYYY><MM><DD>'
DATEFORMAT = '<YYYY>-<MM>-<DD>'
" Populate the hierachy name dimension with names of hierarchies
MAINTAIN aw_times_hiernames ADD 'Calendar' 'Fiscal'
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT

" Fetch values for the CalTimes and FisTimes hierarchies
" and populate self-relation time
SQL DECLARE grabcalyear CURSOR FOR SELECT calendar_year -
                                 FROM sh.times
SQL OPEN grabcalyear
SQL FETCH grabcalyear LOOP INTO :APPEND aw_cal_year
SQL CLOSE grabcalyear
SQL CLEANUP
```

```
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
SQL DECLARE grabfisyear CURSOR FOR SELECT fiscal_year -
                                    FROM sh.times
SQL OPEN grabfisyear
SQL FETCH grabfisyear LOOP INTO :APPEND aw_fis_year
SQL CLOSE grabfisyear
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
SQL DECLARE grabtimeid CURSOR FOR SELECT time_id -
                                    FROM sh.times
SQL OPEN grabtimeid
SQL FETCH grabtimeid LOOP INTO :APPEND aw_time_id
SQL CLOSE grabtimeid
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
UPDATE
COMMIT
ALLSTAT
LIMIT aw_times_hiernames TO 'Calendar'
SQL DECLARE makecalhier CURSOR FOR SELECT calendar_year, -
                                          time_id -
                                FROM sh.times
SQL OPEN makecalhier
SQL FETCH makecalhier LOOP INTO :MATCH aw_cal_year -
                                :MATCH aw_time_id -
       THEN aw_times.parents(aw_times aw_time_id) -
              = aw_times(aw_cal_year aw_cal_year)
SQL CLOSE makecalhier
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
ALLSTAT
UPDATE
COMMIT
LIMIT aw_times_hiernames TO 'Fiscal'
SQL DECLARE makefishier CURSOR FOR SELECT fiscal_year, -
                                       time_id -
                                FROM sh.times
SQL OPEN makefishier
SQL FETCH makefishier LOOP INTO :MATCH aw_fis_year -
                                :MATCH aw_time_id -
       THEN aw_times.parents(aw_times aw_time_id) -
              = aw_times(aw_fis_year aw_fis_year)
SQL CLOSE makefishier
```

```
SQL CLEANUP
" Update the analytic workspace and make the updates permanent
ALLSTAT
UPDATE
COMMIT
```

例 10-20 get_costs プログラム

```
ALLSTAT
NLS_DATE_FORMAT = '<YYYY><MM><DD>'
DATEFORMAT = '<YYYY>-<MM>-<DD>'
" Declare a cursor named grabcosts
SQL DECLARE grabcosts CURSOR FOR SELECT prod_id, -
                                       time_id, -
                                       unit_cost, -
                                       unit_price -
                            FROM sh.costs
" Open the cursor
SQL OPEN grabcosts
" Import the data
SQL FETCH grabcosts LOOP INTO :MATCH aw_prod_id -
                         :MATCH aw_time_id -
                         :aw_unit_cost (aw_products aw_prod_id -
                     aw_times aw_time_id) -
                         :aw_unit_price (aw_products aw_prod_id -
                     aw_times aw_time_id)
" Close the cursor
SQL CLOSE grabcosts
" Cleanup from SQL query
SQL CLEANUP
" Update and make changes permanent
UPDATE
COMMIT
```

例 10-21 get_sales プログラム

```
ALLSTAT
NLS_DATE_FORMAT = '<YYYY><MM><DD>'
DATEFORMAT = '<YYYY>-<MM>-<DD>'
" Declare a cursor named grabsales
SQL DECLARE grabsales CURSOR FOR SELECT prod_id, -
                                        cust_id, -
                                        time_id, -
                                        channel_id, -
                                        promo_id, -
                                        quantity_sold, -
                                        amount_sold -
```

```
                        FROM sh.sales
" Open the cursor
SQL OPEN grabsales
" Import values into analytic workspace objects
SQL FETCH grabsales LOOP INTO :MATCH aw_prod_id -
                              :MATCH aw_cust_id -
                              :MATCH aw_time_id -
                              :MATCH aw_channel_id -
                              :MATCH aw_promo_id -
                              :aw_quantity_sold (aw_products aw_prod_id -
                                           aw_customers aw_cust_id -
                                            aw_times aw_time_id -
                                          aw_channels aw_channel_id -
                                            aw_promos aw_promo_id) -
                              :aw_amount_sold (aw_products aw_prod_id -
                                         aw_customers aw_cust_id -
                                         aw_times aw_time_id -
                                         aw_channels aw_channel_id -
                                          aw_promos aw_promo_id)
" Close the cursor
SQL CLOSE grabsales
" Cleanup from SQL query
SQL CLEANUP
" Update and make changes permanent
UPDATE
COMMIT
```

例 10-22 プロモーション・ディメンション属性の変数の定義

```
DEFINE aw_promo_name VARIABLE TEXT <aw_promo_id>
DEFINE aw_promo_cost VARIABLE NUMBER(10,2) <aw_promo_id>
DEFINE aw_promo_begin_date VARIABLE DATE <aw_promo_id>
DEFINE paw_romo_end_date VARIABLE DATE <aw_promo_id>
```

例 10-23 get_promos_attr プログラム

```
ALLSTAT
" Declare a cursor named grabpromoattr
SQL DECLARE grabpromoattr CURSOR FOR SELECT promo_id, -
                                            promo_name, -
                                            promo_cost, -
                                            promo_begin_date, -
                                            promo_end_date -
                           FROM sh.promotions
" Open the cursor
SQL OPEN grabpromoattr
" Import new values into the analytic workspace objects
```

```
SQL IMPORT grabpromoattr INTO :MATCH aw_promo_id -
                                     :aw_promo_name -
                                     :aw_promo_cost -
                                     :aw_promo_begin_date -
                                     :aw_promo_end_date
" Close the cursor
SQL CLOSE grabpromoattr
" Cleanup from SQL query
SQL CLEANUP
" Update and make changes permanent
UPDATE
COMMIT
```

アナリティック・ワークスペース・オブジェクトからリレーショナル表へのデータの書込み

アナリティック・ワークスペース・オブジェクトからデータをコピーするには、OLAP DML の SQL コマンドの引数として、SQL の INSERT または UPDATE 文を使用します。この場合、OLAP DML の SQL がループするようにコード化して、SQL 文でアナリティック・ワークスペースの変数を入力ホスト変数として使用します。ただし、OLAP DML コマンドの引数として PREPARE および EXECUTE 文を使用してダイレクト・インサートを行うと、パフォーマンスが向上します。

> **ヒント：** アナリティック・ワークスペース・データのビューを定義することによって、アナリティック・ワークスペースからリレーショナル表にデータをコピーせずに、SQL SELECT 文でアナリティック・ワークスペースのデータにアクセスできます。アナリティック・ワークスペース・データのリレーショナル・ビューを定義する方法の詳細は、『Oracle9*i* OLAP ユーザーズ・ガイド』を参照してください。

SQL PREPARE および SQL EXECUTE の使用

PREPARE および EXECUTE 文の構文は次のとおりです。

```
SQL PREPARE statement-name FROM sql-statement [insert-options]
SQL EXECUTE statement-name
```

次に、これらの文の引数について説明します。

- *statement-name* には、*sql-statement* から生成する実行コードに割り当てる名前を指定します。*statement-name* を再定義するには、別の SQL PREPARE コマンドを発行するのみです。

- *sql-statement* には、より効率的に実行するためにプリコンパイルする SQL 文を指定します。アンパサンド（&）置換、またはプログラムのコンパイル時に定義されていない変数を含めることはできません。
- *insert-options* には、DIRECT、NOLOG または PARTITION を指定します。これらのオプションは、*sql-statement* が INSERT 文である場合に適用されます。挿入オプションの値を指定せずに INSERT 文を準備した場合は、Oracle OLAP によって DIRECT および NOLOG 挿入オプションに NO が指定されます。PARTITION オプションの値は指定されません。そのため、デフォルトでは、準備した INSERT では通常の挿入が実行され、REDO 情報が REDO ログ・ファイルに記録され、他のセッションではプログラムが値を挿入している表にデータが挿入されません。これらのオプションの値を変更すると、INSERT のパフォーマンスが向上します。たとえば、ダイレクト・インサートの実行、REDO ログ・ファイルへの REDO 情報の記録の拒否、およびロックする（他のセッションではデータを挿入できない）パーティションまたはサブパーティションを指定できます。

ダイレクト・インサートの実行

ダイレクト・パス・インサートを実行すると、挿入操作のパフォーマンスが向上します。これは、Oracle のダイレクト・パス・ローダー・ユーティリティである SQL*Loader の機能に類似しています。ダイレクト・パス・インサートを指定するには、OLAP DML の SQL PREPARE INSERT コマンドの最初の挿入オプションとして DIRECT=YES を指定します。

リレーショナル表へのアナリティック・ワークスペース・データの挿入例

OLAP DML を使用して新しい製品群の導入を計画し、売上履歴データベースにこれらの新製品の製品 ID および製品名の情報を追加すると想定します。OLAP DML プログラムを使用して、アナリティック・ワークスペースから products 表にこれらの情報をコピーできます。例 10-24 に、これらのデータを含むアナリティック・ワークスペース・オブジェクトの定義を示します。

例 10-25 のプログラムでは、FOR ループを使用して、products という名前の表に現在ステータス内に存在するすべての製品の値をコピーする方法を示しています。例 10-25 のプログラムをより効率的に実行するには、INSERT 文を PREPARE 文でコンパイルします。例 10-26 では、PREPARE 文を使用して INSERT 文が write_products という名前でコンパイルされます。この write_products は、FOR ループ内の EXECUTE 文で実行されます。

例 10-24 add_newprods プログラムのアナリティック・ワークスペースの定義

```
DEFINE aw_prod_id DIMENSION NUMBER (10,0)
DEFINE aw_product_name DIMENSION TEXT
```

例 10-25 のプログラムでは、FOR ループを使用して、products という名前のリレーショナル表に、現在ステータス内に存在するすべての製品の値をコピーする方法を示しています。例 10-25 のプログラムをより効率的に実行するには、INSERT 文を PREPARE 文でコンパイ

ルします。例 10-26 では、PREPARE 文を使用して INSERT 文が write_products という名前でコンパイルされます。この write_products は、FOR ループ内の EXECUTE 文でに実行されます。例 10-27 では、PREPARE 文を使用して、INSERT 文がダイレクト・インサート用にコンパイルされます（DIRECT=YES）。

例 10-25 効率的ではない FOR ループ

```
FOR prod
   DO
      SQL INSERT INTO products -
         VALUES(:aw_prod_id, :aw_product_name)
      IF SQLCODE NE 0
         THEN BREAK
   DOEND
```

例 10-26 プリコンパイルされたコードを使用した効率の改善

```
SQL PREPARE write_products FROM -
   INSERT INTO products -
      VALUES(:aw_prod_id, :aw_product_name)
      .
      .
      .
FOR prod
   DO
      SQL EXECUTE write_products
      IF SQLCODE NE 0
         THEN BREAK
   DOEND
```

例 10-27 ダイレクト・インサートを使用した効率の改善

```
SQL PREPARE write_products FROM -
   INSERT INTO products -
      VALUES(:aw_prod_id, :aw_product_name)
      DIRECT=YES
      .
      .
      .
FOR prod
   DO
      SQL EXECUTE write_products
      IF SQLCODE NE 0
         THEN BREAK
   DOEND
```

リレーショナル表の条件付き更新

アナリティック・ワークスペースの変数の値を使用して、リレーショナル表の値を更新することもできます。OLAP DML プログラムは、FOR ループを使用して、指定されたディメンション値を値ごとにループ処理し、WHERE 句を使用して、リレーショナル表の対応する行を指します。

例 10-28 のプログラムでは、prod_id 列の値が現在ステータス内に存在する aw_prod_id ディメンション値と一致する products 表の行のみが更新されます。

例 10-28 リレーショナル表の条件付き更新

```
FOR prod
   DO
      SQL UPDATE products -
      SET product_name = :aw_newproduct_name -
         WHERE prod_id = :aw_prod_id
      IF SQLCODE NE 0
         THEN BREAK
   DOEND
```

ストアド・プロシージャおよびトリガーの使用

ストアド・プロシージャおよびトリガーがサポートされています。これらに SELECT 文を含めることはできません。アナリティック・ワークスペースのストアド・プロシージャには出力変数またはトランザクションを含めることはできず、また他のプロシージャをコールすることもできません。OLAP DML プログラムでストアド・プロシージャまたはトリガーを作成できます。例 10-29 に、new_products という名前のプロシージャを作成する OLAP DML 構文を示します。

OLAP DML 構文は、標準 SQL 構文とはわずかに異なります。OLAP DML 構文では、終了記号としてセミコロンではなくチルダ（~）が必要です。また、代入文ではコロンが 1 つではなく 2 つ（::）必要です。

例 10-29 new_products という名前のストアド・プロシージャの作成

```
SQL CREATE PROCEDURE new_products -
   (aw_id CHAR, aw_name CHAR, aw_cost NUMBER) IS -
      price number~ -
   BEGIN -
      aw_price ::= aw_cost * 2.5~ -
      INSERT INTO products -
         VALUES(aw_id, aw_name, aw_price)~ -
   END~
```

ストアド・プロシージャの実行

PROCEDURE 文を使用してストアド・プロシージャを実行するには、次の構文を使用します。

```
SQL PROCEDURE procedure-name (arg1, arg2, arg3, . . .)
```

引数にはリテラル・テキストまたは入力ホスト変数を指定できます。入力ホスト変数を使用する場合は、変数の名前の前にコロンを使用する必要があります。また、プロシージャで定義されたパラメータに対して、適切なデータ型を持つ同じ数の引数を使用する必要もあります。ストアド・プロシージャの実行時には、リテラル引数を使用できます（例 10-30 を参照）。この例では、new_products プロシージャを使用して products 表の 1 つの行が挿入されています。また、引数としてアナリティック・ワークスペース・オブジェクトを指定することもできます（例 10-31 を参照）。この例では、同じプロシージャが実行されていますが、アナリティック・ワークスペースのディメンションおよび変数に格納されたデータが products 表に挿入されています。例 10-31「ストアド・プロシージャのパラメータとしてのアナリティック・ワークスペース・オブジェクトの使用」の add-prods プログラムでは、FOR ループを使用してステータス内のすべての値をループ処理する方法を示します。add_prods をコールするには、次のようなコマンドを発行して、更新する値のみを含めるように prod のステータスを設定します。

```
CALL add_prods('last 5')
```

例 10-30 ストアド・プロシージャへのリテラル値の提供

```
SQL PROCEDURE new_products -
   ('P81', '8mm Camcorder')
```

例 10-31 ストアド・プロシージャのパラメータとしてのアナリティック・ワークスペース・オブジェクトの使用

```
DEFINE add_prods PROGRAM
LD Add new products using stored procedure new_products
PROGRAM
ARG newprods TEXT
PUSH aw_prod
LIMIT aw_prod TO &newprods

" Loop over aw_prod to insert the data
FOR aw_prod
   DO
      SQL PROCEDURE new_products(:aw_prod_id, :paw_rod_name)
   DOEND
POP aw_prod
END
```

エラーの確認

OLAP DML は、一部の SQL エラーを通知しますが、SQL 文にエラーが存在する場合は、エラーを自動的に通知しません。かわりに、OLAP DML では戻されたエラーの処理がサポートされています。

プログラムには、SQL エラーを処理するためのロジックが必要です。OLAP DML では、SQLCODE および SQLERRM の 2 つのオプションが提供されています。これらのオプションの値は、データベースに設定された SQLCODE および SQLERRM の値を反映しています。

SQLCODE オプション

SQLCODE には、整数のエラー・コード番号が含まれます。プログラムでは、すべての SQL コマンドを実行した後で SQLCODE の値をテストして、コマンドが正常に実行されたことを確認する必要があります。SQLCODE の値をテストして、ループを中断する必要があるかどうかを判断することもできます。通常、SQLCODE には表 10-3 に示すいずれかの値が含まれます。

表 10-3 SQLCODE の値

コード	意味
0（ゼロ）	最後の SQL 操作が成功しました。
100	要求されたすべての行がフェッチされました。
-1	エラーが発生しました。
0 または 100 以外のすべての値	エラーが発生しました。

SQLERRM オプション

SQLERRM オプションには、現在のエラー・コードに関連付けられたエラー・メッセージが含まれます。これによってエラーの発生原因となった条件が識別されます。このメッセージを現行の送信ファイルに自動的に送信するかどうかを制御できます。プログラムをデバッグする際は、すぐに確認できるように、すべての SQL エラー・メッセージを現行の送信ファイルに送信する場合があります。ただし、アプリケーションの使用中は、エラー・メッセージを抑止して、アプリケーションに適した方法でエラー条件を処理する必要があります。

SQLMESSAGES オプション

SQLMESSAGES オプションを使用すると、SQL メッセージを現行の送信ファイル（通常は画面）に送信するかどうかを制御できます。SQL メッセージを現行の送信ファイルに送信するには、次のコマンドを発行します。

```
SQLMESSAGES = yes
```

11

ファイルからのデータの読込み

この章では、外部ファイルからデータを読み込む方法について説明します。この章では、次の項目について説明します。

- データ読込みプログラムの概要
- ファイルの読込み
- OLAP DML でのファイル名の指定
- ファイルからのデータの読込み
- ディメンション値の読込みおよびメンテナンス
- 入力データの処理
- レコードの個別の処理
- 1 つの変数に対する複数の値の処理

データ読込みプログラムの概要

コマンドのグループ（データ読込みコマンド）をプログラムで使用すると、様々なフォーマット（バイナリ、パック 10 進数またはテキスト）の外部ファイルからデータを読み込むことができます。

一部のデータ読込みコマンドは個別に使用できますが、データ読込みコマンドは、プログラム（データ読込みプログラムという）内に配置することをお薦めします。これによって、入力ミスを最小限に抑え、より少ないデータ・セットでコマンドをテストできます。また、プログラムを使用すると、複数のコマンドを同時に使用して、ファイル内の多数のレコードをループ処理する操作を実行できます。

次の表に、データ読込みコマンドを示します。

ファンクションまたはコマンド	説明
FILEERROR ファンクション	データ読込みコマンド FILEREAD および FILEVIEW を使用して入力ファイルのレコードを処理する際に発生した最初のエラーに関する情報を戻します。
FILENEXT ファンクション	レコードを FILEVIEW コマンドで処理可能にします。レコードを読み込める場合は YES、ファイルの終わりに達した場合は NO を戻します。
FILEREAD コマンド	入力ファイルからレコードを読み込み、データを処理して、入力レコード内のフィールドの記述に従って、アナリティック・ワークスペースのディメンション、複合ディメンション、リレーションおよび変数にデータを格納します。
FILEVIEW コマンド	FILENEXT ファンクションとの併用で動作します。入力ファイルを一度に 1 レコードずつ読み込み、データを処理して、フィールドの記述に従って、アナリティック・ワークスペースのディメンションおよび変数にデータを格納します。
RECNO ファンクション	読込み用に開かれたファイルの現行のレコード番号をレポートします。

データ読込みコマンドは、次に示すようなファイル I/O コマンドとともに使用します。

ファンクションまたはコマンド	説明
FILECLOSE コマンド	開いているファイルを閉じます。
FILEGET ファンクション	読込み用に開かれたファイルのテキストを戻します。
FILEOPEN ファンクション	ファイルを開き、ファイル・ユニット番号（任意の整数）を割り当て、その番号を戻します。
FILEPUT コマンド	テキスト式に指定されているデータを、WRITE または APPEND モードで開かれたファイルに書き込みます。
FILEQUERY ファンクション	1 つ以上のファイルに関する情報を戻します。
FILESET コマンド	指定したファイル・ユニットのページング属性を設定します。

ファイルの読込み

ファイルから読込みを行う場合、各フィールドのデータを個別にフォーマットし、DML ファンクションを使用して情報を処理してから、アナリティック・ワークスペース・オブジェクトにそのデータを割り当てることができます。通常、ファイルを読み込むには、次の手順を実行します。

1. 読み込むファイルを開きます。
2. 一度に 1 レコードまたは 1 行ずつ、ファイルからデータを読み込みます。
3. データを処理し、1 つ以上のアナリティック・ワークスペース・オブジェクトに割り当てます。
4. ファイルを閉じます。

FILEREAD および FILEVIEW コマンドは、同じ属性を持ち、同じデータ処理を実行できます。ただし、次の重要な相違点が存在します。

- FILEREAD コマンドは、ファイル内のすべてのレコードを自動的にループ処理します。FILEREAD は、ファイル内のすべてのレコードが同じ場合に使用します。
- FILEVIEW コマンドは、一度に 1 レコードずつ処理します。FILEVIEW は、ファイル内に複数のタイプのレコードが存在する場合に使用します。

データを読み込むプログラムの作成

次の表に、入力ファイルの開閉、ファイルの読込み、およびエラーが発生した場合の処理に各メソッドで必要なコマンドを示します。

プログラムのセクション	FILEREAD	FILEVIEW
初期化	VARIABLE funit INTEGER TRAP ON error	VARIABLE funit INTEGER TRAP ON error
本体	funit = FILEOPEN(- 'alias/datafile' READ) FILEREAD funit . . . FILECLOSE funit	funit = FILEOPEN(- 'alias/datafile' READ) WHILE FILENEXT(funit) DO FILEVIEW funit . . . DOEND FILECLOSE funit
正常終了	RETURN	RETURN
異常終了	error: IF funit NE na THEN FILECLOSE funit	error: IF funit NE na THEN FILECLOSE funit

注意： プログラムの異常終了セクションでのエラー処理では、ファイルが開いている場合にのみファイルが閉じられます。なんらかの原因でシステムがファイルを開くことができない場合、FILEOPEN ファンクションがエラーを通知します。プログラムは、FUNIT に有効なファイル・ユニット番号が含まれる場合にのみ、ERROR ラベルの後でファイルを閉じようとします。エラー処理セクションにコマンドを追加することもできます。プログラムのこのセクションは、両方のメソッドで同じです。

OLAP DML でのファイル名の指定

FILEOPEN ファンクションを実行すると、ファイルが開かれ、そのファイルを一意に識別する整数が戻されます。このファイル識別子は**ファイル・ユニット**といいます。一度ファイルを開いてファイル・ユニットを取得すると、そのファイルに対するデータ読込みコマンドおよびファイル I/O コマンドの後続のすべてのコールでは、ファイル名ではなく、ファイル・ユニットが参照されます。

ファイル識別子は、ディスク上に格納されたファイルを指定する文字列です。ファイル識別子は、ディレクトリ別名およびファイル名を含みます。これらの 2 つの構成要素は、スラッシュ（/）で区切られます。CDA コマンドを使用すると、カレント・ディレクトリ別名を指

定できます。この場合、Oracle OLAP はファイルがカレント・ディレクトリ別名に存在すると想定するため、ファイル識別子でディレクトリ別名を指定する必要はありません。ファイルの読込みおよび書込みを実行できるディレクトリ別名に対するアクセス権については、Oracle DBA に連絡してください。

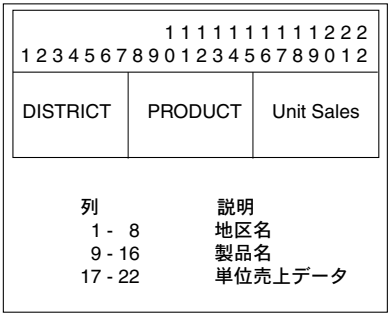
OLAP DML コマンドでファイル識別子を指定する場合、常に一重引用符で識別子を囲むことをお薦めします。これによって、ファイル名の構成要素がアナリティック・ワークスペース・オブジェクト名または予約語である場合も、解析エラーは発生しません。

ファイルからのデータの読込み

データ読込みプログラムは、ファイルから 1 レコードずつデータを読み込み、そのデータをアナリティック・ワークスペース内の変数、リレーション、ディメンションおよび複合ディメンションに割り当てます。ファイル内のレコードにディメンション値が含まれる場合、FILEREAD コマンドを使用すると、ディメンションをこれらの値に制限してから、それらの値によってディメンション化されている変数にデータを割り当てることができます。

例 11-1 データ読込みプログラムでの FILEREAD の使用

アナリティック・ワークスペース内の product ディメンションの単位売上データを更新すると想定します。新しい売上情報は、units.dat という名前のファイルに格納されています。次の図に、このファイルのレイアウトを示します。



この units.dat サンプル・ファイルを読み込む FILEREAD コマンドを次に示します。

```
FILEREAD funit -
    COLUMN 1 WIDTH 8 district -
    COLUMN 9 WIDTH 8 product -
    COLUMN 17 WIDTH 6 units
```

このコマンドは、次の 3 つの手順で処理されます。

1. 列 1 からフィールドの読込みが開始され、district ディメンションは読み込まれた値に制限されます。読み込まれた値が district のディメンション値でない場合、エラーが発生します。
2. 2 番目のフィールドが読み込まれ、product ディメンションが制限されます。
3. 3 番目のフィールドが読み込まれ、手順 1 および 2 で読み込まれた district および product に対応する units 変数のセルに、値が割り当てられます。

ファイルの開閉コマンドを含むプログラム全体を次に示します。

```
DEFINE readit1 PROGRAM
LD Read a data file
VARIABLE funit INTEGER
TRAP ON error
funit = FILEOPEN('olapfiles/units.dat' READ)
FILEREAD funit -
   COLUMN 1 WIDTH 8 district -
   COLUMN 9 WIDTH 8 product -
   COLUMN 17 WIDTH 6 units
FILECLOSE funit
RETURN
error:
IF funit NE na
   THEN FILECLOSE funit
END
```

構造化された PRN ファイルの読込み

多くの PC ソフトウェア製品によって生成される構造化 PRN ファイルも、データ読込みコマンドを使用して読み込むことができます。PRN ファイルでは、レコードのフィールドが、空白またはカンマによって区切られた引用符付きテキストまたは一連の数値で構成されます。フィールドの開始列を指定するのではなく、STRUCTURED キーワードを使用して、構造化されたファイルを読み込むように指定できます。1 つ以上の FIELD キーワードを使用して、読み込むフィールドの数を指定することもできます。

例 11-2 構造化された PRN ファイルの読込み

次に示す構造化された PRN ファイルから売上データを読み込むと想定します。

```
010195  "TENTS"   "BOSTON"    307  50808.96
010195  "TENTS"   "ATLANTA"   279  46174.92
010195  "TENTS"   "CHICAGO"   189  31279.78
010195  "TENTS"   "DALLAS"    308  50974.46
010195  "TENTS"   "DENVER"    215  35582.82
010195  "TENTS"   "SEATTLE"   276  45678.41
010195  "CANOES"  "BOSTON"    352  70489.44
```

```
010195  "CANOES"  "ATLANTA"   281  56271.40
010195  "CANOES"  "CHICAGO"   243  48661.74
010195  "CANOES"  "DALLAS"    176  35244.72
010195  "CANOES"  "DENVER"    222  44456.41
010195  "CANOES"  "SEATTLE"   335  67085.12
```

このファイルには、2 番目のフィールドに product 値、3 番目のフィールドに district 値、5 番目のフィールドに売上データが含まれています。

month ディメンションを任意の月に制限して、次のコマンドを使用すると、ファイルの最初の 6 つのレコードから売上データを読み込むことができます。

```
FILEREAD unit STOPAFTER 6 STRUCTURED FIELD 2 product -
   district FIELD 5 sales
```

ディメンション値の読込みおよびメンテナンス

データ・ファイル内のレコードには、多くの場合、データ値を格納するセルの識別に使用されるディメンション値が含まれています。ファイル内のすべてのディメンション値がアナリティック・ワークスペースにすでに存在している場合、ディメンションのフィールド記述でデフォルト属性の MATCH を使用できます。MATCH を使用すると、アナリティック・ワークスペースにすでに存在しているディメンション値のみが受け入れられます。

FILEREAD で認識されない値が検出された場合、不良データを警告するエラーが通知されます。データ読込みプログラムは、不良データをスキップして処理を続行するか、または処理を停止してデータ・ファイルの妥当性を確認することによって、エラーを処理できます。

例 11-3 既存のディメンション値のみを含むレコードの読込み

次の例では、productid ディメンションの 6 文字の値、各製品の名前および製品の販売数を含むデータ・ファイルを示しています。

```
1234AA00CHOCOLATE CHIP COOKIES        123
1099BB00OATMEAL COOKIES               145
2344CC00SUGAR COOKIES                 223
3222DD00BROWNIES                      432
5553EE00GINGER SNAP COOKIES           233
```

次のアナリティック・ワークスペース・オブジェクトが、サンプル・プログラムで使用されています。

```
DEFINE productid DIMENSION ID
DEFINE productname VARIABLE TEXT <productid>
DEFINE units.sold VARIABLE INTEGER <month productid>
```

dr.prog プログラムがファイルを読み込みます。関連する製品名を含む productid の値はすでにアナリティック・ワークスペースに含まれているため、プログラムは productid の値

を使用して、ステータスの設定および units.sold 変数の右のセルへの販売数データの割当てのみを行います。

MATCH 属性は、デフォルトではフィールド記述から省略されます。アナリティック・ワークスペースに存在しない productid の値が検出されると、プログラムは trap ラベルに分岐します。ユーザーがプログラムを中断するか（エラー名は attn）、またはデータ・ファイルを開けない場合は、プログラムは終了します。そうでない場合、プログラムはエラー・トラップをリセットし、FILEREAD に戻って次のレコードの処理を続行します。

サンプル・プログラム dr.prog の定義は次のとおりです。

```
DEFINE dr.prog PROGRAM
LD Reads a file with existing dimension values
PROGRAM
VARIABLE funit INTEGER
TRAP ON error
PUSHLEVEL 'save'
PUSH month productid
LIMIT month TO FIRST 1
funit = FILEOPEN('olapfiles/dr.dat' READ)
next:
FILEREAD funit -
   COLUMN 1 WIDTH 6 productid -
   COLUMN 39 WIDTH 3 units.sold
FILECLOSE funit
POPLEVEL 'save'
RETURN
error:
"Skip current record and continue processing
IF funit NE na and ERRORNAME NE 'ATTN'
   THEN DO
     TRAP ON error
     GOTO next
     DOEND
"Close the file
IF funit NE na
   THEN FILECLOSE funit
POPLEVEL 'save'
END
```

データ・ファイルからの新しいディメンション値の追加

データ・ファイルに既存のディメンション値と新しいディメンション値が混在している場合は、フィールド記述で APPEND 属性を使用して、アナリティック・ワークスペースに新しい値およびそのすべての関連データを追加できます。

例 11-4 データ・ファイルからの新しいディメンション値の追加

dr.prog2 プログラムの最初の FILEREAD コマンドは、APPEND を使用してアナリティック・ワークスペースに新しい productid の値を追加します。2 番目の FILEREAD コマンドに製品名を読み込むフィールドが含まれているため、新しいデータは完全に読み込まれます。

APPEND によって実行されるディメンションのメンテナンスが、データを読み込む同じ FILEREAD コマンドで実行される場合がありますが、この場合はデータ処理の効率が低下します。ディメンションのメンテナンスおよびデータの読込みがファイルの別々のパスで実行されると、データ処理の効率は向上します。

存在しない製品の値をスキップして分岐から戻る必要がないため、この場合のエラー処理は短縮されます。エラーが発生した場合、プログラムはファイルを閉じ、送信された値をリストアして終了します。

プログラム dr.prog2 の定義は次のとおりです。

```
DEFINE dr.prog2 PROGRAM
LD Reads a file with new dimension values
PROGRAM
VARIABLE funit INTEGER
TRAP ON error
PUSHLEVEL 'save'
PUSH month productid
LIMIT month TO FIRST 1
funit = FILEOPEN('olapfiles/dr.dat' READ)
FILEREAD funit COLUMN 1 APPEND WIDTH 6 productid
FILECLOSE funit
funit = FILEOPEN('olapfiles/dr.dat' READ)
FILEREAD funit -
   COLUMN 1 WIDTH 6 productid -
   COLUMN 9 WIDTH 30 productname -
   COLUMN 39 WIDTH 3 units.sold
FILECLOSE funit
POPLEVEL 'save'
RETURN
error:
IF funit NE na
   THEN FILECLOSE funit
POPLEVEL 'save'
END
```

位置でのディメンション値の読込み

ターゲット・ディメンションのデータ型が TEXT、NTEXT または ID で、ファイルの入力フィールドにディメンション値ではなくディメンションの位置番号が含まれる場合は、フィールド記述で INTEGER 変換型を指定する必要があります。変換型は、入力データをターゲット・ディメンションの値に変換する方法を指定します。

ターゲット・ディメンションを month と想定すると、次のコマンドを使用して、month のデフォルト・ステータス内の位置を表す入力値を読み込むことができます。

```
FILEREAD unit COLUMN 1 WIDTH 8 INTEGER month
```

入力フィールドに位置番号が含まれている場合は、APPEND キーワードを使用してターゲット・ディメンションに新しい値を追加することはできません。

複合ディメンションの使用

複合ディメンションは自動的にメンテナンスされます。複合ディメンションを定義および使用する方法によって、パフォーマンスが大幅に向上したり、低下する場合があります。複合ディメンションをより効率的に使用するには、アナリティック・ワークスペースの設計、特にアナリティック・ワークスペースで使用するアプリケーションについて理解します。

結合ディメンションの読込みおよびメンテナンス

アナリティック・ワークスペースに結合ディメンションが存在する場合は、FILEREAD コマンドを使用してファイルを読み込む間に、それらのディメンションのステータスを設定できます。通常、データ・ファイルのレコードには、結合ディメンションの各ベース・ディメンションに対する個別のフィールドが含まれます。たとえば、ファイルが、最初のフィールドに市場名、2 番目のフィールドに製品名を持ち、さらに売上データを含む 1 つ以上のフィールドを持つ場合があります。

例 11-5 結合ディメンションの読込みおよびメンテナンス

結合ディメンション（markprod など）によってディメンション化された変数に売上データを読み込むには、次のように FILEREAD コマンドを使用します。

```
FILEREAD funit markprod -
   = <W 8 market W 8 product> W 10 sales
```

このコマンドによって、レコードの最初の 8 文字のフィールドから market ディメンションの値、次の 8 文字のフィールドから product ディメンションの値が読み込まれます。

コマンドは、この結果を使用して markprod のステータスを設定します。この連結ディメンションの定義は次のとおりです。

```
DEFINE markprod DIMENSION <market product>
```

コマンドは最後のフィールドを読み込み、markprod によってディメンション化された sales 変数に値を割り当てます。

フィールド記述に APPEND キーワードを含めると、FILEREAD コマンドで既存のディメンション値と一致しないファイルの値が検出された場合に、market、product および markprod に新しい値を追加できます。

```
FILEREAD funit APPEND markprod -
   = <W 8 APPEND market W 8 APPEND product> W 10 sales
```

コード化されたディメンション値の変換

データ・ファイルのディメンション情報を含むフィールドが、アナリティック・ワークスペースのディメンション値と異なる値を持つ場合があります。これは、ファイルの値が略されるか、またはエンコードされている場合です。コード化されたディメンション値を変換する方法は、コードが単なる略称か（product に対する P など）、またはコードがより複雑かどうかによって異なります。

ファイルに略称コードが含まれる際は、RSET または LSET 属性を使用してファイル内の値の右または左にテキストを追加することで、値を完全にできる場合があります。

たとえば、ファイルで製品が数値のみの製品番号で識別されているのに対し、アナリティック・ワークスペースでは、product ディメンションの値が P で始まる同じ製品番号である場合があります。この場合、LSET 属性を使用して、ファイル内の値に P という文字を追加できます。

```
FILEREAD funit COLUMN 1 WIDTH 6 LSET 'P' product
```

文字 P は、値がファイルから読み込まれる際に追加されます。変更された値が product ディメンションに一致または割り当てられる際には追加されません。

より複雑なコードを持つ値を正確に読み込むには、データ・ファイルに存在するコード化された値を含む別のディメンションを、実際のディメンションへのリレーションとともに設定します。FILEREAD は、そのリレーションを使用して実際のディメンション値を判断できます。または、OLAP DML ファンクションを使用して、コード化された値を変更または操作し、アナリティック・ワークスペースの値に一致させることもできます。

ターゲットに格納する前になんらかの方法で操作する必要があるコード化されたデータを読み込む場合は、次のようにフィールド記述で代入文を使用します。

```
target = expression
```

expression 引数には、実行する処理または計算を指定します。ファイルから読み込んだ直後の値を *expression* の一部として含めるには、VALUE キーワードを使用します。

次の両方のフィールド記述は同様に機能します。

```
COLUMN n WIDTH n target

target = COLUMN n WIDTH n VALUE
```

例 11-6 ディメンション値へのコードの変換

この例では、コードをディメンション値に変換する式の使用方法を示します。

次の例では、月を表す 3 文字のコード、および地区と製品を表す 2 文字のコードを持つデータ・ファイルを示しています。

```
SEP BO CH 113945 115
OCT BO CH 118934 115
SEP BO CO  92013 119
OCT BO CO  95820 119
SEP BO WI  83201 110
OCT BO WI  82986 110
SEP DA CH 111792 115
OCT DA CH 136031 114
SEP DA CO  91641 121
OCT DA CO  96347 120
SEP DA WI  89734 109
OCT DA WI  88264 109
```

次の OLAP DML オブジェクトがサンプル・プログラムで使用されています。

```
DEFINE distcode DIMENSION ID
DEFINE district.dcode RELATION district <distcode>
DEFINE prodcode DIMENSION ID
DEFINE Product.pcode RELATION product <prodcode>
```

サンプル・プログラム dr.prog3 の定義は次のとおりです。

```
DEFINE dr.prog3 PROGRAM
LD Translates coded values into valid dimension values
PROGRAM
VARIABLE funit INT
funit = FILEOPEN('olapfiles/dr3.dat' READ)
FILEREAD funit -
   COLUMN 1 WIDTH 3 APPEND RSET '96' month
FILECLOSE funit
funit = FILEOPEN('olapfiles/dr3.dat' READ)
FILEREAD funit -
   COLUMN 1 WIDTH 3 RSET '96' month -
   COLUMN 5 WIDTH 2 district = district.dcode -
     (distcode VALUE) -
   COLUMN 8 WIDTH 2 product = product.pcode -
     (prodcode VALUE) -
   COLUMN 11 WIDTH 6 STRIP units -
   COLUMN 18 WIDTH 3 SCALE 2 price
FILECLOSE funit
END
```

このプログラムは、地区および製品を表す 2 文字のコードを district ディメンションおよび product ディメンションの値に変換します。また、2 桁の年を月に追加します。

最初の FILEREAD コマンドが、APPEND キーワードを使用して新しい月を MONTH ディメンションに追加します。

```
FILEREAD fileunit COLUMN 1 WIDTH 3 APPEND RSET '96' month
```

地区および製品フィールドに対しては、データ・ファイルから値を読み込み、district.dcode および product.pcode リレーションを使用して、対応するディメンション値を検出します。

```
COLUMN 5 WIDTH 2 district = district.dcode distcode VALUE)
COLUMN 8 WIDTH 2 product = product.pcode (prodcode VALUE)
```

プログラムは、データ・ファイルから読み込むコードを表す VALUE キーワードとともに QDR を使用します。地区に対しては、distcode VALUE QDR が地区名を保持する district.dcode リレーションを変更します。データ・ファイルから読み込まれた直後の distcode の値に対応する地区を指定します。product に対しても、QDR は同様に動作します。

プログラムは、product および district ディメンション値が、distcode と prodcode ディメンション、およびそれらを district と product に関連付けるリレーションとともに、アナリティック・ワークスペースにすでに存在していると想定しています。コード化された値が処理されると、district および product の結果の値を使用してディメンションのステータスが制限され、units および price 変数の右のセルにデータが挿入されます。

最後に、データ・ファイルでは、列 18 から始まる価格データに小数点が付いていません。FILEREAD コマンドの最後の行の SCALE 属性によって、価格の各値に 2 桁の小数位が付けられます。

入力データの処理

= コマンドを使用して作成する代入文は、データ読込みコマンドで様々な用途に使用できます。 = コマンドを使用すると、ファイルから読み込む値を様々な方法で処理できます。 読み込んだ値を単に変数またはリレーションに割り当てるかわりに、これらの値をアプリケーションにより適したものに変更できます。

必要に応じて、単純な式を使用したり、複雑な式を使用することができます。アナリティック・ワークスペース内にすでに格納されている他のデータ、またはファイルから読込み済のデータに基づいて、読み込んだ値の条件分岐処理を実行することもできます。

フィールド記述で代入文を使用した FILEREAD コマンドの使用例については、11-7 ページの「ディメンション値の読込みおよびメンテナンス」を参照してください。

例 11-7 ファイルから読み込んだ値の変更

次のコマンドを実行すると、売上データを読み込み、sales 変数にそのデータを割り当てて、その変数にすでに格納されている値を置換できます。

```
FILEREAD funit W 8 district W 8 product W 10 sales
```

ただし、式を使用すると、変数に現在格納されている値に新しいデータを追加できます。

```
FILEREAD funit W 8 district W 8 product sales -
  = sales + W 10 VALUE
```

ファイルから読み込んだ直後のデータは、式では VALUE キーワードによって表されます。

ファイル内に 2 つの異なるレコード型が存在する場合は、各レコード型の異なるフィールドを読み込むことができます。

```
FILEREAD funit W 1 rectype W 8 district W 8 -
   APPEND product -
   prodname = -
     IF rectype EQ 'A' THEN COL 25 W 16 VALUE -
     ELSE COL 42 W 16 VALUE
```

データの変換型の指定

通常、アナリティック・ワークスペースの変数への入力値を読み込む際に、データ型を指定する必要はありません。デフォルトでは、入力値はターゲット変数のデータ型に変換されます。

ただし、ターゲット変数のデータ型が DATE である場合は、デフォルトの DATE 変換型か、または代替の RAW DATE 変換型のいずれかを使用できます。

ターゲット変数に格納する前に式を使用して入力値を処理する場合も、変換型を指定できます。

レコードの個別の処理

データ・ファイル内のすべてのレコードが、常に同じデータ型を持つとはかぎりません。レコードごとに異なるフィールド記述や異なるターゲット・オブジェクトが必要になる場合、または 1 つのファイル内に複数の異なるレコード型が混在する場合があります。1 つ以上のフィールドの内容に基づいて、レコードのデータの処理方法を決定する必要がある場合もあります。

FILENEXT ファンクションおよび FILEVIEW コマンドを使用すると、ファイルから一度に 1 つのレコードを取得し、そのデータを何度でも参照できます。FILENEXT は、データ・ファイルからレコードを読み込むブール・ファンクションです。レコードを検出した場合は YES、ファイルの終わりに達した場合は NO を戻します。FILENEXT によって読み込まれたレコードは、FILEVIEW コマンドを使用して処理できます。

通常、FILENEXT は WHILE コマンドの条件として使用されるため、データ読込みプログラムは、ファイルの終わりに達してそれ以上レコードを検出しなくなるまで読込みを続行します。WHILE ループ内で、FILEVIEW コマンドは現行レコードの任意のフィールドからデータを処理するために 1 回以上使用されます。多くの場合、FILEVIEW コマンドの操作は、WHILE ループ内の前のコマンドで処理されたデータによって異なります。

例 11-8 同じレコードからの異なるデータの読込み

次のデータでは、各レコードの 2 番目のフィールドに、最後のフィールドのデータに対するターゲット変数の名前が含まれています。

```
CEREALS     DOL         VS100       US    JUN96    5000000
CEREALS     LBS         VS100       US    JUN96    4800000
CEREALS     CASE        VS100       US    JUN96     180000
CEREALS     DOL         VS100       BOS   JUN96      62500
CEREALS     LBS         VS100       BOS   JUN96      62830
CEREALS     CASES       VS100       BOS   JUN96       2750
CEREALS     DOL         VS100       CHI   JUN96      75290
CEREALS     LBS         VS100       CHI   JUN96      73000
CEREALS     CASES       VS100       CHI   JUN96       2700
CEREALS     DOL         VS100       LASF  JUN96     143070
CEREALS     LBS         VS100       LASF  JUN96     150500
CEREALS     CASES       VS100       LASF  JUN96         NA
```

次の OLAP DML オブジェクトがサンプル・プログラムで使用されています。

```
DEFINE dol VARIABLE DECIMAL <month item market>
DEFINE lbs VARIABLE INTEGER <month item market>
DEFINE cases VARIABLE INTEGER <month item market>
```

dr.prog4 プログラムは、値を取得する前に基準に対してレコードをテストします。このプログラムでは、最初の FILEVIEW コマンドによって変数の名前が取得され、varname という名前のローカル変数にその名前が格納されます。2 番目の FILEVIEW コマンドによって値が取得され、varname で指定したオブジェクトにその値が割り当てられます。

サンプル・プログラム dr.prog4 には、次のコードが含まれます。

```
VARIABLE funit INTEGER
VARIABLE varname TEXT
funit = FILEOPEN('olapfiles/dr4.dat' READ)
WHILE FILENEXT(funit)
  DO
    FILEVIEW funit COLUMN 13 WIDTH 12 varname
    FILEVIEW funit COLUMN 25 WIDTH 12 item -
                   COLUMN 37 WIDTH  6 market -
                   COLUMN 43 WIDTH  5 month -
                   COLUMN 48 WIDTH 10 &varname
  DOEND
FILECLOSE funit
```

異なるレコードの読込み

レコード自体のいくつかの基準に基づいて、ファイル内の一部のレコードのみを処理する場合があります。1 つの FILEVIEW コマンドを使用して該当する値のフィールドを確認し、そのフィールドが検出された場合は、2 番目の FILEVIEW コマンドを使用して残りのレコードを処理できます。

レコードが処理の基準を満たしていない場合は、FILEPUT コマンドを使用して別のファイルに保存できます。FILEPUT で FROM キーワードを使用すると、FILENEXT で読み込んだ最後のレコードが指定した出力ファイルに直接書き込まれます。プログラムのエラー・セクションで FILEPUT コマンドを使用すると、エラーのために処理できなかったレコードを追跡することもできます。

データ読込みプログラムで FILEPUT を使用する前に、2 番目のファイルを書込みモードで開いておく必要があります。プログラムの最後にそのファイルを閉じる必要があります。

1 つの変数に対する複数の値の処理

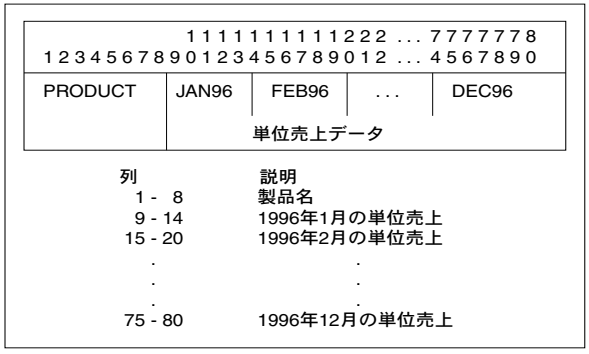
ファイル内の連続する複数のフィールドに、同じ変数に割り当てるデータ値が含まれる場合があります。各フィールドは、ターゲット変数の 1 つのディメンションの異なる値に対応します。

繰返しのフィールドでは、フィールド記述で ACROSS 句を使用して連続するフィールドを読み込み、ターゲット変数の適切なセルに値を挿入できます。ACROSS 句を使用すると、現行のステータス内またはレコードの終わりに達するまで、各ディメンション値のデータが抽出されます。FILEREAD（または FILEVIEW）コマンドを実行する前に ACROSS ディメンションを制限するか、または ACROSS 句で一時的に制限することもできます。

ACROSS ディメンションの制限に必要な情報がデータ・ファイルに含まれている場合は、一時変数を使用してディメンション値を抽出し、ディメンションを制限してから、ファイルの残りを読み込みます。

例 11-9 同じ変数への複数のフィールドの割当て

連続するフィールドが連続する月の売上データを保持していると想定します（次の図に示す unitsale.dat のレイアウトを参照）。



unitsale.dat ファイルでは、列 9 ～ 80 に 6 文字の 12 のフィールドが存在します。各フィールドには、1996 年の月ごとの売上データが含まれています。

ファイルの開閉コマンドを含むデータ読込みプログラム全体を次に示します。

```
DEFINE dr.prog5 PROGRAM
LD Read a data file
VARIABLE funit INTEGER
TRAP ON error
funit = FILEOPEN('olapfiles/unitsale.dat' READ)
FILEREAD funit -
   COLUMN 1 WIDTH 8 product -
      ACROSS month jan96 TO dec96: WIDTH 6 units
FILECLOSE funit
RETURN
error:
IF funit NE na
   THEN FILECLOSE funit
END
```

ACROSS 句によって、これらの各フィールドが units 変数の別々のセルに読み込まれます。

```
ACROSS month jan96 TO dec96: WIDTH 6 units
```

FILEREAD コマンドによって、サンプル・ファイル unitsale.dat が読み込まれます。

```
FILEREAD funit -
   COLUMN 1 WIDTH 8 product -
      ACROSS month jan96 TO dec96: WIDTH 6 units
```

このコマンドではまず、列 1 からフィールドの読込みが開始され、product ディメンションが読み込まれた値に制限されます。（読み込まれた値が product のディメンション値でな

い場合、エラーが発生します。) その後、次の 12 個のフィールドが読み込まれ、読み込まれた値が 1996 年の各月の units 変数に割り当てられます。

例 11-10 入力データを使用した ACROSS ディメンションの制限

次の例に示すとおり、データ・ファイル内の最初のレコードには、データの各列のラベルとして month の値が含まれています。

```
                   JAN96      FEB96      MAR96      APR96
TENT            50,808.96  34,641.59  45,742.21  61,436.19
CANOES          70,489.44  82,237.68  97,622.28 134,265.60
RACQUETS        56,337.84  60,421.50  62,921.70  74,005.92
SPORTSWEAR      57,079.10  63,121.50  67,005.90  72,077.20
FOOTWEAR        95,986.32 101,115.36 103,679.88 115,220.22
```

次のアナリティック・ワークスペース・オブジェクトが、サンプル・プログラムで使用されています。

```
DEFINE enum DIMENSION INTEGER
DEFINE monthname VARIABLE ID <enum> TEMPORARY
DEFINE salesdata VARIABLE DECIMAL <month product>
```

サンプル・プログラム dr.prog6 の定義は次のとおりです。

```
DEFINE dr.prog6 PROGRAM
PROGRAM
VARIABLE funit INTEGER
TRAP ON cleanup
PUSHLEVEL 'save'
PUSH month product
funit = FILEOPEN('olapfiles/dr6.dat' READ)
IF FILENEXT(funit)
   THEN FILEVIEW funit COLUMN 16 ACROSS enum: -
     W 11 monthname
LIMIT month TO CHARLIST(monthname)
FILEREAD funit W 15 product COLUMN 16 ACROSS month: -
   W 11 salesdata
cleanup:
FILECLOSE funit
POPLEVEL 'save'
END
```

このプログラムは、ファイルに含まれる月数を認識していません。プログラムは、INTEGER ディメンションによってディメンション化された一時変数を使用して、ファイルから月名を読み込みます。INTEGER ディメンション enum には、月を含む最大のデータ・ファイルと同じ数以上の値が含まれる必要があります。

FILENEXT によって最初のレコードのみが読み込まれます。CHARLIST ファンクションは、month ディメンションの制限に使用される月名のリストを作成します。

最後に、FILEREAD コマンドが、ACROSS ディメンションとして month を使用して残りのレコードを処理します。ユーザーが指定しなくても、すべての売上データが正しい月に割り当てられます。

12

データの集計

この章では、OLAP DML の集計機能を使用する方法について説明します。この章では、次の項目について説明します。

- ディテール・データの集計
- 集計の前の準備手順
- 集計マップの作成
- RELATION コマンドの概要
- 非階層データの集計
- 事前計算されるデータを生成する方法
- 実行時にデータを計算する方法
- カスタム集計の作成
- 事前計算される集計と実行時の集計のバランス
- 部分集計の実行
- 予測およびプログラムと AGGREGATE の組合せ

ディテール・データの集計

一般に、ビジネス分析アプリケーションでは、データ用に階層ディメンションが使用されます。Oracle OLAP では、1 つの階層のすべてのメンバーは、レベルに関係なく、1 つのディメンションに格納されます。セルフ・リレーションおよび親リレーションが、メンバー間の親子関係を識別します。他の非階層ディメンション（明細項目ディメンションなど）では、値の計算にモデルが必要な場合があります。

通常、ディテール・レベルのデータは他のソース（トランザクション・データベースやフラット・ファイルなど）から取得しますが、集計データは計算する必要があります。この計算は、次の 2 つの異なる方法で実行できます。

- データのメンテナンス処理として行う。DBA がディテール・データを取得し、集計値を計算します。集計値は、すべてのユーザーが共有できるように、アナリティック・ワークスペースに格納されます。このような集計データは、**事前計算**集計または**格納**集計と呼ばれる場合があります。問合せ時間は最短になりますが、アナリティック・ワークスペースのサイズが増加するため、リレーショナル・データベースのサイズも増加します。事前計算データの量は、データ・タスクを実行できる時間（**バッチ・ウィンドウ**ともいう）によって制限される場合もあります。
- 必要に応じて実行時に行う。集計値のセルは、問合せで集計値が要求されるまでは NA（空）です。その後、問合せに応じて集計が計算されます。結果は、そのセッション中に使用するために一時キャッシュに格納できます。セッションでアナリティック・ワークスペースへの書込み権限を持っている場合、結果を永続的に格納することもできます。このような集計データは、**即時**集計または**実行時**集計といいます。データの取得のみでなく計算も必要になるため、問合せ時間は長くなります。ただし、集計値用の永続記憶域は必要ありません。

Oracle OLAP では両方のタイプの集計がサポートされており、1 つのデータ変数内で、一部の値を事前計算し、他の値を実行時に計算するメカニズムが提供されています。

> **参照：** 階層ディメンションの詳細は、3-21 ページの「階層ディメンションおよび階層ディメンションを使用する変数の定義」を参照してください。

AGGREGATE で使用可能な機能

OLAP DML では、最初、最後、平均、加重平均および合計を含む様々な集計メソッドがサポートされています。多次元変数では、ディメンションごとに異なる集計メソッドを使用することもできます。

変数が詳細な複数レベルの階層でディメンション化されている場合、集計データのセル数は、ディテール・データのセル数の数倍になる場合があります。これに対して、通常、ユーザーは、データの一部のレベルを頻繁に問い合せ、他のレベルにはあまり頻繁に問い合せません。中間レベルの集計が集計データで最大の割合を占めるにもかかわらず、ユーザーによる問合せは最上位レベルの集計に集中し、中間レベルの集計までドリルすることはまれである傾向があります。

このため、OLAP DML では、一部のデータを集計および格納し、他のデータを実行時に集計するための集計メソッドが提供されています。DBA は、レベル、個々のメンバー、メンバー属性、時間範囲、データ値などの基準に基づいて、適切なメソッドを選択できます。スキップ・レベル集計という方法では、ディメンション階層の 1 レベルおきに事前集計が行われます。詳細は、12-22 ページの「スキップ・レベル方法を使用したデータの計算」を参照してください。

表 12-1 に、集計をサポートする OLAP DML のコマンドを示します。

表 12-1 集計をサポートするコマンド

コマンド	説明
AGGREGATE コマンド	アナリティック・ワークスペースに永続的に格納するデータを計算します。
AGGREGATE ファンクション	問合せに応じてその場でデータを計算します。
AGGMAP コマンド	AGGREGATE コマンドで計算する集計および AGGREGATE ファンクションで計算する集計を識別する内容を aggmap オブジェクトに追加します。また、セッション中に使用するために実行時集計をキャッシュするかどうかを指定します。これによって、ディテール値に対する実行時の変更が集計値に反映されるかどうかが決まります。
AGGMAPINFO コマンド	集計マップ・オブジェクトの内容に関する情報（集計またはアロケーション用のコマンドが含まれているかどうかなど）を戻します。
CLEAR コマンド	集計キャッシュ内のデータ値を消去します。
MULTIPATHHIER オプション	複数のパス上でのディテール・データの集計を許可するかどうかを制御します。
POUTFILEUNIT オプション	AGGREGATE コマンドの処理過程に関する情報を受け取る位置を識別します。
SESSCACHE オプション	セッション中に集計キャッシュを保持するかどうかを制御します。
VARCACHE オプション	即時集計を格納する方法を制御します。

プロセスの概要 : 集計

集計データを生成および管理するには、次の基本手順を実行します。

1. データの初期分析を実行し、データが適切に設定されていることを確認します（12-4 ページの「集計の前の準備手順」を参照）。
2. 事前計算するデータおよび必要に応じて計算するデータを識別する集計マップを作成します。ディメンション化の方法が同じ変数を識別します。これらの変数は、1 つの集計マップを共有できます。

3. POUTFILEUNIT オプションを設定し、集計の処理過程を監視できるようにします。
4. 集計マップで AGGREGATE コマンドを使用し、データの事前計算およびデータベースへの格納を行います。
5. 集計マップで実行時の計算が指定されている場合、次の手順を実行します。
   a. 集計マップをコンパイルします。
   b. 実行時のデータ要求に応じて AGGREGATE ファンクションをトリガーするプロパティを変数に追加します。

これらの手順の詳細は、後続の項を参照してください。

集計の前の準備手順

集計のパフォーマンスを最適化するには、集計前に次のようないくつかの手順を実行します。

- DML で定義されているセルフ・リレーションの名前、または親と階層リレーションの名前を取得します。
- すべての複合ディメンションを調べて、BTREE 索引が存在することを確認します。

親リレーションおよびレベル・リレーションの識別

すべての集計マップでは、集計する各ディメンションの**親リレーション**を識別することが必要です。親リレーションは、各ディメンション値の親を識別することによって階層を定義するセルフ・リレーションです。

一部のデータを実行時に集計する場合、**事前計算をスキップするレベルを区別するために、レベル・リレーション**を使用する場合があります。レベル・リレーションは、各ディメンション値の階層のレベルを識別します。このリレーションは、事前計算するレベルおよび実行時に計算するレベルを識別するために必要です。レベル・リレーションを使用する方法であるスキップ・レベル集計を使用することをお薦めします（12-21 ページの「事前計算される集計と実行時の集計のバランス」を参照）。

例 12-1 に、親リレーションおよびレベル・リレーションを示します。

OBJ ファンクションを使用して、アナリティック・ワークスペース・オブジェクトに関する情報を取得することもできます。たとえば、次のコマンドを実行すると、geography ディメンションのレベル・ディメンションの名前が表示される場合があります。

```
REPORT OBJ(PROPERTY 'leveldim' 'geography')
```

注意： これらのリレーションを作成した方法によって、PROPERTY キーワードを介してこの情報を取得できるかどうかが決まります。

OBJ ファンクションによって結果が生成されない場合、アナリティック・ワークスペース内の変数の内容を参照して、これらのリレーションが存在するかどうかを確認する必要があります。リレーションが存在しない場合、作成します。

例 12-1 親リレーションおよびレベル・リレーションの識別

3 つのディメンションおよび 2 つのリレーションのオブジェクト定義を次に示します。これらのオブジェクトは、集計マップが geography によってディメンション化されたデータを集計する必要があるという情報を提供します。

```
DEFINE GEOGRAPHY DIMENSION TEXT WIDTH 12
LD Geography dimension values

DEFINE GEOGRAPHY.HIERARCHIES DIMENSION TEXT
LD Hierarchy dimension for Geography

DEFINE GEOGRAPHY.LEVELDIM DIMENSION TEXT
LD List of hierarchy levels for GEOGRAPHY

DEFINE GEOGRAPHY.PARENTREL RELATION GEOGRAPHY <GEOGRAPHY GEOGRAPHY.HIERARCHIES>
LD Parent-child relation for Geography

DEFINE GEOGRAPHY.LEVELREL RELATION GEOGRAPHY.LEVELDIM <GEOGRAPHY GEOGRAPHY.HIERARCHIES>
LD Level of each member in each Geography hierarchy
```

geography ディメンションでは、WORLD、AMERICAS、CANADA、TORONTO、MONTREAL、NEWYORK、CHICAGO、SEATTLE、MEXICO など、階層のすべてのレベルに値が含まれます。

geography.hierarchies ディメンションは、階層の名前を識別します。たとえば、geography は、STANDARD および CONSOLIDATED の 2 つの階層を持ちます。

geography.leveldim ディメンションは、レベルの名前（CITY、STATE、COUNTRY、REGION、WORLD など）を識別します。

geography.parentrel リレーションは、セルフ・リレーションです。これは、各階層および各ディメンション値の親値を識別します。たとえば、STANDARD 階層で、KYOTO の親は JAPAN、JAPAN の親は ASIA です。

geography.levelrel リレーションは、各階層内の各ディメンション値のレベルを識別します。たとえば、STANDARD 階層で、KYOTO は CITY レベル、JAPAN は COUNTRY レベル、ASIA は REGION レベルです。

すべての複合ディメンションで BTREE 索引が使用されているかどうかの確認

すべての変数の複合ディメンションが BTREE 索引アルゴリズムを使用している場合、AGGREGATE のパフォーマンスが最適化されます。DESCRIBE コマンドを使用すると、複合ディメンションが BTREE または HASH のどちらを使用しているかを判別できます。複合ディメンションが HASH を使用している場合、複合ディメンションの定義に HASH が表示

されます。複合ディメンションが BTREE を使用している場合、複合ディメンションの定義に索引アルゴリズムは表示されません。これは、BTREE が複合ディメンションのデフォルトのアルゴリズムであるためです。

market.prod 複合ディメンションの次のオブジェクト定義では、HASH 索引が使用されていることが示されています。

```
DEFINE MARKET.PROD COMPOSITE <MARKET PRODUCT> HASH
```

BTREE 索引に変更するには、次の CHGDFN コマンドを使用します。

```
CHGDFN market.prod BTREE
```

BTREE 索引が使用されている場合、複合ディメンション定義は次のようになります。

```
DEFINE MARKET.PROD COMPOSITE <MARKET PRODUCT>
```

集計マップの作成

集計マップは、アナリティック・ワークスペース・オブジェクトです。モデルまたはプログラムの作成と同様に、最初にオブジェクトを定義し、次にその内容を追加します。集計マップの内容は、変数定義内の各ディメンションで集計する必要があるデータを指定するコマンドです。また、事前計算するデータおよびその場で計算するデータを識別します。そのため、AGGREGATE コマンドと AGGREGATE ファンクションの両方で集計マップが必要です。

集計マップを作成するには、次の手順を実行します。

1. aggmap オブジェクトを定義します。
2. aggmap オブジェクトに内容を追加します。

aggmap オブジェクトを定義する方法

集計マップを定義するには、DEFINE AGGMAP コマンドを使用します。DEFINE AGGMAP コマンドの構文は次のとおりです。

```
DEFINE name AGGMAP
```

name は集計マップの名前です。

aggmap オブジェクトに内容を追加する方法

aggmap オブジェクトの定義後、そのオブジェクトに内容を追加する必要があります。次の方法を使用して、集計マップを編集できます。詳細は、編集方法の次に示す例を参照してください。

- AGGMAP コマンドを使用して、集計マップの内容を入力または置換します。
- OLAP Worksheet の EDIT AGGMAP コマンドを使用します。

- 集計マップの内容を含むテキスト・ファイルを作成し、次に INFILE コマンドを使用してそのファイルをアナリティック・ワークスペースに読み込みます。

例 12-2 AGGMAP コマンドの使用

次のプログラムでは、AGGMAP コマンドとともに JOINLINES ファンクションを使用して、aggmap オブジェクトに RELATION コマンドが追加されます。

```
DEFINE AGGTEST PROGRAM
LD Create an aggregation map
PROGRAM
IF NOT EXISTS('test.agg')
     THEN DEFINE test.agg AGGMAP
     ELSE CONSIDER test.agg
AGGMAP JOINLINES(-
     'RELATION geography.parentrel' -
     'RELATION product.parentrel' -
     'RELATION channel.parentrel' -
     'RELATION time.parentrel' -
     'END')
END
```

例 12-3 OLAP Worksheet の EDIT AGGMAP コマンドの使用

OLAP Worksheet の EDIT コマンドを使用して aggmap オブジェクトを編集するには、次の手順を実行します。

1. 次の DML コマンドを発行します。myaggmap は既存の aggmap オブジェクトの名前です。

   ```
   EDIT AGGMAP myaggmap
   ```

   AGGMAP の「Edit」ウィンドウが表示されます。
2. 集計マップの本体を入力します。または、既存の集計マップに任意の変更を加えます。
3. 「File」メニューの「**Save**」を選択して、変更を保存します。
4. 「File」メニューの「**Quit**」を選択して、「Edit」ウィンドウを閉じます。

例 12-4 INFILE コマンドを使用したテキスト・ファイル内のコマンドの実行

集計マップの内容を含むテキスト・ファイルを作成できます。このテキスト・ファイルを使用して、集計マップを作成または変更できます。

gpct.aggmap という名前の aggmap オブジェクトを定義したと想定します。次の内容を含むファイルを作成できます。

```
CONSIDER gpct.aggmap
AGGMAP
```

```
RELATION geography.parentrel
RELATION product.parentrel
RELATION channel.parentrel
RELATION time.parentrel
END
```

このファイルの名前が aggmap.inf で、ディレクトリ別名 userfiles 内に存在する場合、次の INFILE コマンドを使用すると、セッション中にこれらのコマンドを実行できます。

```
INFILE 'userfiles/aggmap.inf'
```

集計マップの内容

集計マップには、次のコマンドが含まれます。

- AGGMAP コマンド。集計マップの始まりを識別します。aggmap オブジェクトに内容を追加する方法によっては、このコマンドを明示的に含める必要がない場合があります。
- RELATION コマンド。データを集計するために使用される、ディメンションの（階層として機能する）親リレーションまたはセルフ・リレーションを識別します。また、集計のタイプおよび集計するデータの選択を識別することもできます。デフォルトでは、すべてのデータが合計されます。すべての集計マップには、1 つ以上の RELATION コマンドが含まれます。
- MODEL コマンド。事前定義された MODEL オブジェクトを実行します。モデルを使用すると、親リレーションを持たない、非階層ディメンション上のデータを集計できます。
- CACHE コマンド。その場で計算されるデータを AGGREGATE ファンクションで格納するかどうか、またはその格納方法を記述します。これによって、変数のデータに対する実行時の変更がその変数のすべてのデータに反映される速度が制御されます。
- AGGINDEX コマンド。Oracle OLAP で、MODEL コマンドおよび ACROSS 句を使用するコマンドで必要な索引（複合タプル）を作成する必要があるかどうかを記述します。これは、変数が複合ディメンションを持つ場合にのみ使用します。
- END コマンド。集計マップの終わりを識別します。aggmap オブジェクトに内容を追加する方法によっては、このコマンドを明示的に含める必要がない場合があります。

注意： CACHE コマンドと AGGINDEX コマンドの両方には、デフォルト設定が存在します。これらのデフォルト設定がアプリケーションに適している場合、これらのコマンドを集計マップから省略できます。Oracle9*i* OLAP DML Reference ヘルプで各コマンドの詳細を参照し、これらのコマンドを使用する必要があるかどうかを判断してください。

例 12-5 単純な集計マップ

SUM 演算子を使用してすべてのディメンション上のデータが事前計算される単純な集計マップを次に示します。集計マップの本体が AGGMAP コマンドで始まり、END コマンドで終わっていることに注意してください。RELATION コマンドは、aggmap オブジェクトの定義に示されているディメンションの順序に従って示されます。

```
DEFINE GPCT.AGGMAP AGGMAP
LD Aggregation map for sales, units, quota, costs
AGGMAP
RELATION geography.parentrel
RELATION product.parentrel
RELATION geography.parentrel
RELATION time.parentrel
END
```

集計マップをコンパイルする方法

作成した集計マップは、コンパイルして保存する必要があります。この手順は、AGGREGATE ファンクションを使用して実行時に行う集計では重要です。コンパイルしたバージョンの集計マップを保存していない場合、その集計マップを使用する各セッションによって、集計マップが再コンパイルされます。

AGGREGATE コマンドで FUNCDATA 引数を使用すると、集計マップは自動的にコンパイルされます。たとえば、次のコマンドを実行すると、集計データが事前計算され、実行時の集計用に、コンパイルした集計マップのコピーが保存されます。

```
AGGREGATE sales USING gpct.aggmap FUNCDATA
UPDATE
COMMIT
```

別の方法として、COMPILE コマンドを使用して集計マップを明示的にコンパイルすることもできます。集計マップの明示的なコンパイルは、集計マップを使用してデータの生成を試行する前に集計マップ内の構文エラーを検出するためにも有効です。

次のコマンドを実行すると、コンパイルしたバージョンの sales.agg 集計マップが作成および保存されます。

```
COMPILE gpct.aggmap
UPDATE
COMMIT
```

重要： 一部のデータをその場で計算する場合、AGGREGATE コマンドを実行した後で、集計マップをコンパイルおよび保存する必要があります。

集計マップのコンパイルは長時間かかる場合があります。集計マップのコンパイルが正常に実行されていない場合、その場で計算を実行するために必要な情報を取得するために、

AGGREGATE ファンクションが集計マップを自動的にコンパイルします。この場合、問合せのパフォーマンスは低下します。ユーザーがアナリティック・ワークスペースに初めて問い合せるたびに、AGGREGATE ファンクションがデータの計算前に集計マップをコンパイルする必要があります。100 人のユーザーが同じアナリティック・ワークスペースに問い合せると、集計マップは 100 回コンパイルされます。集計マップをプリコンパイルしてアナリティック・ワークスペースに保存しておくと、集計マップは、構築プロセスで 1 度コンパイルされるのみです。ユーザーの問合せの結果としてコンパイルが行われるようにすると、集計マップは、各ユーザーに対して繰り返しコンパイルされます。

1 つのコマンドを使用した複数の変数の集計

次の条件を満たす場合、1 つの AGGREGATE コマンドを使用して、複数の変数のデータを集計できます。

- すべての変数が同じ次元性を持っている。これは、各変数定義に、同じディメンションが同じ順序で存在することを意味します。
- すべての変数に対して同じ集計マップを使用できる。これは、各変数で同じレベルのデータが事前計算されることを意味します。そのため、ユーザーが、各変数において同じレベルのデータに問合せを行う傾向があることを確認する必要があります。

例 12-6 1 つのコマンドを使用して集計可能な変数

アナリティック・ワークスペースに次の名前付き複合ディメンションおよび変数の定義が含まれると想定します。

```
DEFINE PROD.GEOG.CHAN COMPOSITE <PRODUCT GEOGRAPHY CHANNEL>

DEFINE SALES DECIMAL <TIME PROD.GEOG.CHAN <PRODUCT GEOGRAPHY CHANNEL>>
DEFINE UNITS INTEGER <TIME PROD.GEOG.CHAN <PRODUCT GEOGRAPHY CHANNEL>>
DEFINE PROJECTED_SALES DECIMAL <TIME PROD.GEOG.CHAN <PRODUCT GEOGRAPHY CHANNEL>>
```

これらの変数は同じ次元性を持つため、1 つの AGGREGATE コマンドを使用して、3 つすべての変数のデータを集計できます。

sales.agg という名前の集計マップを定義したと想定すると、次のコマンドを使用して、3 つすべての変数のデータを集計できます。

```
AGGREGATE sales units projected_sales USING sales.agg
```

例 12-7 1 つのコマンドを使用して集計できない変数

アナリティック・ワークスペースに次の名前付き複合ディメンションおよび 3 つの変数の定義が含まれると想定します。

```
DEFINE PROD.GEOG.CHAN COMPOSITE <PRODUCT, GEOGRAPHY, CHANNEL>

DEFINE SALES DECIMAL <TIME PROD.GEOG.CHAN <PRODUCT, GEOGRAPHY, CHANNEL>>
```

```
DEFINE UNITS INTEGER <TIME SPARSE <PRODUCT, GEOGRAPHY, CHANNEL>>
DEFINE PROJECTED_SALES DECIMAL <TIME SPARSE <PRODUCT, GEOGRAPHY>>
```

各変数を比較し、それらの次元性が異なることを次に示します。

- sales 変数は名前付き複合ディメンション prod.geog.chan を使用しており、そのベース・ディメンションは product、geography および channel です。
- units 変数は名前なし複合ディメンションを使用しており、そのベース・ディメンションは product、geography および channel です。この名前なし複合ディメンションは名前付き複合ディメンションと同じディメンションを同じ順序で持ちますが、Oracle OLAP では、名前付き複合ディメンションと名前なし複合ディメンションは 2 つの異なるアナリティック・ワークスペース・オブジェクトとみなされます。そのため、sales および units は、同じ次元性を持ちません。
- project_sales 変数も名前なし複合ディメンションを使用しており、そのベース・ディメンションは product および geography です。ただし、この名前なし複合ディメンションには channel ディメンションが含まれないため、units 変数が使用する名前なし複合ディメンションとは異なります。

各変数の次元性が異なるため、各変数用の異なる集計マップを定義して、データを集計する必要があります。そのため、各変数に対して異なる AGGREGATE コマンドを使用する必要があります。

RELATION コマンドの概要

RELATION コマンドの基本構文は次のとおりです。

```
RELATION parent-rel [PRECOMPUTE (limit-phrase)] [OPERATOR opvar]
```

集計マップには、変数定義内の各階層ディメンションに対して 1 つの RELATION コマンドが必要です。パフォーマンスを最適化するには、変数定義に示されている順序と同じ順序で RELATION コマンドをリストします。この順序によってデータの格納方法が指定され、最も速く変化するディメンションから順に格納されます（3-16 ページの「変数データを格納する方法」を参照）。データを集計する際は、最も速く変化するディメンションを最初に集計し、最も遅く変化するディメンションを最後に集計すると効率的です。

たとえば、次のように、sales 変数が time および prod.geog.chan 複合ディメンションによってディメンション化されていると想定します。

```
<time prod.geog.chan <product, geography, channel>>
```

この場合、最初の RELATION コマンドは time 用、2 番目は product 用、3 番目は geography 用、4 番目は channel 用にする必要があります。

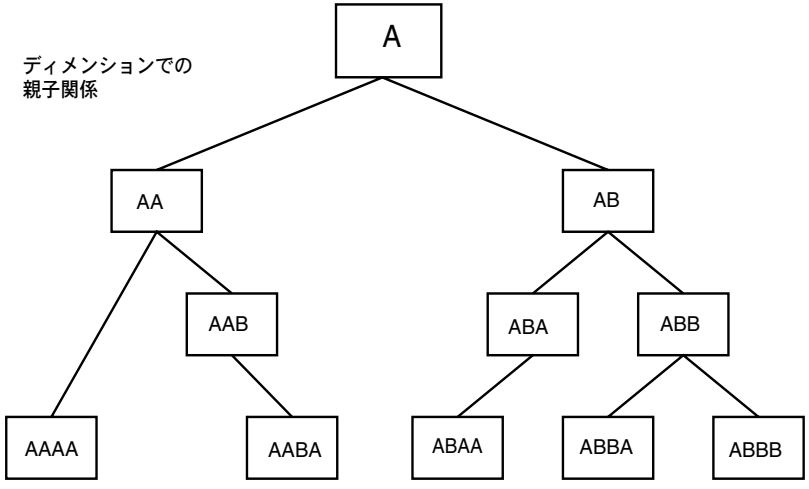
例 12-8 SUM または MAX を使用した集計

次の例では、letter ディメンション、letter.letter 親リレーションおよび units 変数が使用されます。

```
LETTER          LETTER.LETTER    UNITS
--------------- ------------- -------------
a               NA                       NA
aa              a                        NA
ab              a                        NA
aab             aa                       NA
aba             ab                       NA
abb             ab                       NA
aaaa            aa                        1
aaba            aab                       2
abaa            aba                       1
abbb            abb                       1
abba            abb                       1
```

次の図に、letter.letter によって定義されるリレーションを示します。

図 12-1 LETTER ディメンションでの親子関係



LETTER.AGGMAP では、SUM を使用して aa の値が計算されます。

```
DEFINE LETTER.AGGMAP AGGMAP
AGGMAP
RELATION letter.letter PRECOMPUTE ('aa')
END
```

次に示すとおり、データを集計すると、aa の値は 3 になります。

```
aa = (aab + aaaa) = (aaba + aaaa) = (2 + 1) = 3
```

aab は aaba の親で aa の子ですが、この計算の結果としてその値が格納されないことに注意してください。

集計メソッドの指定

各ディメンションに対する集計メソッドは、RELATION コマンドに指定します。デフォルトの集計メソッドは SUM で、子セルの値を足し、合計を親セルに格納します。次に示す他の集計メソッドを使用することもできます。

合計（SUM）
スケール合計（SSUM）
加重合計（WSUM）
平均（AVERAGE）
階層平均（HAVERAGE）
加重平均（WAVERAGE）
階層加重平均（HWAVERAGE）
最大（MAX）
最小（MIN）
最初（FIRST）
階層の最初（HFIRST）
最後（LAST）
階層の最後（HLAST）
積（AND）
和（OR）

これらの集計メソッドは、RELATION コマンドの引数です。これらのメソッドの詳細は、Oracle9*i* OLAP DML Reference ヘルプの RELATION コマンドの項を参照してください。RELATION 集計メソッドを DML 集計ファンクションと混同しないでください。

例 12-9 集計メソッドの指定

次の RELATION コマンドでは、OPERATOR キーワードによって、集計メソッドがデフォルトの SUM から MAX に変更されます。

```
RELATION letter.letter PRECOMPUTE ('aa') OPERATOR MAX
```

変更した集計マップを使用してデータを集計すると、aa の値は 2 になります。これは、aa に提供されている最大値が 2 であるためです（図 12-1「LETTER ディメンションでの親子関係」を参照）。

例 12-10 加重変数の使用

WSUM、WAVERAGE および HWAVERAGE 集計メソッドでは、加重変数が使用されます。最初に加重変数を定義し、次に、ARGS WEIGHTBY 引数を使用して RELATION コマンドにその変数を指定する必要があります。

次の集計マップでは、変数 letter.weights で定義された加重を使用して aa の値が計算されます。

```
DEFINE LETTER.AGGMAP AGGMAP
AGGMAP
RELATION letter.letter PRECOMPUTE ('aa') OPERATOR WSUM -
     ARGS WEIGHTBY letter.weights
END
```

次の REPORT コマンドの出力に、集計が表示されます。

```
report down letter letter.weights units


LETTER          LETTER.LETTER  LETTER.WEIGHTS      UNITS
--------------- -------------- --------------- ---------------
a               NA                          NA              NA
aa              a                           NA               7
ab              a                           NA              NA
aab             aa                          NA              NA
aba             ab                          NA              NA
abb             ab                          NA              NA
aaaa            aa                           5               1
aaba            aab                         NA               2
abaa            aba                         NA               1
abbb            abb                         NA               1
abba            abb                         NA               1
```

units 変数の aa の値は、次のように計算されます。

```
aa = ((5 * aaaa) + aab) = ((5*aaaa) + aaba) = (5*1) + 2 = 7
```

集計用のデータの選択

PRECOMPUTE 句は、AGGREGATE コマンドによって集計されるデータを制限します。最も単純な形式の PRECOMPUTE 句は、LIMIT *dimension* TO コマンドと同様に機能します。デフォルトの制限がディメンションに対して設定されていることに注意してください。これは、RELATION コマンドで明示的に指定されていません。

たとえば、次の LIMIT コマンドを実行すると、product ディメンションの AUDIODIV、VIDEODIV および ACCDIV の値が選択されます。

```
limit product to 'audiodiv' 'videodiv' 'accdiv'
```

これと同様の RELATION コマンドは次のとおりです。

```
RELATION product.parentrel PRECOMPUTE ('AUDIODIV' 'VIDEODIV' 'ACCDIV')
```

これらすべての値は product の STANDARD 階層の同じレベル（L2）に存在するため、次の LIMIT コマンドを実行すると、同じ結果が生成されます。

```
limit product to product.levelrel 'L2'
```

これと同様の RELATION コマンドは次のとおりです。

```
RELATION product.parentrel PRECOMPUTE (product.levelrel 'L2')
```

TO 句を使用しても目的の結果を得られない場合もあります。その他の選択句（KEEP、REMOVE、COMPLEMENT など）を使用する場合、次に示すとおり、LIMIT ファンクションを明示的にコールする必要があります。

```
RELATION product.parentrel PRECOMPUTE (limit(product complement 'TOTALPROD'))
```

例 12-11 PRECOMPUTE 句を使用した集計マップ

この集計マップでは、PRECOMPUTE 句を使用して、AGGREGATE コマンドによって集計されるデータを制限します。

```
DEFINE GPCT.AGGMAP AGGMAP
LD Aggregation map for sales, units, quota, costs
AGGMAP
RELATION geography.parentrel PRECOMPUTE (geography.levelrel 'L3')
RELATION product.parentrel PRECOMPUTE (limit(product complement 'TOTALPROD'))
RELATION channel.parentrel
RELATION time.parentrel PRECOMPUTE (time ne '2001')
END
```

実行時の集計のキャッシュ

集計マップ内の CACHE コマンドは、その場で計算されるデータをセッション期間中に使用可能にするかどうかを決定します。デフォルトでは、データは、問合せされるたびに再計算する必要があります。データをキャッシュし、後続の問合せでは単にそのデータを取得するようにすると、問合せ時間が短縮されます。ただし、キャッシュの保持によって、問題が発生する場合があります。

ユーザーがセッション中にデータを変更した場合（予測や what-if 分析の実行時など）、それ以前に集計したデータには、データの変更が反映されません。このようにしてデータの同期が失われると、ユーザーが不適切なデータを参照することを意味します。ユーザーがセッション中にデータを変更する場合、キャッシュを保持しないでください。

ユーザーがアナリティック・ワークスペースへの書込み権限を持っている場合、このユーザーが UPDATE および COMMIT コマンドを発行すると、実行時の計算は他の変更とともに保

存されます。これによって、記憶域を節約するという、実行時の集計の目的は達成できなくなります。

ユーザーがアナリティック・ワークスペースを保存できる場合、CACHE SESSION コマンドを使用してキャッシュを作成します。ユーザーがアナリティック・ワークスペースを保存できない場合、CACHE SESSION または CACHE STORE のいずれかを使用できます。

キャッシュの効果は、V$AW_CALC 動的パフォーマンス・ビューで追跡されます。このビューを問い合せる方法の詳細は、『Oracle9*i* OLAP ユーザーズ・ガイド』を参照してください。

非階層データの集計

明細項目などの一部のディメンションは、階層構造を持ちません。かわりに、個々の明細項目が、(場合によっては複雑なフォーミュラを使用して) 1 つ以上の他の明細項目またはアナリティック・ワークスペース・オブジェクトから計算されます。このようなタイプのディメンション上のデータを解決するには、モデルが必要です。

モデルを実行するには、aggmap 内に MODEL コマンドを含めます。このコマンドの基本構文は次のとおりです。

```
MODEL modelname [PRECOMPUTE ALL|NA]
```

modelname は、集計マップの 1 つ以上のディメンションの値を計算する、既存の MODEL オブジェクトの名前です。

PRECOMPUTE ALL に指定すると、AGGREGATE コマンドでデータのメンテナンス手順としてモデルが実行されます。集計マップ内で AGGREGATE コマンドより前に示されるすべての RELATION または MODEL コマンドも、PRECOMPUTE ALL に指定する必要があります。ただし、集計マップ内で AGGREGATE コマンドの後に示されるすべての RELATION または MODEL コマンドは、PRECOMPUTE ALL または PRECOMPUTE NA の両方に指定できます。

PRECOMPUTE NA に指定すると、AGGREGATE ファンクションで実行時にモデルが実行されます。実行時にモデルを実行するには、次の条件を満たす必要があります。

- aggmap 内のすべての RELATION コマンドが、PRECOMPUTE NA に指定されている MODEL コマンドの前に示されている。
- その後のすべての追加 MODEL コマンドも、PRECOMPUTE NA に指定されている。
- モデルは、連立方程式または時系列 (LEAD や LAG ファンクションなど) を解決できない。
- モデルは、AGGREGATE ファンクションをコールするオブジェクトを参照できない。たとえば、モデルには、TAX=PROFIT*RATE (RATE は変数またはフォーミュラ) などの等式が含まれている場合がありますが、RATE は実行時に集計できません。

参照： 第 8 章「モデルの使用」を参照してください。

例 12-12 集計でのモデルの解決

この例では、budget 変数が使用されます。

```
DEFINE BUDGET VARIABLE DECIMAL <LINE TIME>
LD Budgeted $ Financial
```

time ディメンションは、2 つの階層（STANDARD および YTD）および time.parentrel という名前の親リレーションを持ちます。

```
                -----TIME.PARENTREL------
                ----TIME.HIERARCHIES-----
TIME              STANDARD       YTD
--------------- ------------ ------------
LAST.YTD        NA           NA
CURRENT.YTD     NA           NA
JAN01           Q1.01        LAST.YTD
FEB01           Q1.01        LAST.YTD
MAR01           Q1.01        LAST.YTD
APR01           Q2.01        LAST.YTD
MAY01           Q2.01        LAST.YTD
JUN01           Q2.01        LAST.YTD
JUL01           Q3.01        LAST.YTD
AUG01           Q3.01        LAST.YTD
SEP01           Q3.01        LAST.YTD
OCT01           Q4.01        LAST.YTD
NOV01           Q4.01        LAST.YTD
DEC01           Q4.01        LAST.YTD
JAN02           Q1.02        CURRENT.YTD
FEB02           Q1.02        CURRENT.YTD
MAR02           Q1.02        CURRENT.YTD
APR02           Q2.02        CURRENT.YTD
MAY02           Q2.02        CURRENT.YTD
Q1.01           2001         NA
Q2.01           2001         NA
Q3.01           2001         NA
Q4.01           2001         NA
Q1.02           2002         NA
Q2.02           2002         NA
2001            NA           NA
2002            NA           NA
```

明細項目間の関係は、次のモデルで定義されています。

```
DEFINE INCOME.BUDGET MODEL
MODEL
dimension line time
opr.income = gross.margin - marketing
gross.margin = revenue - cogs
```

```
revenue = lag(revenue, 12, time) * 1.02
cogs = lag(cogs, 1, time) * 1.01
marketing = lag(opr.income, 1, time) * 0.20
END
```

次の集計マップは、すべてのデータを事前集計します。モデルには LAG ファンクションと連立方程式の両方が含まれているため、すべてのデータを事前集計する必要があることに注意してください。

```
DEFINE BUDGET.AGGMAP1 AGGMAP
AGGMAP
MODEL income.budget
RELATION time.parentrel
END
```

事前計算されるデータを生成する方法

通常、事前計算される集計は、データベース内のデータのメンテナンスの一部として、バッチ・ウィンドウで生成します。必要に応じて、Job Manager を使用して Oracle Enterprise Manager でバッチ・ジョブをスケジュールできます（『Oracle9*i* OLAP ユーザーズ・ガイド』を参照）。

AGGREGATE コマンドを実行すると、集計マップで提供される指定に応じて、1 つ以上の変数のデータが集計されます。AGGREGATE コマンドの基本構文は次のとおりです。

```
AGGREGATE variables USING aggmap
```

variables は 1 つ以上の変数の名前です。

aggmap は集計マップの名前です。

例 12-13 バッチ・ジョブでのデータの事前計算

バッチ・ジョブには、次のようなコマンドを含める必要があります。

```
ALLSTAT
POUTFILEUNIT=FILEOPEN('userfiles/progress.txt' WRITE)
AGGREGATE sales units USING gpct.aggmap
UPDATE
COMMIT
FILECLOSE POUTFILEUNIT
```

ディメンションのステータスの影響

RELATION コマンドを実行すると、ステータス内のソース・データ値（アナリティック・ワークスペースにロードされ、集計の基礎として使用される値）のみが集計されます。親値は、ステータス内であるかどうかにかかわらず計算されます。たとえば、time ディメン

ションで JAN01、FEB01 および MAR01 のみがステータス内である場合、Q1.01 は計算されますが、他の四半期は計算されません。また、2001 は、Q1.01 のみを入力として使用して（他の四半期は NA であるため）計算されますが、他の年は計算されません。

これは、アナリティック・ワークスペース内の新しいデータのみを集計する場合に有効です。ただし、これは慎重に行う必要があります（12-24 ページの「部分集計の実行」を参照）。

処理過程の監視

POUTFILEUNIT オプションを設定すると、集計の処理過程を監視できます。OUTFILEUNIT オプションまたは OUTFILE ファンクションを使用して、POUTFILEUNIT の値を設定できます。

次のコマンドを実行すると、POUTFILEUNIT が現行の送信ファイルのファイル・ユニット番号（通常は画面）に設定されます。

```
POUTFILEUNIT=OUTFILEUNIT
```

次のコマンドを実行すると、userfiles ディレクトリ別名の progress.txt という名前のファイルが開かれ、POUTFILEUNIT が progress.txt のファイル・ユニット番号に設定されます。

```
POUTFILEUNIT=FILEOPEN('userfiles/progress.txt' WRITE)
```

集計の完了後、FILECLOSE コマンドを使用してそのファイルを閉じる必要があります。

実行時にデータを計算する方法

AGGREGATE ファンクションを実行すると、RELATION コマンドの PRECOMPUTE 句に指定されているデータ以外のデータが計算されます。現在ステータス内の値が戻されます。

たとえば、次の RELATION コマンドを含む集計マップを使用していると想定します。

```
RELATION letter.letter PRECOMPUTE ('aa')
```

この場合、AGGREGATE ファンクションは、次に示すとおり、aa を除くすべての集計を計算します。

```
REPORT AGGREGATE(units USING letter.aggmap)

                  AGGREGATE(UNITS
LETTER          USING LETTER.AGGMAP)
--------------- --------------------
a                                  3
aa                                NA
ab                                 3
aab                                2
```

```
aba                             1
abb                             2
aaaa                            1
aaba                            2
abaa                            1
abbb                            1
abba                            1
```

その場での計算の設定

一部のデータをその場で計算する場合、次の手順を実行する必要があります。

1. 事前計算するデータおよびその場で計算するデータを決定します。
2. 1 つ以上の RELATION または MODEL コマンドに PRECOMPUTE キーワードを含む集計マップを定義します。また、デフォルト値が不適切な場合、CACHE コマンドを含める必要があります。
3. 集計マップで AGGREGATE コマンドを使用し、ディスクに格納するデータを事前計算します。
4. AGGREGATE コマンドの実行後、集計マップをコンパイルします（12-9 ページの「集計マップをコンパイルする方法」を参照）。
5. 集計マップを使用する変数に $NATRIGGER プロパティを追加して、問合せされたデータ内の NA によって AGGREGATE ファンクションが実行されるようにします。

変数への $NATRIGGER プロパティの追加

集計データを戻すすべてのコマンドに AGGREGATE ファンクションを指定するかわりに、変数にプロパティを追加して、AGGREGATE ファンクションを自動的に実行できます。

変数に $NATRIGGER プロパティを設定すると、問合せされたデータ内の NA 値によって特定の動作が実行されます。集計をトリガーするには、AGGREGATE ファンクションへのコールを $NATRIGGER プロパティの値として割り当てます。

次のコマンドを実行すると、sales 変数に $NATRIGGER プロパティが追加され、未解決のデータが sales.aggmap 集計マップを使用して集計されます。

```
CONSIDER sales
PROPERTY '$NATRIGGER' 'AGGREGATE(sales USING sales.aggmap)'
```

カスタム集計の作成

ほとんどの集計は、ディメンション内の親子関係を識別する親リレーションとともに定義されます。ただし、ユーザーは、予測や what-if 分析などのために、または想定外の方法でデータを参照するために、実行時に独自の集計を作成する場合があります。カスタム集計を作成するプロセスを次に示します。

1. カスタム集計のディメンション値を作成します。次のコマンドを実行すると、letter ディメンションに「bb」が追加されます。

   ```
   maintain letter add 'bb'
   ```

2. AGGREGATION ファンクションを含む MODEL オブジェクトを作成し、子ディメンション値に新しいディメンション値を関連付けます。次のモデルは、bb を aab および aba の親として識別します。親ディメンション値（この場合は bb）は、まだ親リレーション（letter.letter）で親として定義できないことに注意してください。

   ```
   DEFINE LETTER.MODEL MODEL
   MODEL
   DIMENSION letter
   bb=AGGREGATION('aab' 'aba')
   ```

3. AGGMAP ADD コマンドを実行し、モデルを既存の AGGMAP オブジェクトに追加します。

   ```
   AGGMAP ADD letter.model TO letter.aggmap
   ```

   これで、例 12-8 の集計マップは次のようになります。

   ```
   DEFINE LETTER.AGGMAP AGGMAP
   AGGMAP
   RELATION letter.letter PRECOMPUTE ('aa')
   END
   AGGMAP ADD letter.model
   ```

4. モデルは、次に示す AGGREGATE ファンクションによってのみ実行されます。AGGREGATE コマンドは、モデルを無視します。

   ```
   REPORT AGGREGATE(units USING letter.aggmap)
   ```

5. セッション中に集計マップからモデルを削除する場合、AGGMAP REMOVE コマンドを使用します。

   重要： AGGMAP ADD コマンドは、セッションの終了時に、aggmap オブジェクトから自動的に削除されます。

事前計算される集計と実行時の集計のバランス

AGGREGATE を使用すると、次のすべての方法で集計を実行できます。

- すべてのデータを事前集計する。変数のすべてのデータを集計し、データベースに格納します。通常、構築は比較的低速で実行され、ユーザーの問合せは非常に高速で実行されます。

- すべてのデータをその場で（実行時に）計算する。この場合、構築プロセスから集計が排除され、構築は非常に高速に実行されます（アナリティック・ワークスペースにデータをロードする時間のみに短縮されます）。ただし、ユーザーの問合せは非常に低速で実行されます。
- 一部のデータを事前集計し、残りのデータをその場で計算する。

パフォーマンスを向上するには、妥協が必要です。そのため、全体的なパフォーマンスを向上するための最も有効な手順の 1 つは、集計してアナリティック・ワークスペースに格納するデータの量と、その場で計算するように指定するデータの量のバランスをとることです。

代表的な方法として、**スキップ・レベル集計**があります。この方法では、変数の 1 つまたは 2 つのディメンションを選択し、これらのディメンションの階層を 1 レベルおきに事前集計します。ユーザーが最も頻繁に問い合せるレベルがわかっている場合、それらのレベルのデータを事前計算する必要があります。

例 12-14 スキップ・レベル方法を使用したデータの計算

sales データを集計すると想定します。sales 変数は、geography、product、channel および time によってディメンション化されています。

最初に、各ディメンションの階層について考えます。各階層に存在するレベルの数、ユーザーが通常問い合せるデータのレベル、および（新しいアナリティック・ワークスペースを設計している場合は）ユーザーが問い合せる予定のデータのレベルについて考えます。

ユーザーが time 階層の sales データに問合せを行う方法の傾向に関して、次の情報を得たと想定します。

time のレベル名	レベルの説明的な名前	ディメンション値の例	このレベルに対するユーザーの問合せの頻度
L1	Year	YEAR99、YEAR00	高い
L2	Quarter	Q3.99、Q3.99、Q1.00	高い
L3	Month	JAN99、DEC00	高い

ユーザーが geography 階層の sales データに問合せを行う方法の傾向に関して、次の情報を得たと想定します。

geography のレベル名	レベルの説明的な名前	ディメンション値の例	このレベルに対するユーザーの問合せの頻度
L1	World	WORLD	高い
L2	Continent	EUROPE、AMERICAS	低い

geography のレベル名	レベルの説明的な名前	ディメンション値の例	このレベルに対するユーザーの問合せの頻度
L3	Country	HUNGARY、SPAIN	高い
L4	City	BUDAPEST、MADRID	高い

ユーザーが product ディメンション階層の sales データに問合せを行う方法の傾向に関して、次の情報を得たと想定します。

product のレベル名	レベルの説明的な名前	ディメンション値の例	このレベルに対するユーザーの問合せの頻度
L1	All Products	TOTALPROD	高い
L2	Division	AUDIODIV、VIDEODIV	高い
L3	Category	TV、VCR	高い
L4	Product	TUNER、CDPLAYER	高い

ユーザーによるデータの問合せ方法に関するこれらの情報を使用して、次の方法で集計を実行します。

- time および product では、すべてのレベルが頻繁に問合せされているため、全体を集計します。
- geography ディメンションでは、L1（World）および L3（Country）が頻繁に問合せされているため、これらのレベルのデータを集計します。これに対して、L2 は頻繁に問合せされていないため、その場で計算できます。

最下位レベルのデータがアナリティック・ワークスペースにロードされ、このソース・データから集計データが計算されます。

その結果、集計マップの内容は、次のようになります。

```
RELATION time.parentrel
RELATION geography.parentrel PRECOMPUTE (geog.leveldim 'L3' 'L1')
RELATION product.parentrel
```

実行時の計算のためのディメンションの選択

スキップ・レベルの方法は、1 つまたは 2 つのディメンションに対してのみ使用します。スキップ・レベルの方法は、変数定義内の半数以下のディメンションに対して使用する必要があります。たとえば、3 つのディメンションが存在する場合、1 つのディメンションに対してスキップ・レベルの方法を使用できます。4 つ以上のディメンションが存在する場合、2 つのディメンションに対してスキップ・レベルの方法を使用できます。

スキップ・レベル集計の使用が最も有効なディメンションは、多くのレベルが存在する階層のディメンションです。

可能な場合、変数定義内で、最も速く変化するディメンションまたは中間で変化するディメンションのいずれかを選択します。その場で行う計算のパフォーマンスは、このようなディメンションで常に最適化されます。

実行時の計算のためのレベルの選択

ディメンション階層のレベルを、1 レベルおきにスキップします。連続したレベルを 3 つ以上スキップすることは避けます。たとえば、階層に 7 つのレベルが存在する場合、L2、L4 および L6 をスキップします。これは、L1、L3 および L5 を事前計算することを意味します。(ディテール・データのレベルは L7 です。) レベルに対する問合せの頻度について考えます (例 12-14 を参照)。パフォーマンスを最適化するには、最も頻繁に問合せされるデータを事前集計し、頻繁に要求されないデータをその場で集計します。

連続したレベルをスキップしないでください。たとえば、L2、L3、L4 および L5 をスキップした場合、L2 のデータを問い合せると、AGGREGATE によって L5 が計算され、次にそのデータから L4、L3、最後に L2 と、順次集計を行う必要があります。これに対して、L2、L4 および L6 をスキップした場合、L2 のデータを問い合せると、AGGREGATE によって L3 のデータのみの集計が必要になります。

このルールの例外は、各レベルでそれぞれの親に存在する子が非常に少ない場合です。スキップすることを検討している連続した各レベルでこの条件が満たされている場合、2 つ以上の連続したレベルをスキップできます。

部分集計の実行

通常、アナリティック・ワークスペースのメンテナンスは、限定されたバッチ・ウィンドウ内で実行する必要があります。このため、多くの DBA は、データが更新されるたびに、全体集計ではなく、部分集計を実行します。すべてのデータを事前集計する場合、この方法を使用しても問題は発生しません。ただし、事前集計と実行時の集計の両方を使用するデータに対して部分集計を実行する場合、正しい結果を得るには、いくつかの手順を実行する必要があります。データのエラーは、PRECOMPUTE キーワードによって生成されるステータス・リストが古い場合に発生します。

PRECOMPUTE 句によって生成されるステータス・リストの機能は次のとおりです。

- AGGREGATE コマンドに対する、事前計算する必要があるデータの通知

- AGGREGATE ファンクションに対する、AGGREGATE コマンドが行った動作の通知

AGGREGATE コマンドを AGGREGATE ファンクションとともに使用しない場合は、次の項を読む必要はありません。

問題が発生する集計の変更

次の操作を実行する場合は、この項を読む必要があります。

- 増分データのロード : アナリティック・ワークスペースを構築済で、新しいデータを定期的にロードします。集計マップで、RELATION コマンドの 1 つ以上の PRECOMPUTE 句に対して変更を加えています。
- データに依存した PRECOMPUTE 句の使用 : 集計マップで PRECOMPUTE キーワードを使用する場合、その PRECOMPUTE 句が、単にディメンション値またはレベルを識別するのではなく、データに依存する場合があります。
- 階層の変更 : データの集計後にディメンションの階層を変更した場合、すべてのデータを再集計する必要があります。アナリティック・ワークスペース内のデータを再集計する時間を短縮可能な手順を実行できる場合があります。

増分データのロード

増分データのロードとは、既存のアナリティック・ワークスペースに新しい入力データをロードして、そのデータを集計するプロセスです。通常、このプロセスは定期的に（毎月、毎週、毎日など）行います。

たとえば、新しいアナリティック・ワークスペースを設計すると想定します。このアナリティック・ワークスペースには、sales および units の 2 つの変数が含まれます。最初にアナリティック・ワークスペースを構築する際に、両方の変数に対して 1 年分のデータを入力したと想定します。sales および units の定義には同じディメンションが同じ順序で含まれているため、sales と units の両方によって共有される 1 つの集計マップを定義します。アナリティック・ワークスペースに入力データをロードし、次に AGGREGATE コマンドを使用してその入力データをロールアップします。

毎月の初日に sales および units の新しい入力データを取得することがわかっています。たとえば、3 月 1 日には、前月（2 月）の sales および units のデータを受け取ります。既存のアナリティック・ワークスペースに 2 月のデータをロードし、その入力データを集計する必要があります。これが、増分データのロードです。次の増分データのロードは 4 月 1 日に実行され、それ以降も同様です。

通常、この新しいデータを集計する場合、LIMIT コマンドを使用して、新しい入力データのみが集計されるようにします。たとえば、ロードした 2 月分の新しい入力データのみを集計するには、次のコマンドを使用します。

```
LIMIT month TO 'FEB99'
AGGREGATE sales units USING salesunits.aggmap
```

これは、集計マップのすべての PRECOMPUTE 句を変更していない場合に使用できます。変更した場合、すべてのデータを事前集計する必要があります。

問題 : PRECOMPUTE ステータス・リストが正しくない

PRECOMPUTE 句を変更すると、ステータス・リストが変更されます。これは、PRECOMPUTE 句の変更後に AGGREGATE コマンドによって生成されるデータが正しい場合でも、Oracle OLAP が、AGGREGATE ファンクションを使用してユーザーから要求されたデータを戻すことができない場合があることを意味します。ステータス・リストで、実際には計算されていない値が計算済として示されている場合があります。

処置 : PRECOMPUTE ステータス・リストを再生成する

集計マップに含まれる 1 つ以上の RELATION コマンドのいずれかの PRECOMPUTE 句を変更した場合、すべてのデータを事前集計する必要があります。これを行わない場合、データとの同期が失われた PRECOMPUTE ステータス・リストが AGGREGATE ファンクションによって使用されるため、一部の必要な値が生成されない場合があります。

データを適切に集計するには、次の手順を実行します。

1. 集計マップの PRECOMPUTE 句に対するすべての変更を行ったことを確認します。
2. 増分入力データをロードします。
3. すべてのディメンションの現在のステータスを ALL に設定します。(集計マップ内のすべてのディメンションに対して、1 つの ALLSTAT コマンドまたは LIMIT TO ALL コマンドを使用できます。)
4. AGGREGATE コマンドを実行します。
5. 集計マップを再コンパイルします。(または、手順 4 で AGGREGATE コマンドを実行する際に、FUNCDATA キーワードを使用することもできます。)

データに依存した PRECOMPUTE 句の使用

PRECOMPUTE キーワードには、LIMIT コマンドなどが続きます。この制限式は、データの値を使用して柔軟に指定できます。たとえば、ある時間間隔内で売上高が最も低い 5 つの地域を指定できます。RELATION コマンドは、次のようになります。

```
RELATION geography.parentrel PRECOMPUTE (BOTTOM 5 BASEDON sales)
```

問題 : データを更新するたびに制限句の値が変化する

データに依存した制限式では、結果が変化する場合があります。2 月に構築したアナリティック・ワークスペースの「売上高が最も低い 5 つ」の地域は、3 月に増分データのロードを実行した後の「売上高が最も低い 5 つ」の地域とは異なる場合があります。また、3 月の「売上高が最も低い 5 つ」の地域は、4 月に増分データのロードを実行した後の「売上高が最も低い 5 つ」の地域とは異なる場合があります。

この場合、PRECOMPUTE ステータス・リストの同期は失われています。ある値が事前計算されていることがステータス・リストに示されているために、AGGREGATE ファンクションによって必要な値が計算されない場合があります。

処置 : 値セットを保持する

データに依存した PRECOMPUTE 句を使用するかわりに、次のいずれかの操作を実行できます。

- 特定の値の指定
- 特定の値を格納する値セットの作成

これらの方法のいずれかを使用すると、時間の経過に伴って増分データのロードおよび集計を行っても、PRECOMPUTE キーワードによって生成されるステータス・リストは変化しません。ただし、制限式内の 5 つの店または値セットは、売上高が最も低い店を表しているかどうかにかかわらず、変化しません。

制限句を最新の状態に保つには、次の手順を実行します。

1. 新しいデータをロードするたびに、制限式を再計算します。
2. 計算結果が異なる場合、値セットを変更します。
3. 影響を受ける変数の全体集計を実行します。
4. AGGREGATE ファンクションによって使用される集計マップを再コンパイルします。

従う必要がある一般的なガイドラインは、12-25 ページの「増分データのロード」を参照してください。

入力データまたは階層を変更した場合、NA 値を使用して集計されたすべてのデータを置換します。これを行うには、ディメンションを入力データに制限し、新しい変数を作成して、元の変数から新しい変数にデータをコピーします。次に、元の変数を削除して、新しい変数の名前を元の変数の名前に変更します。

例 12-15 ディメンション値の指定

データに依存した PRECOMPUTE 句を使用するかわりに、PRECOMPUTE 句で特定のディメンション値を使用します。データのロード後、データに依存した LIMIT コマンドを発行して、ディメンション値を識別します。次に、PRECOMPUTE 句にそれらの値を指定します。次に例を示します。

```
LIMIT time TO '2001'
LIMIT channel TO 'TOTALCHANNEL'
LIMIT product TO 'TOTALPROD'
LIMIT geography TO BOTTOM 5 BASED ON sales

STATUS geography
The current status of GEOGRAPHY is:
BOGOTA, BORDEAUX, EDINBURGH, KYOTO, BRUSSELS
```

次に、PRECOMPUTE 句を変更して、これらの地域を指定します。

```
RELATION geography.parentrel PRECOMPUTE ('BOGOTA' 'BORDEAUX' 'EDINBURGH' -
    'KYOTO' 'BRUSSELS')
```

データに依存した PRECOMPUTE 句を使用する場合、PRECOMPUTE 句を使用した値セットを作成して使用します。

例 12-16 値セットの使用

値セットは、値のリストを格納するために使用できます。たとえば、次のコマンドを実行すると、geography ディメンション用の値セットが作成されます。増分更新の実行後、値セットを更新する必要がありますが、集計マップを編集する必要はありません。

次のコマンドを実行すると、geography ディメンション用の値セットが作成されます。

```
DEFINE lowsales.geog VALUESET geography
LIMIT time TO '2001'
LIMIT channel TO 'TOTALCHANNEL'
LIMIT product TO 'TOTALPROD'
LIMIT lowsales.geog TO BOTTOM 5 BASED ON sales
```

VALUES ファンクションによって、この値セットの次のステータス・リストが戻されます。

```
SHOW VALUES(lowsales.geog)

BOGOTA
BORDEAUX
EDINBURGH
BRUSSELS
KYOTO
```

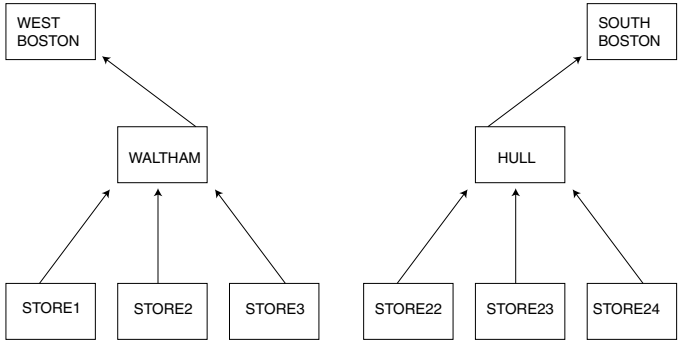
次の RELATION コマンドでは、この値セットが使用されます。

```
RELATION geography.parentrel PRECOMPUTE (lowsales.geog)
```

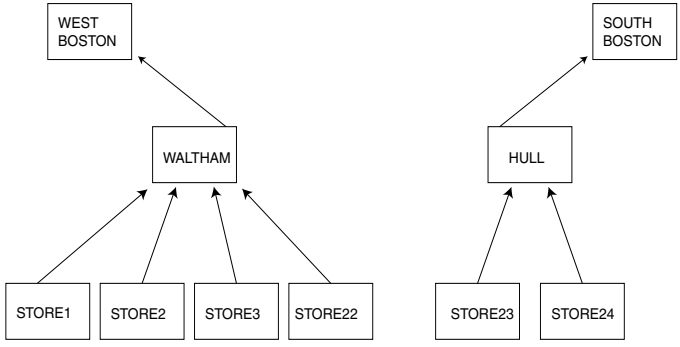
階層の変更

一度階層を定義してデータを集計すると、1 つ以上のディメンション値を階層内の異なる親に移動した場合、階層が変更されます。

たとえば、geography 階層が店の入力データを含むと想定します。店のデータは、都市にロールアップします。都市は、地域にロールアップします（以後、同様に続きます）。



ディメンションおよび変数を定義し、ディメンションの階層を定義します。データをロードし、ロールアップします。数か月後、増分データをロードおよびロールアップした後、店の 1 つが移転します。たとえば、STORE22 は、Massachusetts 州 Hull の店舗を閉鎖し、新しい場所である Massachusetts 州 Waltham で再度開店します。そのため、STORE22 は、SOUTH BOSTON 地域ではなく、WEST BOSTON 地域に含まれます。



そのため、STORE22 のディメンション値を移動して、そのデータを上位階層の異なるディメンション値にロールアップさせる必要があります。たとえば、HULL パスから WALTHAM パスに STORE22 を移動する必要があります。

階層内の異なるパスにデータがロールアップするように 1 つ以上のディメンション値を移動すると、階層が変更されます。

問題 : 以前集計したデータが正しくない

STORE22 の最新月分のデータを受け取ると想定します。データをロードし、集計します。

その後、その店が別の地域内の新しい都市に先月移転していたことが判明します。これは、STORE22 のデータは HULL に集計済ですが、WALTHAM に集計する必要があることを意味します。

問題は、階層を変更するのみでなく、STORE22 のデータが HULL ではなく WALTHAM に集計されるようにデータを修正する必要があるということです。

処置 : 変更されたブランチを再集計する

階層を変更する場合、次のいずれかの方法を使用して、(階層の変更後に) アナリティック・ワークスペースのデータを再集計できます。

- 全体集計の実行。階層に重要な変更を行った場合、これが最適な方法です。
- 部分集計の実行。移動したディメンション値およびそのディメンション値の移動前の兄弟に対して集計を行います。この方法は、階層に行った変更がわずかである場合に行うことができます。

部分集計のメリットは、全体集計より短時間で完了することです。これに対して、全体ロールアップのメリットは、確実に正しい結果が得られることです。

そのため、階層内で 1 つまたは 2 つのディメンション値を移動し、アナリティック・ワークスペースのロールアップを実行できる時間が限定されている場合は、部分集計を実行できます。そうでない場合は、全体集計を実行します。

階層のブランチを集計する方法

階層内で移動したディメンション値の以前の親および現在の親のデータを集計するには、次の手順を実行します。

1. 階層内で移動したディメンション値 (のグループ) を識別します。たとえば、STORE22 は、データが HULL ではなく WALTHAM に集計されるディメンション値です。
2. 移動したディメンション値の以前の兄弟を識別します。(複数のディメンションが移動した場合、それぞれの兄弟を識別する必要があります。) たとえば、STORE22 は以前 STORE23 および STORE24 とグループ化されていたため、このいずれかが STORE22 の以前の兄弟に該当します。
3. ディメンションの現在のステータスを、移動したディメンション値およびその以前の兄弟に制限します。たとえば、次のコマンドを使用して、geography ディメンションを STORE22 および STORE23 に制限できます。

   ```
   LIMIT time TO 'STORE22' 'STORE23'
   ```

4. 変数のデータを集計します。たとえば、次のコマンドを使用して、sales 変数を集計します。

```
AGGREGATE sales USING sales.agg
```

移動したディメンション値を識別することによって、新しい祖先（WALTHAM など）を再計算できます。移動したディメンション値の以前の兄弟を識別することによって、以前の祖先（HULL など）を再計算できます。

予測およびプログラムと AGGREGATE の組合せ

（AGGREGATE コマンドに加えて）代替方法を使用して 1 つ以上のディメンションのデータを事前集計する場合、1 つの変数に対して複数の集計マップを使用する必要があります。データを集計するためのこれらの代替方法は、次のとおりです。

- 予測
- OLAP DML プログラム

たとえば、売上変数が、地域、製品、チャネルおよび時間でディメンション化されていると想定します。変数の一部のデータを AGGREGATE で集計し、その変数の他のデータを予測または DML プログラムで集計する場合、すべての集計データが正しいことを確認するには、追加手順を実行する必要があります。

重要： 複数の集計マップで AGGREGATE ファンクションを使用するには、次のことを確認しておく必要があります。

AGGREGATE ファンクションで使用するために集計マップをコンパイルする際に、各 PRECOMPUTE 句によって生成されるステータスが、データが事前計算されるディメンション内のノードを正確に定義している。

このことを確認できない場合、複数の集計マップで AGGREGATE ファンクションを使用しないでください。

複数の集計マップを使用する場合

同じ変数（のグループ）のデータを集計する場合、コマンドおよびファンクションで同じ集計マップを使用することが最適な方法です。ただし、正しい結果を得るために、複数の集計マップをコマンドで使用し、別の集計マップをファンクションで使用する必要がある場合があります。

問題 : 異なる集計マップによって異なるステータス・リストが生成される

複数の集計マップを使用する理由は、各集計マップによって、異なるタスクが実行され、そのため、異なるステータス・リストが生成されることです。

AGGREGATE コマンドおよび AGGREGATE ファンクションが同じ集計マップを使用する場合、問題は発生しません。存在するステータス・リストは 1 つのみであるため、同じステータス・リストが使用されます。

同じ変数（のグループ）に対して複数の集計マップを使用して AGGREGATE コマンドを実行する場合は、問題が発生します。各集計マップによって異なるステータス・リストが生成され、どの集計マップを使用しても、単独では AGGREGATE ファンクションの現在のステータスを正確に識別できない場合があります。

処置 : AGGREGATE ファンクション用の個別の AGGMAP を作成する

複数の集計マップを使用してデータを事前計算する場合、次の手順を実行する必要があります。

1. AGGREGATE ファンクション用の別の集計マップを作成します。この集計マップは、ユーザーの問合せで使用されます。
2. ユーザーの問合せ用の集計マップの内容が、データの事前計算に使用する集計マップの内容を組み合せたものであることを確認します。これを行う方法の例は、例 12-17「複数の集計マップの使用」を参照してください。

例 12-17 複数の集計マップの使用

予測を使用する場合、予測で必要なすべての入力データを事前計算する必要があります。これを行わない場合、予測で不適切または存在しないデータが使用されます。

たとえば、予測ですべての明細項目を集計する必要があると想定します。time、line および division によってディメンション化された budget 変数を使用している場合、通常、line ディメンションの全体集計を実行し、time ディメンションを予測して、その後で残りの division ディメンションを集計します。最初の集計マップ（forecast.agg1）を定義します。この集計マップでは、予測に必要なデータを集計します。この集計マップには、次のコマンドが含まれます。

```
RELATION line.parentrel
```

2 番目の集計マップ（forecast.agg2）を定義します。この集計マップでは、最初の集計マップおよび予測を使用して生成されたデータを集計します。この集計マップには、次のコマンドが含まれます。

```
RELATION division.parentrel PRECOMPUTE ('L3')
```

3 番目の集計マップ（forecast.agg3）を定義します。この集計マップには、前述の 2 つの集計マップの内容が含まれます。

```
RELATION line.parentrel
RELATION division.parentrel PRECOMPUTE ('L3')
```

予測が fore.prg という名前のプログラムに含まれる場合、次のコマンドを使用してデータを集計します。

```
AGGREGATE budget USING forecast.agg1   "Aggregate over LINE
CALL fore.prg                          "Forecast over TIME
AGGREGATE budget USING forecast.agg2   "Aggregate over DIVISION

"Compile the limit map for LINE and DIVISION
COMPILE forecast.agg3

"Use the combined aggregation map for the AGGREGATE function
CONSIDER budget
PROPERTY 'NATRIGGER' 'AGGREGATE(budget USING forecast.agg3)'
```

索引

記号

数字

A

B

L

M

N

O

P

Q

R

S

T

U

V

W

Y

あ

い

う

え

お

か

く

け

こ

さ

し

す

せ

そ

た

ち

て

な

に

は

ひ

ふ

へ

ほ

も

ゆ

ら

り

れ

ろ

わ